Entwicklung eines diagnostischen Verfahrens zur Bestimmung der Gelenkachsen des Sprunggelenks

DISSERTATION

Von der Fakultät für und Wirtschafts- und Sozialwissenschaften der Universität Stuttgart zur Erlangung der Würde eines Doktors der Philosophie (Dr. phil.) genehmigte Abhandlung.

Vorgelegt von

Harald Hochwald

aus Geislingen an der Steige

Hauptberichter:	Prof. Dr. W. W. Alt
Mitberichter:	Prof. Dr.-Ing. J. F. Wagner
Tag der mündlichen Prüfung:	18. Dezember 2006

Institut für Sportwissenschaft der Universität Stuttgart 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis und verwendete Begriffe

Inklination	Winkel zwischen der auf die Sagitalebene projizierten Achse und der Transversalebene
Deviation	Winkel zwischen der auf die Tranversalebene projizierten Achse und der Sagittalebene
Inversion	Bewegung des Fußes um die Achse des unteren Sprunggelenks mit einer Hebung des medialen Fußrandes
Eversion	Bewegung des Fußes um die Achse des unteren Sprunggelenks mit einer Hebung des lateralen Fußrandes
Neutral-Null-Stellung	aufrechter, hüftbreiter Stand des Probanden
Ink.	Inklination
Dev.	Deviation
USG	unteres Sprunggelenk
USGA	untere Sprunggelenkachse
OSG	oberes Sprunggelenk
OSGA	obere Sprunggelenkachse
M.	Musculus
MRT	Magnetresonanztomographie
GUI	General User Interface
STAB	Standardabweichung
ΔE	absoluter Fehler
ΔF	relativer Fehler

Zusammenfassung

Ziel dieser Arbeit war es eine Methode zu entwickeln mit der es möglich ist, die Sprunggelenkachsen in vivo und nicht invasiv zu bestimmen. Ein hypothetischer Zusammenhang zwischen der individuellen Gelenkanatomie und chronischen sowie akuten Überlastungen der Strukturen der unteren Extremitäten wird in der Literatur zwar mehrfach angesprochen, konnte aber von bisher durchgeführten Studien aufgrund von methodischen Restriktionen und dadurch bedingten kleine Fallzahlen weder bestätigt noch verworfen werden.

Folgende Anforderungen mussten von dem neuen diagnostischen Verfahren erfüllt werden: Im Feld einsetzbar (benötigt keine Laborbedingungen), in vivo, nicht invasiv, automatisierte Datenverarbeitung und Datenerfassung, einfach zu handhaben. Um diese Anforderungen zu erfüllen wurde das Ultraschall-Laufzeit-Bewegungsanalyse-System CMS 20 der Firma Zebris® modifiziert und eine eigene Software und die benötigten mathematischen Methoden entwickelt. Wie bisher angewandte nicht invasive Methoden zur Bestimmung der Lage der Sprunggelenkachsen in vivo stützt sich auch dieses Verfahren auf eine Bewegungsanalyse des Sprunggelenkkomplexes. Das eingesetzte mathematische Verfahren betrachtet die Bewegungen um die obere und untere Sprunggelenkachse als eine Rotation in der Ebene und reduziert somit die Bewegungsanalyse auf ein zweidimensionales Problem. Für die Berechnung der Gelenkachsen und damit des COR (Center of Rotation) aus den aufgezeichneten Bewegungsdaten genügen demnach die Koordinatendaten von drei verschiedenen Positionen eines Ultraschallmarkers. Der gesamte Messvorgang dauert weniger als zehn Minuten. Dabei muss in einem ersten Schritt das *Tibiakoordinatensystem* – ein Koordinatensystem markanter anatomischer Punkte – , mithilfe eines speziell für diesen Zweck entwickelten Ultraschalltaststiftes festgelegt werden. In der Neutral-Null-Stellung (hüftbreiter Aufrechter Stand) des Probanden müssen zur Bestimmung des *Tibiakoordinatensystems* zwei Punkte auf der Tibiakante und ein weiterer Punkt auf der Achillessehnenmitte aufgezeichnet werden. Mittels eines vierten Punktes in der Standfläche wird der Ursprung des Koordinatensystems festgelegt. Alle Ergebnisse werden relativ zu diesem *Tibiakoordinatensystem* ausgegeben. Anschließend erfolgt die Aufzeichnung der Bewegungsdaten. Dazu muss der Fuß aus der Neutral-Null-Stellung in die maximale Dorsalflexion bewegt werden. In dieser maximalen Dorsalflexion müssen mehrere Inversions/Eversionbewegungen durchgeführt werden. Die Software berechnet anhand der aufgezeichneten Bewegungskoordinaten alle benötigten Parameter des oberen und unteren Sprunggelenks und stellt diese nach der Messung numerisch und grafisch dar. Für die Achsdaten des unteren Sprunggelenks werden Mittelwerte und Standardabweichung ausgegeben, die aus den berechneten Sprunggelenk-Achsdaten der einzelnen Inversions- und Eversionsbewegungen

gebildet werden. Mithilfe der Standardabweichung können direkt nach einer Messung Aussagen über die Qualität der Bewegungsausführung und damit über die Genauigkeit des Messergebnisses gemacht werden. Das Gesamtsystem wurde einer Reihe von Validitäts- und Reliabilitätstest am mechanischen Modell und in vivo unterzogen. Der ermittelte Fehler der Winkel für die Reliabilitätstests am mechanischen Modell war $\Delta E < 0,9°$. Bei den in vivo Tests war der Fehler der Winkel mit $\Delta E < 2,5°$ etwas höher als bei den Tests am mechanischen Modell, was aufgrund von Hautverschiebungen und Abweichungen bei der willkürlichen Bewegungsausführung zu erwarten war.

In ersten durchgeführten Studien wurden die Sprunggelenkachsen von $n = 97$ Probanden in vivo bestimmt. Die Standardabweichung bzw. der Messfehler der Einzelmessungen von Deviation und Inklination betrug $\Delta E < 5°$. Der Mittelwert des Deviationswinkels der gesamten Stichprobe war $\alpha_{Dev} = 3,4°$ bei einer Standardabweichung von $STAB < 11,4°$. Dieses Messverfahren ist dazu in der Lage, die Achsen des Sprunggelenkkomplexes in vivo und nicht invasiv bei großen Stichproben im Feld zu bestimmen. Der hypothetische Zusammenhang zwischen der individuellen Gelenkanatomie und chronischen und akuten Überlastungen der Strukturen der unteren Extremitäten sollte mit diesem neuen diagnostischen Verfahren aufgeklärt werden können.

Abstract

The aim of this study is to develop a new diagnostic method to determine the talocrural and subtalar ankle joint axis in vivo and non invasive. Dependencies between individual spatial axis orientation and acute or chronic injuries have been discussed widely in current literature. However, because of methodical limitations and small number of subjects today's studies are not able to shed light on these dependencies satisfactorily.
A new diagnostically method has to meet the following specifications: Suitable in field application, in vivo, non invasive, fully automated data acquisition and data processing, easy to use. To meet these specifications the ultrasonic pulse-echo based hardware CMS 20 from Zebris® has been modified. Moreover, a new software and mathematical methods has been developed. According to already existing in vivo procedures for the determination of the ankle joint axis, this method is based on motion analysis of the ankle joint. The mathematical procedure used regards the movement of the talocrulal and subtalar joint as rotations in a plane. Thus, the issue can be reduced to a two dimensions of freedom problem. For the computation of the ankle joint axis and the center of rotation the mathematical method needs three coordinates from a ultrasonic marker. The total measurement takes less than ten minutes. First, a so-called tibia-coordinate-system has to be determined. A custom-built ultrasonic-pointer helps to record prominent anatomical points. To compute the tibia-coordinates two points on the tibia-border and an additional point in the middle of the achilles tendon has to be measured. A fourth point at the base defines the origin of the tibia-coordinates. All points have to be recorded with the subject standing upright in a neutral body position with legs hip-width apart. All results are displayed relative to these tibia-coordinates. The actual recording of ankle movement is done with the subject seated elevated so that the foot can follow its physiological range of movement necessary for the system to compute the ankle joint axis. The subject's foot has to move out of the neutral position to the position of maximum dorsalflexion. In this position of maximum dorsal flexion the foot needs to perform several inversion- and eversion movements. The software now calculates the talocrural and subtalar ankle joint axis from the movement data of the recorded ultrasonic pulse markers. The results are displayed graphically and numerically directly after the measurement. As a result of the measurement subtalarjoint axis the software shows the average and standard deviation of the subtalarjoint axis, calculated out of the specific results of a single inversion/ eversion movement of the subject's foot. Based on the displayed standard deviation, the quality of the required movement or the error of measurement can be estimated. The entire diagnostical method had to pass several validity and reliability tests on a mechanical model and in vivo. The calculated error for the reliability tests using the mechanical model was $\Delta E < 0,9°$. The error concerning the in vivo tests was $\Delta E = 2,5°$.

This is slightly higher than the error on the mechanical model, which is most likely due to skin movements and less accurate movement of the subject's foot in maximum dorsalflexion.
In a first study using this new diagnostical method the ankle joint axis's of $n = 97$ subjects were determined. The standard deviation of the results of a single subject was less than $\Delta E < 5°$. Results with standard deviations higher than 5° were rejected. The average of the $n = 97$ subjects of the deviation was $\alpha_{dev} = 3,4°$ with a standard deviation of $STD < 11,4°$.
This new method is capable of determining the ankle joint and subtalar joint axis in vivo even with a large number of subjects and is therefore able to contribute to our current knowledge regarding the hypothetical dependencies of the spatial orientations of the ankle and subtalar joint axis and acute or chronic injuries.

1 Einleitung

Allein in Deutschland ziehen sich jährlich ca. 600.000 Sportreibende Verletzungen der unteren Extremitäten zu [Henke und Gläser 2001]. Der größte Anteil entfällt mit etwa 25% auf den Fuß- und Sprunggelenkkomplex [Gross et al. 1994, Hertel 2000, Beynnon et al. 2001, Baumhauer et al. 1995, Henke und Gläser 2001]. In den großen Ballsportarten wie z.B. dem Fußball, Handball, Volleyball oder auch Basketball dominieren die Sprungelenksverletzungen: Im Volleyball entfallen ca. 50% aller Verletzungen auf das Sprunggelenk [Henke und Gläser 2001], aber auch im Handball und Fußball liegen die Sprunggelenksverletzungen mit knapp 25% an erster Stelle. **Peterson et al.** schätzen die Zahl der täglichen Sprunggelenksverletzungen in den USA auf 23.000 [Peterson und Renström 2002]. Dabei gehören die Sprunggelenksverletzungen nicht nur im Sport, sondern ebenfalls in Beruf und Alltag zu den häufigsten Verletzungen. **Lassiter et al.** rechnen, dass bei ca. 40% der Betroffenen Langzeitsymptome zu erwarten sind [Lassiter et al. 1989]. Einige Autoren stufen das Verletzungsrisiko bei einer vorangegangenen Sprungglenksverletzung als zwei- bis dreimal höher ein als bei Personen ohne Verletzungsanamnese [Denegar et al. 2002, Hertel 2002, Olmsted et al. 2004, Peterson und Renström 2002]. Im Spitzensport stellen chronische Überlastungsschäden an den Strukturen des Bewegungsapparates ein zusätzliches Problem dar [Kannus et al. 2002, Wilder und Sethi 2004, Tibesku und Pässler 2005]. Nach **Wilder et al.** sind ca. 50% der Sportverletzungen Sekundärverletzungen, die aus Überlastungs- bzw. Mikrotraumata resultieren [Wilder und Sethi 2004]. Sowohl für akute Sprungelenkverletzungen [Phillips und Lidtke 1992,Sarrafian 1993] als auch für chronischen Überlastungsschäden [Jones 1945, Inman 1976, Hintermann und Holzach 1992] liefern verschiedene Autoren Erklärungsansätze, bei denen der räumlichen Orientierung der Rotationsachsen des Talocrural- und Subtalargelenks besondere Bedeutung zugeschrieben wird. Die Frage nach einem Zusammenhang zwischen der individuellen Gelenkanatomie und chronischen bzw. akuten Überlastungsschäden der Strukturen der unteren Extremitäten speziell des Sprunggelenks, konnte aufgrund von methodischen Restriktionen bisher eingesetzter Verfahren und den daraus resultierenden kleinen Stichproben nicht beantwortet werden.

Es wird dringend ein neues einfach zu handhabendes diagnostisches Verfahren benötigt, mit dem es möglich ist, die Sprunggelenkachsen-Parameter in großen Fallzahlen in vivo und nicht invasiv hinreichend genau zu bestimmen [Alt 2001]. In dieser Arbeit wurde ein solches Instrumentarium entwickelt, mit dem es nachfolgenden Studien möglich sein wird, den hypothetischen Zusammenhang zwischen der individuellen Gelenkanatomie und chronischen Überlastungsschäden bzw. akuten Verletzungen aufzuklären.

2 Stand der Forschung

Der hypothetische Zusammenhang zwischen der individuellen Gelenkanatomie und chronischen und akuten Überlastungen der Strukturen der unteren Extremitäten wird in der Literatur mehrfach - allerdings wenig spezifiziert - angesprochen. Besonders die sehr große interindividuelle Variationsbreite (siehe Tabelle 1 auf Seite 10) der Lage der Sprunggelenkachsen wurde in diesem Zusammenhang diskutiert [van Langelaan 1983, McClay und Bray 1996, Phillips und Lidtke 1992, Alt 2001]. Unter Berücksichtigung der Erkenntnisse zur Bewegungsübertragung zwischen Calcanaeus und Tibia wird deutlich, dass die im Zusammenhang mit Inversion/Eversion des Fersenbeines einhergehende tibiale Rotation [Jones 1945, Inman 1976, Hintermann et al. 1994, Hintermann 1998] maßgeblich von der Lage der Achse des unteren Sprunggelenkes beeinflusst wird. Mittels in vivo implantierter Knochen-Marker konnte gezeigt werden, dass diese Bewegungsübertragung erhebliche interindividuelle Differenzen beim Gehen und Laufen aufweist [Stacoff et al. 2000]. Vor allem in der Phase des ersten Fersenaufsatzes bis zum vollständigen Fußkontakt findet eine Übertragung vom Rückfuß auf tibiale Rotation statt.
Ähnliche Überlegungen liegen dem Modellierungsansatz von **Wright et al.** zu Grunde [Wright et al. 2000]. Mittels Computersimulation versuchten sie, den Einfluss unterschiedlicher Fußpositionen auf das Verletzungsrisiko zu erfassen. Für die Variation im Bereich des oberen Sprunggelenkes (mehr Plantarflexion) fanden sie ein erhöhtes Risiko zu lateralen Bandverletzungen. Andere Arbeiten weisen darauf hin, dass die Kausalität zwischen der Bewegung im unteren Sprunggelenk und chronischen Achillessehnenbeschwerden nicht hinreichend aufgeklärt ist [Mayer und Dickhut 2002]. Die Tatsache, dass einerseits nicht alle Athleten mit einer sogenannten „Überpronation“ Beschwerden entwickeln und andererseits Athleten mit Achillessehnenproblemen zum Teil nur wenig Pronation beim Laufen zeigen, verdeutlicht die Notwendigkeit, diesbezüglich weiter nach individuellen modulierenden Faktoren zu forschen. Betrachtet man außerdem die Lage der wichtigsten Sprunggelenkmuskeln bzw. deren Ansätze in Relation zur Achse des unteren Sprunggelenks, so wird deutlich, dass die starke Variabilität dieser Achse auch erheblich Konsequenzen auf die wirksamen „Hebelarme“ der entsprechenden Muskeln besitzt und damit deren Muskelkraftmomente maßgeblich beeinflusst [Sarrafian 1993](Abbildung 1).

Phillips berechnete den senkrechten Abstand zwischen den Metatarsalenköpfchen und der auf die Transversalebene projizierten Achse des unteren Sprunggelenks und zeigte, dass eine Verringerung des Deviationswinkels um $\alpha = 8°$ zu einem zehnfach verringerten Drehmoment bezogen auf die Subtalarachse führt [Phillips und Lidtke 1992] (siehe Abbildung 2).

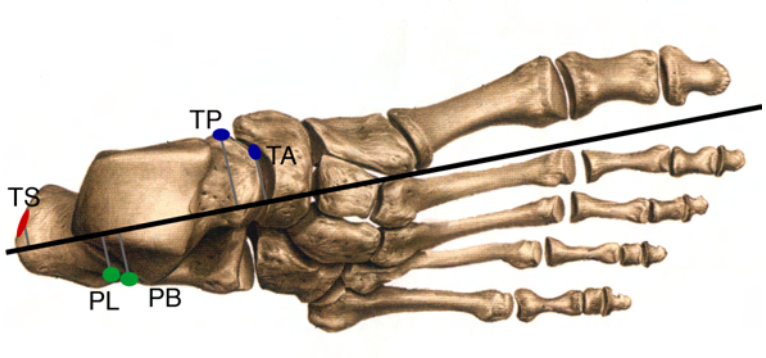

Abbildung 1: Achse des unteren Sprungelenks mit markierten Sehnen der Skelettmuskeln aus *Sobota* modifiziert nach **Sarrafian** [Sarrafian 1993]. PB - M. peronaeus brevis, PL - M. peronaeus longus, TS - M. triceps surae, TA - M. tibialis anterior, TP - M. tibialis posterior. Dargestellt sind die wirksamen Hebel zur USGA. Eine Veränderung der Deviation bedeutet gleichzeitig eine Veränderung der wirksamen Hebel.

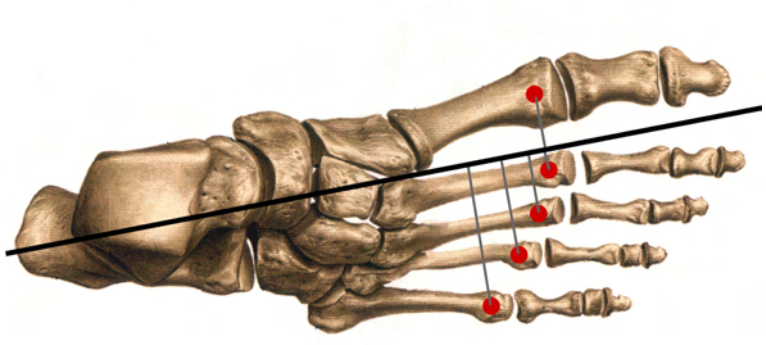

Abbildung 2: Subtalare Gelenksachse und resultierende Momente auf die Transversalebene projiziert (aus *Sobota* [Sobota 2000] modifiziert nach **Phillips** [Phillips und Lidtke 1992]). Je nach Lage der USGA relativ zu den Metatarsalköpfchen lassen sich unterschiedliche Momente für Inversion und Eversion in Abhängigkeit von der Deviation der Achse berechnen.

Nach **Alt** [Alt 2001] fomulierten **Isman et al.** [Isman et al. 1969] insgesamt zehn Parameter, die bei Messungen der Achsen des Sprungelenks (OSGA/USGA) erfasst werden sollten:

1. Winkel zwischen Trochlea tibiae und der Längsachse der Tibia.
2. Winkel der Achse des oberen Sprunggelenks mit der Tibialängsachse[1].
3. Winkel der Achse des oberen Sprunggelenks mit der Fußlängsachse in der Projektion auf die Transversalebene[2].
4. Winkel der Achse des unteren Sprunggelenks mit der Fußlängsachse in der Projektion auf die Transversalebene (Deviation, siehe Abbildung 3).
5. Winkel zwischen der Achse des oberen Sprunggelenks und der Achse des unteren Sprunggelenkes in der Projektion auf die Transversalebene.
6. Tatsächlicher Winkel zwischen der Achse des oberen- und unteren Sprunggelenks.
7. Winkel zwischen der Achse des unteren Sprunggelenks und der Horizontalen in der Projektion auf die Sagittalebene (Inklination, siehe Abbildung 3).
8. Winkel zwischen der Verbindungslinie vom metatarsalen Köpfchen II zum metatarsalen Köpfchen V und der Fußmittellinie.
9. Rechtwinkliger, d. h. minimaler Abstand zwischen den beiden Gelenkachsen.
10. Ein Quotient, der als Maßzahl zur Abschätzung der Lage der subtalaren Gelenkachse in vio dienen sollte.

Die Ergebnisse der Rotationsachsen (OSGA und USGA) werden durch Geraden im Raum angegeben. In der Literatur kann man nach **Alt**, sowohl für die OSGA als auch für die USGA, prinzipiell drei methodische Typen von Achsen Ergebnisse unterscheiden:

Finite Achse: als Ergebnis einer Mittelwertbildung in einem durch zwei willkürliche Positionen begrenzten endlichen Bewegungsabschnitt.

biphasische Achsen: als Kombination zweier finiter Achsen für zwei bestimmte Bewegungsabschnitte (z. B. eine Achse für die Inversion und eine für die Eversion).

[1] Dieser Winkel wird nachfolgend als Inklination der OSGA bezeichnet.
[2] Dieser Winkel wird nachfolgend als Deviation der OSGA bezeichnet.

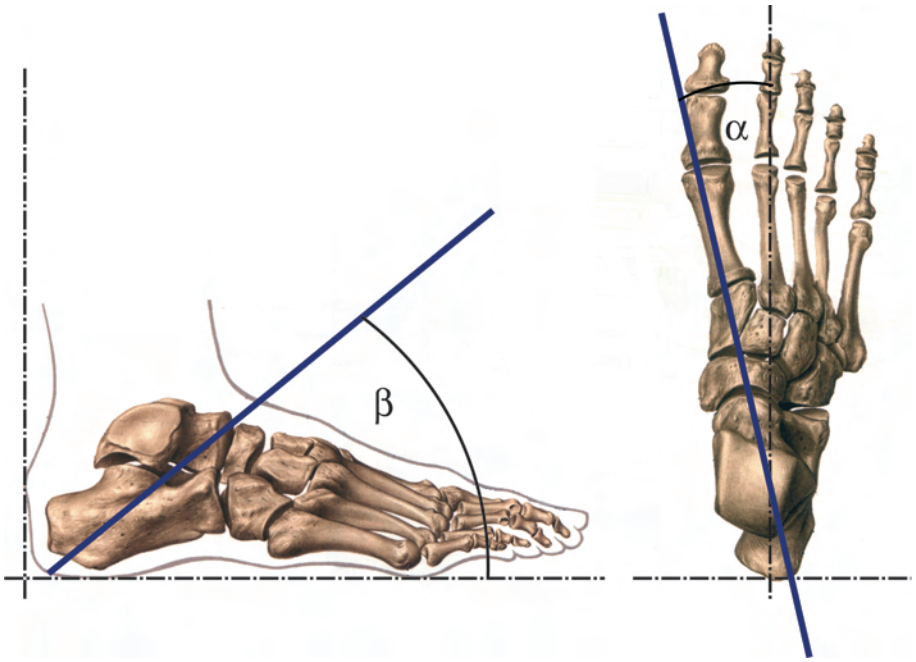

Abbildung 3: Inklination und Deviation der Achse des unteren Sprungelenkes. Bild links zeigt die USGA in der Projektion auf die Sagittalebene, das Bild rechts zeigt die USGA in der Projektion auf die Transversalebene. Der Winkel β gibt die Inklination, der Winkel α die Deviation an (aus Sobota [Sobota 2000] modifiziert nach **Isman et al.** [Isman et al. 1969]).

Instantane (infinite) Achse: Momentane Achse als Ergebnis einer quasi kontinuierlichen Bewegungsanalyse mit annähernd unendlich kleinen Inkrementen zwischen den einen Bewegungsabschnitt begrenzenden Positionen.

Verschiedene Autoren [Fick 1911, Inman 1976, Debrunner und Jacob 1998] beschreiben die Bewegung im oberen Sprunggelenk als reine Scharnierbewegung um die finite Achse des OSG ohne Translationsanteile. Eine zweite Gruppe Autoren geht von einer nicht fixierten Achse im oberen Sprunggelenk aus. Für Dorsalflexion und Plantarflexion werden meist zwei unterschiedliche Achsen beschrieben [Hicks 1953, Barnett und Napier 1952]. **Van Langelaan** und **Demarais et al.** beschreiben die Bewegung im oberen Sprunggelenk mithilfe instantaner Achsen [van Langelaan 1983, Demarais et al. 2002].

Zur Lage der USGA werden in der Literatur ebenfalls unterschiedliche Angaben gemacht (siehe Tabelle 1). Es lässt sich allerdings feststellen, dass der Großteil der Autoren eine finite USGA angibt [Elftman und Manter 1935, Isman et al. 1969, van den Bogert et al. 1994, Alt 2001]. Biphasische Rotationsachsen werden lediglich von **Downing et al.** [Downing et al. 1978] und **Lundberg** [Lundberg 1989] angegeben und nur **Van Langelaan** [van Langelaan 1983] gibt die

Ergebnisse der USGA als instantane Achsen aus. Die Aussagen bezüglich Schraubungscharakter der USGA sind ebenfalls widersprüchlich. Während **Manter** bei einer Rotation um 10° eine Translation von $1,5mm$ angibt findet **Inman** bei 58% aller untersuchten Präperate eine anteriore Translation, bei 28% eine posteriore Translation. Die Aussagen von **Van Langelaan** sind differenzierter, wobei große Rotationen nicht zwangsläufig mit großen Translationen einher gehen. **Alt** sieht mögliche Ursachen für die widersprüchlichen Angaben bezüglich des Schraubungscharakters der USGA in:

- ◇ den untersuchungsmethodischen Restriktionen in Bezug auf die Messgenauigkeit.
- ◇ der starken interindividuellen Variabilität.
- ◇ der Art und Anzahl der in die Untersuchung einbezogenen Gelenkstrukturen.
- ◇ der Art und Weise der Fixation der zu untersuchenden Gelenkstrukturen bzw. der Einleitung der Bewegung auf das Gelenk.

Er stellt darüber hinaus fest,

> „[...] dass die translatorische Bewegung um die Helicalaxis im unteren Sprunggelenk, wenn sie vorhanden ist, nur in geringem Ausmaß stattfindet, welches einen Wert von $2 - 3mm$ für die gesamte Bewegung im unteren Sprunggelenk wohl kaum überschreiten dürfte."

Nach **Alt** lassen sich die Arbeiten zur qualitativen und quantitativen Beschreibung der funktionellen Sprunggelenksanatomie in drei Funktionskategorien einteilen:

- ◇ direkte Verfahren mit unmittelbarer Erfassung der Bewegung der gelenkbildenden Knochen
- ◇ indirekte Verfahren über Erfassung analoger Bewegungsparameter
- ◇ mathematische Modellierung.

Die Ergebnisse dieser Untersuchungen sind in Tabelle 1 zusammengefasst. Die ersten Studien zur Lage der Sprunggelenkachsen setzten ausschließlich die direkten Verfahren ein. Dabei wurden in in vitro Experimenten meist die artikulierenden Gelenkflächen manuell gegeneinander bewegt. Zur Bestimmung der Gelenkachsen kamen verschiedene Techniken zum Einsatz:

1. Bei fixiertem Calcaneus werden die Raumbewegungen von am Talus angebrachten Zeigern verfolgt und anschließend mithilfe eines graphisch-mechanischen Verfahrens die Gelenkachse bestimmt [Manter 1941, Root et al. 1966].

2. Es wird der oberflächliche Punkt gesucht, der bei der durchgeführten Rotation eine minimale Bewegung aufweist [Hicks 1953, Isman et al. 1969].

3. In den beteiligten Knochen werden kleine Stahlspitzen eingebracht, die dann bei einer Rotationsbewegung Spurrillen in den Gelenkflächen hinterlassen. Aus diesen Spurrillen werden dann die Rotationsachsen berechnet [Isman et al. 1969].

Bei den in vivo Methoden zur Bestimmung der Sprunggelenkachsen handelt es sich um indirekte Verfahren, wobei man die invasiven und nicht invasiven Verfahren unterscheiden muss. Bei beiden Vorgehensweisen werden die Rotationsachsen mithilfe verschiedener mathematischer Methoden aus den erfassten Bewegungsparametern berechnet. Die Bewegung des Talus kann bei den invasiven Verfahren objektiviert werden. Mittels Stereoröntgen-Fotogrammetrie [Lundberg 1989, van Langelaan 1983] oder Fotogrametrie mit angebrachten Intrakortikalmarken [Arndt et al. 2004] können die Bewegungsdaten erfasst werden.

Für das Problem der nicht objektivierbaren Talusbewegungen bei nicht invasiven in vivo Messungen entwickelten verschiedene Autoren unterschiedliche Lösungsansätze. **Alt** zeigte in einem in vitro Versuch, dass bei maximaler Dorsalflexion das obere Sprunggelenk als festgesetzt betrachtet werden kann [Alt et al. 1998]. Eine Inversions/Eversionsbewegung in der maximalen Dorsalflexion kann demnach als eine isolierte Bewegung im unteren Sprunggelenk betrachtet werden. Zur Erfassung der Daten setzt der Autor 3D Photogrammetrie ein. Insgesamt sechs Infrarot-Reflex-Marker werden an Fuss und Tibia der Probanden angebracht. Die drei an der Tibia angebrachten Marker dienen zur Berechnung des *Tibiakoordinatensystems*. Dazu werden Marker 1 und Marker 2 parallel zur Tibiakante ausgerichtet, Marker 3 wird so justiert, dass er senkrecht zu Marker 2 und gleichzeitig parallel zur Fußlängsachse steht. Aus den Momentaufnahmen der endgradigen Eversion- und Inversionstellungen berechnet der Autor mithilfe des *Helical Axis Verfahrens* eine finite Rotationsachse des unteren Sprunggelenks (siehe Tabelle 1).

Van den Bogert geht sowohl beim oberen als auch beim unteren Sprunggelenk von einfachen Scharniergelenken aus. Dadurch reduziert sich das Problem auf eine zweidimensionale Rotation in der Ebene. In einem $500mm \cdot 500mm \cdot 500mm$ großen Kalibrationsraum, lässt er die Versuchspersonen mehrere Bewegungen ausführen, die von einem 3D Videoanalysesystem aufgezeichnet werden:

1. aus der maximalen Dorsalflexion in die maximale Plantarflexion,
    a. bei endgradiger Eversion im unteren Sprunggelenk,
    b. in der Neutralstellung im unteren Sprunggelenk,
    c. bei endgradiger Inversion im unteren Sprunggelenk.
2. aus der maximalen Eversion in die maximale Inversion,
    a. in der maximaler Plantarflexion,
    b. in der Semi-Plantarflexion
    c. in der maximalen Dorsalflexion
3. eine volle kreisförmige Bewegung.

Das mathematische Modell des Sprunggelenks wird durch einen Optimierungsalgorithmus und eingezogenen Bewegungsdaten jeder oben aufgeführten Einzelbewegung solange kalibriert, bis das Modell und das reale Sprunggelenk hinreichend genau übereinstimmen. **Demarais** stellt eine weitere Lösungsstrategie zur Bestimmung der Sprunggelenksachsen in vivo vor [Demarais et al. 2002]. Auch seine Methode basiert auf einem 3D Videoanalysesystem. Aus den Daten von insgesamt sechs Markern berechnet er eine Ebene und eine Kugeloberfläche, die möglichst viele der aufgezeichneten Markerpunkte enthalten sollen. Durch diese Annäherung kann dann die Rotationsachse, die senkrecht durch die Ebene und durch das Zentrum der Kugeloberfläche verlaufen muss, angegeben werden. Das Verfahren wurde bisher in keiner Studie eingesetzt.

Zusammenfassend lässt sich sagen:

- ◇ Die meisten Autoren gehen von einer monoaxialen Bewegung sowohl im oberen als auch im unteren Sprunggelenk aus.
- ◇ Die Aussagen bezüglich des Schraubungscharakters der Achse des unteren Sprunggelenks sind widersprüchlich. Die translatorische Bewegung um die Schraubungsachse wird, wenn sie existiert, einem Umfang von $2 - 3mm$ kaum überschreiten, d. h. das Ergebnis liegt bei nicht invasiven in vivo Messungen innerhalb der Fehlergrößen und kann somit vernachlässigt werden.
- ◇ Eine große interindividuelle Varianz, besonders die Deviation der USGA betreffend, wird von allen Autoren bestätigt.

- ◇ Nicht invasive in vivo Messungen[3] versuchen die nicht direkt objektivierbare Talus Bewegungen entweder mechanisch zu minimieren [Alt 2001] oder benutzen mathematische Optimierungsalgorithmen um eine Annäherung eines mathematischen Modells an die tatsächlichen Achslagen des Sprunggelenks zu schaffen.

Aufgrund der geringen Probandenzahlen bei bisher durchgeführten in vivo Studien, lässt sich die Frage nach einem möglichen Zusammenhang, zwischen chronischen Überlastungsschäden oder akuten Verletzungen des Sprunggelenks und diskreten Sprunggelenksachslagen noch nicht beantworten. Um dieser Frage nachzugehen, muss ein diagnostisches Verfahren geschaffen werden, das in der Lage ist, in vivo und nicht invasiv die Sprunggelenkachsen hinreichend genau zu bestimmen.

[3] Diese Methoden beruhen meist auf Bewegungsanalysen des Sprunggelenkkomplexes.

Tabelle 1: Literatur Ergebnisse zur Lage der subtalaren Gelenkachse modifiziert nach **Alt**.

Autor	Jahr	n	Lage	$\alpha[^\circ]$	h.A.	Typ	Methode
Elftmann u. Manter	1935	n.a.	Ink. Dev.	43 28	n.a.	finite	in vitro Gelenkgeometrie
Manter	1941	16	Ink. Dev.	42 16	ja	finite	in vitro graphisch-mechanisches Verfahren mit am Talus fixierten Zeigern
Root et al.	1966	22	Ink. Dev,	41 17	nein	n.a.	in vitro gaphisch-mechanisches Verfahren mit am Talus fixierten Stahlpins
Ismann u. Inman	1969	47	Ink. Dev.	42 23	n.a.	finite	in vitro manuelles Bewegen des Talus, Suche nach Punkt minimaler Bewegung
Downing et al.	1978	50	Ink. Dev.	43 18	n.a.	biphasisch	in vivo Berechnung aus Endstellung max. Pronation/Supination, mithilfe angebrachter Zeiger an Calcaneus und Tibia.
Van Langelaan	1983	10	Ink. Dev.	43 17	ja	instantan	in vitro Stereo-Röntgen-Photogrammetrie
Lundberg	1989	8	Ink. Dev.	33 29	n.a.	biphasisch	in vivo, invasiv Stereo-Röntgen-Photogrammetrie mit applizierten Tantalum-markern.
van den Bogert et al.	1994	14	Ink. Dev.	37 18	n.a.	finite	in vivo, nicht invasiv 3D-Photogrammetrie, Berechnung durch Anpassung eines 2D Modells an die kinematischen Daten des Experiments.
Alt	2001	22	Ink. Dev.	56 8	n.a.	finite	in vivo, nicht invasisv 3D-Photogrammetrie, Berechnung mithilfe des Helical-Axis Verfahrens.

3 Entwicklung des diagnostischen Verfahrens

Die folgenden Kapitel beschreiben die einzelnen Phasen der Planung und Entwicklung des in dieser Arbeit entwickelten diagnostischen Verfahrens zur Bestimmung der Sprunggelenkachsen in vivo. Ausgehend von einem in einem ersten Schritt erstellten Anforderungsprofil werden Programmstruktur, mathematische Methoden, Software und Hardware entwickelt. Zwei Vorgehensmodelle aus der Softwareentwicklung halfen bei der Planung der einzelnen Entwicklungsschritte: das *Wasserfall* [Royce 1970]- und das *Spiralmodell* [Boehm 1988]. Das *Wasserfallmodell* untergliedert den Entwicklungsprozess in folgende Phasen:

1. Anforderungsanalyse
2. Systemdesign und -spezifikationen
3. Programmierung und Modultests
4. Integration und Systemtests
5. Auslieferung, Einsatz und Wartung

Das *Wasserfallmodell* besitzt einen eindeutigen Start- und Endpunkt und wird nur ein einziges Mal durchlaufen. Da es bei Projektbeginn schwierig ist, alles endgültig im Detail festzulegen - oft verändern sich während der Entwicklung die Anforderungen an die Methode - kommen in der Praxis meist alternative Modelle zur Softwareentwicklung zum Einsatz. Die Entwicklung des neuen diagnostischen Verfahrens zur Bestimmung der Sprunggelenkachsen orientiert sich an der Struktur des *Spiralmodells*[4]. In der Unterteilung der Phasen ähnelt das *Spiralmodell* dem *Wasserfallmodell*, allerdings werden diese Phasen mehrmals durchlaufen. Das *Spiralmodell* betrachtet die Prototypenentwicklung als iterativen Prozess. Abbildung 4 auf Seite 12 zeigt die einzelnen Entwicklungsabschnitte sowie die verschiedenen Prototypenstadien bis zum betriebsfähigen Prototypen.

[4] Die Gliederung der Arbeit lässt zwar eine Orientierung am *Wasserfallmodell* vermuten, diese Struktur ist aber lediglich der besseren Übersichtlichkeit wegen gewählt worden.

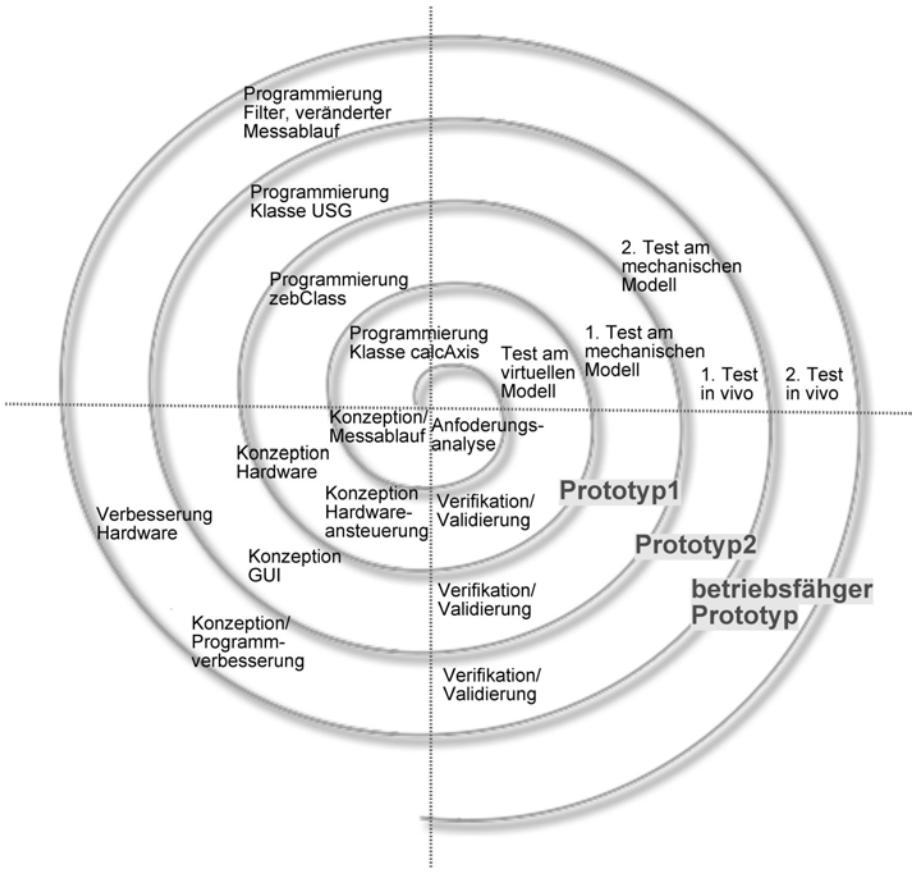

Abbildung 4: Spiralmodell, Phasen der Entwicklung des diagnostischen Verfahrens. Ausgehend von einer ersten Anforderungsanalyse wird das Messsytem entwickelt, geprüft und ständig verbessert. Über die Entwicklung verschiedener Prototypstadien kommt es schließlich zur Fertigstellung eines betriebsfähigen Prototypen.

3.1 Problemanalyse und Anforderungsprofil

In Kapitel 2 ab Seite 2 wurden bereits verschiedene Verfahren zur Bestimmung der Sprunggelenkachsen vorgestellt. Die meisten dieser Verfahren liefern durchaus brauchbare Ergebnisse, haben aber den entscheidenden Nachteil, dass sie Laborbedingungen benötigen und meist mit einem erheblichen instrumentellen Aufwand verbunden sind.
Daraus ergeben sich folgende Anforderungen an das neu zu entwickelnde Verfahren:

- ◇ in vivo Messung
- ◇ nicht invasiv
- ◇ einfache Handhabung
- ◇ schneller Auf- und Abbau
- ◇ Ergebnisse sollen ohne weitere Auswertungsschritte nach der Messung vorliegen
- ◇ Methode darf keine Laborbedingungen benötigen, soll in Feldversuchen einsetzbar sein
- ◇ Zuverlässig, auch bei Messung mit großen Probandenzahlen (geringe Störanfälligkeit)

Diese Methode wird wie auch die bereits vorhandenen Methoden zur Gelenkachsenbestimmung in vivo auf einer Bewegungsanalyse des Sprunggelenks basieren. So kann indirekt über die Bewegung der am Sprunggelenk angebrachten Marker die räumliche Orientierung und Position der Rotationsachsen berechnet werden. Es wird zunächst also eine Hardware benötigt, mit der es möglich ist, Markerpositionen aufzuzeichnen und diese dann an eine geeignete Software weiterzugeben. Diese Software muss dann wiederum in der Lage sein, aus den 3D Koordinatendaten der aufgezeichneten Bewegung die Rotationsachsen zu berechnen.
Eine Unterteilung in nachfolgende Arbeitsschritte erschien daher sinnvoll:

1. Überprüfung verschiedener Verfahren zur Positionsbestimmung im dreidimensionalen Raum. Dabei sollten die vom Verfahren verwendeten physikalischen Mess- bzw. geometrische Beobachtungsprinzipien unbedingt die in diesem Kapitel formulierten Anforderungen erfüllen.
2. Suche nach einer geeigneten Hardware, die das gewünschte Verfahren zur Positionsbestimmung einsetzt. Diese Hardware muss Schnittstellen besitzen, die eine automatisierte Datenakquisition zulassen.

3. Entwicklung eines mathematischen Verfahrens, dass in der Lage ist, aus den Bewegungskoordinaten die Rotationsachsen zu berechnen. Gegebenenfalls kann auf bereits bestehende Algorithmen zurückgegriffen werden.

4. Entwicklung einer Software, die die Aufgaben der Datenverarbeitung mit allen dafür notwendigen Auswerteschritten übernimmt. Die Ergebnisse sollen sowohl numerisch als auch grafisch dargestellt werden.

5. Pilotversuche zur Fehlerabschätzung und Überprüfung der Gütekriterien des gesamten Messverfahrens bestehend aus Hardware, mathematischen Algorithmen und Software zur automatisierten Datenverarbeitung. Gegebenenfalls Durchführung von verschiedenen Modifikationen zur Fehlerminimierung.

6. Erster Einsatz im Feld. Möglicherweise Modifikationen des Messablaufs um eine einfache Handhabung des Systems zu gewährleisten.

3.2 Verfahren zur Positionsbestimmung im dreidimensionalen Raum

Unter dem Begriff „Verfahren zur Positionsbestimmung“ oder einfacher „3D-Meßsysteme“ versteht man Hilfsmittel, mit denen es möglich ist, die Form, die Lage oder Lagetrajektorie eines Objektes dreidimensional zu erfassen [Ziegler 1996]. Eine sinnvolle Einteilung der Systeme kann entweder nach der gesuchten Zielgröße (Form, Lage, Orientierung), dem verwendeten physikalischen Messprinzip (z. B. optische oder mechanische Abtastung) oder dem geometrischen Beobachtungsprinzip (Hyperbel-, Triangulations- oder Trilaterationsverfahren) erfolgen. Eine weitere wichtige Eigenschaft von 3D-Messsystemen, ist die Fähigkeit der Positionsbestimmung im kinematischen Zustand, wobei der Bewegungsablauf eines Objektpunktes durch einen zeitvariablen Ortsvektor in Bezug auf ein gewähltes Koordinatensystem beschrieben werden kann. Ein wesentliches Kriterium kinematischer Messsysteme ist die mögliche Abtastfrequenz der Positionsbestimmung, um den Bewegungsablauf hinreichend gut zu beschreiben. Die geforderte Messfrequenz ist hierbei abhängig von den möglichen Bewegungsraten und den gestellten Genauigkeitsanforderungen [Mönicke 1994].
Da bei der Versuchsanordnung zur Bestimmung der Sprunggelenkachsen kleine Bewegungsamplituden und niedere Bewegungsfrequenzen auftreten werden, sind die Anforderungen an die Abtastfrequenz des Messsystems eher gering.
Sowohl die Bewegungsamplitude als auch die Bewegungsausführung dürfen aber auf keinen Fall durch das Anbringen von mechanischen Sensoren oder die Bauformen von Markern eingeschränkt werden. Das Verfahren darf auch bei häufigem Einsatz keine Abnutzungserscheinungen zeigen, nur so kann gewährleistet werden, dass auch

bei einer Messung einer großen Anzahl an Probanden das Gesamtsystem im Einsatz zuverlässige Werte liefert.

3.2.1 Verfahren mechanischer Abtastung

Verfahren zur mechanischen Abtastung bzw. Positionsbestimmung sind im einfachsten Falle Messgeräte wie mechanische Entfernungsmesser, mit denen der Abstand eines Punktes zu einem Referenzpunktes eines Koordinatensystems ermittelt werden kann. In diesem Fall lassen sich die Vorgänge zur Automation von Datenerfassung und Datenverarbeitung der Messwerte fast nicht realisieren. Eine etwas komfortablere Lösung, die eine automatische Erfassung der Messwerte zulassen würde, sind Konstruktionen aus Linearpotentiometer und verschiedenen Seilzügen. Allerdings haben alle mechanische Abtastverfahren den Nachteil, dass sie durch die mechanische Konstruktion unter Umständen die Freiheitsgrade des zu vermessenden Systems einschränken könnten. Ein solches Verfahren wird durch den häufigen Einsatz mit großer Wahrscheinlichkeit mechanische Abnutzungserscheinungen und dadurch Messungenauigkeiten aufweisen. Deswegen soll an dieser Stelle nicht weiter auf diese Verfahren eingegangen werden.

3.2.2 Magnetische Verfahren

Die magnetischen Verfahren sollen hier nur der Vollständigkeit halber erwähnt werden. Sie basieren meist auf permanent-magnetischen Kugeln, die als Marker fungieren und einfachen Magnetfeldsensoren. Der Vorteil eines solchen Verfahrens liegt in dem berührungslosen Messaufbau und der Möglichkeit der Automation von Messdatenerfassung und -Verarbeitung. Allerdings scheiden diese Systeme aufgrund ihrer hohen Störanfälligkeit, für den Einsatz im Feld aus. Das zur Bewegungsbestimmung verwendete inhomogene magnetische Wechselfeld könnte z. B. durch elektronische Geräte (Netzteil, Computermonitor etc.) gestört werden.

3.2.3 Optische Verfahren

Optische Trackingverfahren arbeiten auf der Basis digitaler Kameras. Der Vorteil dieser Verfahren liegt in der berührungslosen Messwertaufnahme. Um bestimmte Punkte aufzuzeichnen müssen lediglich optische Marker angebracht werden. **Alt** et. al. [Alt et al. 1998] haben bereits Messungen zur Bestimmung der Sprunggelenksachsen mit Hilfe des auf Infrarot-Basis arbeitenden Bewegungsanalysesystems MotionAnalysis® durchgeführt. Dabei wurden bestimmte Positionen des Sprunggelenks mit einem 4-Kamera-Verfahren in einem Kalibrationsraum aufgezeichnet. Die

Raumkoordinaten wurden anschließend mit der Software Kintrack® ermittelt und zuletzt die untere Sprunggelenksachse mit Hilfe des Programms Helical Axis berechnet [Jacob 1989]. Wegen der Notwendigkeit eines Kalibrationsraumes benötigt dieses Verfahren allerdings unbedingt Laborbedingungen und ist somit für den Einsatz im Feld nicht geeignet. Obwohl die Marker- und Datenerfassung wie auch die Datenverarbeitung weitgehend automatisiert war, benötigte dieses Verfahren doch einige Auswertungsschritte, bis das Ergebnis der Lage und Position der unteren Sprunggelenksachse vorlag.

3.2.4 Impulslaufzeitverfahren

Impulslaufzeitverfahren sind weit verbreitete Verfahren zur Entfernungsmessung. Bei diesen Verfahren wird ein Impuls ausgesendet, der nach Durchlaufen einer Messstrecke am Empfänger detektiert wird. Für die Zeitmessung bietet sich das digitale Zeitmessverfahren an, wobei sich die Laufzeit durch die Zählung der von einem Oszillator erzeugten Impulse ergibt.
Abbildung 5 auf Seite 16 zeigt den schematischen Aufbau einer Laufzeitmessung.

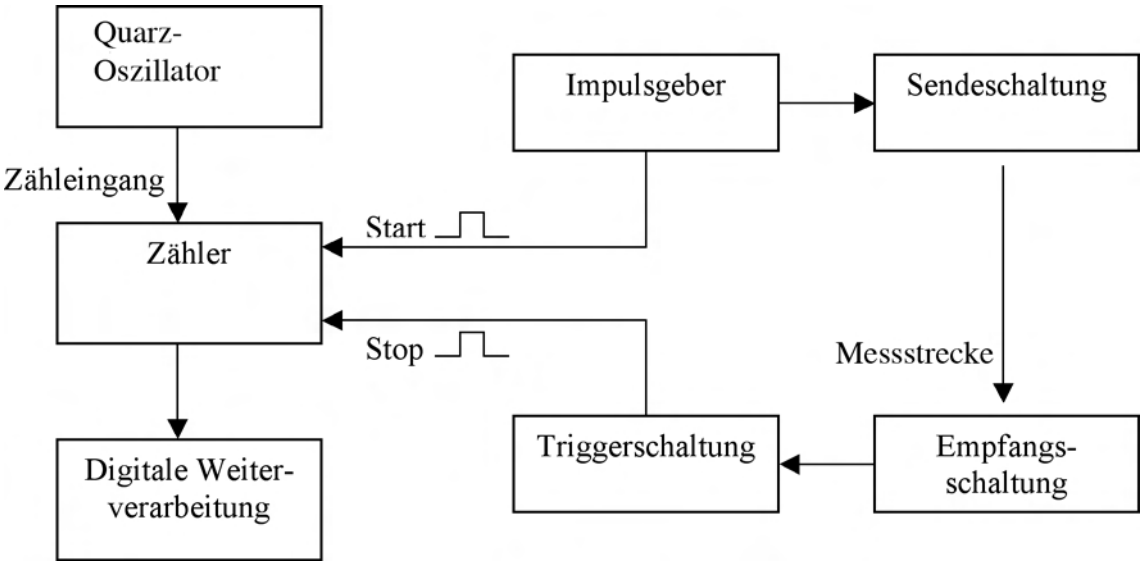

Abbildung 5: Schematischer Aufbau von Impulslaufzeitverfahren: Der Impulsgeber veranlasst die Sendeschaltung zum Aussenden eines Impulses, gleichzeitig wird ein meist digitaler Zähler gestartet. Die Empfangsschaltung detektiert den Impuls, nach Durchlaufen seiner Messstrecke, über die Triggerschaltung wird der Zähler gestoppt. Aus der benötigten Laufzeit und der Ausbreitungsgeschwindigkeit des Impulses im jeweiligen Medium, kann die Entfernung von Sender zu Empfänger berechnet werden.

Impulslaufzeiten können sowohl mit Schallwellen als auch mit elektromagnetischen Wellen, also z. B. Lichtwellen (Laser), durchgeführt werden. Als vorteilhaft erweist

sich bei kürzeren Messtrecken[5] die geringe Ausbreitungsgeschwindigkeit der Schallwellen gegenüber der von elektromagnetischen Wellen, so dass bei relativ geringen Zählfrequenzen hohe Auflösungen erzielt werden können. Unterschieden werden des weiteren die *direkte Streckenmessung* und das *Transponderverfahren* [Mönicke 1994]. Während beim *Transponderverfahren* der ausgesendete Impuls von einem Reflektor reflektiert wird, also die doppelte Messstrecke durchlaufen wird, befinden sich bei der *direkten Streckenmessung* auf den Endpunkten der Strecke jeweils ein Impulssender und ein Impulsempfänger. Dadurch können Auswirkungen von Störechos praktisch ausgeschaltet werden, da das direkte Signal immer den kürzesten Laufweg besitzt.

Funktionsweise von Trilaterationsverfahren

Um ausgehend von der gemessenen Strecke die Position eines Markers bzw. Impulssenders berechnen zu können, werden Trilaterationsverfahren eingesetzt. Unter dem Begriff *Trilaterationsverfahren* lassen sich sehr viele unterschiedliche Verfahren innerhalb der industriellen Messtechnik zusammenfassen. Die Auflösung solcher Systeme erreicht mit handelsüblichen Sensoren Größenordnungen von $1\mu m$ [Hartrumpf 1991], ist also für eine Bewegungsanalyse des Sprunggelenkkomplexes durchaus geeignet. Das Prinzip der *Trilaterationsverfahren* ist denkbar einfach. Aus dem bekannten Abstand a zweier Empfänger lassen sich mit Hilfe des *Impulsmessverfahrens* die Abstände b und c bestimmen (siehe Abbildung 6 auf Seite 18). Die exakte Position des Senders lässt sich damit konstruieren. Um die Position des Senders im Raum zu ermitteln, werden drei Empfänger und ihre Abstände zueinander benötigt[6].
Rechnerisch läst sich die Position des Senders mit Hilfe des Kosinussatzes ermitteln[7]:

$$c^2 = b^2 + a^2 - 2 \cdot b \cdot a \cdot cos\gamma$$
$$b^2 = c^2 + a^2 - 2 \cdot c \cdot a \cdot cos\beta$$

Ein auf der Impulslaufzeitmessung basiertes Verfahren besitzt gegenüber den vorgestellten Verfahren (in den Kapiteln 3.2.1, 3.2.2, 3.2.3, auf Seite 15) einige Vorteile:

- ◇ Die Markerpositionen müssen nicht wie bei optischen Verfahren, aus einer großen Menge von Information (Videomaterial) herausgefiltert werden. Lediglich der Sender sendet Informationen. Meist senden die Sender nur in einem schmalen Frequenzbereich. Die Empfänger empfangen wiederum nur innerhalb dieses Frequenzbandes; Störeinflüsse werden so minimiert.

[5] Die Marker-Empfänger Abstände werden bei der Bewegungsanalyse des Sprunggelenkkomplexes unter $a = 50cm$ liegen. Ein Impulslaufzeitverfahren, das auf der Basis von Schallwellen arbeitet, scheint für diese Problemstellung geeignet zu sein.

[6] Der Schnittpunkt ihrer Kugelflächen ergibt die Position des Senders im dreidimensionalen Raum.

[7] Die Strecke a ist bekannt, die Strecken b und c können aus den jeweiligen Impulslaufzeiten und Ausbreitungsgeschwindigkeiten berechnet werden.

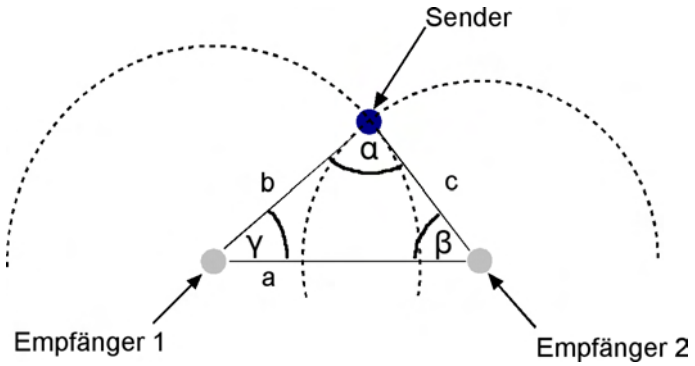

Abbildung 6: Funktionweise eines zweidimensionalen Trilaterationsverfahrens: Aus dem bekannten Abstand a von Empfänger 1 und Empfänger 2 und der jeweils benötigten Laufzeit des Impulses vom Sender zum Empfänger 1 (Strecke b) bzw. Empfänger 2 (Strecke c), lässt sich die exakte Position des Senders geometrisch konstruieren beziehungsweise berechnen. Die zweite Lösung kann wegen unplausibler Strecken ignoriert werden.

- ◇ Die dadurch erreichte Reduktion der Datenmenge begünstigt eine automatisierte Datenakquisition und Datenverarbeitung.
- ◇ Impulslaufzeitverfahren benötigen keinen speziellen Kalibrationsraum. Diese Systeme sind in der Lage, sich selbst zu kalibrieren. Abweichungen der Ausbreitungsgeschwindigkeit z. B. durch Temperaturänderung bei Schallwellen, können von solchen Systemen selbstständig berechnet werden.
- ◇ Sie sind berührungslos, weder Abnutzungserscheinungen noch Beeinträchtigung der Bewegungsamplitude oder des Bewegungsausmaßes sind in der Regel zu erwarten. Das Befestigungs- bzw. Applikationsproblem kann dadurch ebenfalls minimiert werden.

Zusammenfassend lässt sich sagen: Ultraschall scheint für diese Anwendung am besten geeignet zu sein. Durch die relativ geringe Ausbreitungsgeschwindigkeit von Schallwellen ($c \approx 331 m/s$) lassen sich auch bei geringer Zählfrequenz hohe Ortsauflösungen erzielen. Außerdem werden Schallwellen kugelförmig abgestrahlt[8], d. h. der Empfänger kann bei verschiedenen Raumwinkeln ein Schallsignal detektieren.

3.3 Ultraschall-Laufzeit-Messsystem

Da bereits verschiedenste Ultraschall-Messsysteme auf dem Markt erhältlich sind, ist es nicht notwendig, eine eigene Hardware zu entwickeln. Die existierenden Systeme sollten aber bestimmte Eigenschaften besitzen:

[8] Bei einem System, das mit den gebündelten Lichtwellen eines Lasers arbeitet müssen Sender und Empfänger aufeinander ausgerichtet werden.

- ◇ Das Messsystem sollte auf dem Verfahren der direkten Impulslaufzeitmessung basieren. Das heißt Sender und Empfänger sind an den Endpunkten der Messstrecke angebracht. Nur so lassen sich Störungen durch Reflektionen minimieren.

- ◇ Sowohl Sender als auch Empfänger sollten möglichst klein und leicht sein, um ein variables Anbringen am Fuß zu ermöglichen.

- ◇ Das Messsystem sollte eine Schnittstelle zur Verfügung stellen, um das Entwickeln einer speziellen Software (Messprogramm) zu ermöglichen.

- ◇ Im Nahbereich sollte eine größtmögliche Messgenauigkeit erzielt werden können.

Das Bewegungsanalysesystem CMS 20 von Zebris®
Das Messsystem CMS 20 der Firma Zebris® scheint alle diese Anforderungen zu erfüllen. Das Ultraschall-Bewegungsanalysesystem CMS 20 wurde bereits erfolgreich bei Kiefergelenk-, Gleichgewichts-und Wirbelsäulenanalysen eingesetzt. Mit dieser Hardware lassen sich gleichzeitig maximal 8 Miniatur-Ultraschallmarker (Sender) mit drei Mikrofonen erfassen.
Technische Daten des CMS 20:

- ◇ maximal 8 Ultraschallsender (Masse $= 1g$; Frequenz $= 40KHz$; Abstrahlwinkel $> 120°$) und 3 Mikrofone

- ◇ Einzugsfrequenz im Nahbereich ($< 0,5m$), $300Hz/$ Anzahl der Marker

- ◇ Genauigkeit im Nahbereich liegt zwischen $1/100mm - 1/10mm$ [Zebris Medizintechnik 2000]

- ◇ Einzug über die RS232 oder USB Schnittstelle

- ◇ Dynamic Link Lybrary (DLL) ermöglicht das Entwickeln von eigener Messsoftware

- ◇ Gesamtsystem hat eine Masse von 3 kg und ist somit transportabel

Das CMS 20 ist für somit die Entwicklung eines eigenen diagnostischen Verfahrens zur Bestimmung der Sprunggelenkachsen bestens geeignet.

3.3.1 Eigenschaften von Ultraschall

Mit der Bezeichnung Ultraschall kennzeichnet man in Anlehnung an die Definition des ultravioletten Lichts diejenigen akustischen Erscheinungen, die aufgrund ihrer hohen Frequenz der menschlichen Wahrnehmung nicht zugänglich sind. Da die obere Grenzfrequenz individuell verschieden ist und insbesondere vom Lebensalter abhängt, ist keine scharfe Abgrenzung vom Bereich des hörbaren Schalls gegeben. Die Grenze wird jedoch im Allgemeinen zwischen 16 und 20 kHz angenommen.
Bei einer Verwendung von Schall- oder Ultraschall für das Impulslaufzeitverfahren müssen bestimmte Eigenschaften beachtet werden. Die Ausbreitungsgeschwindigkeit ist abhängig von den physikalischen Eigenschaften des durchlaufenden Mediums; werden Messungen im Medium Luft durchgeführt, so ist die Schallgeschwindigkeit eine Funktion der Luftzusammensetzung, insbesondere des Wasserdampfgehalts, der Temperatur und des Luftdrucks.

Abhängigkeit der Schallgeschwindigkeit von der Temperatur
Bei Flüssigkeiten und Gasen sind die Moleküle nicht elastisch miteinander gekoppelt, d. h. Schallwellen können nur *longitudinalen* Charakter haben. Durch Schalldruckwellen werden die Volumenelemente des Mediums komprimiert [Paus 1995]. Man erwartet daher in dem Ausdruck für die Phasengeschwindigkeit das Auftreten verschiedener Zustandsgrössen des Mediums:

$$c = \sqrt{\frac{\kappa RT}{M}}$$

mit:
κ: Adiabatenexponent;
R: Molare Gaskonstante;
M: Molmasse;
T: Temperatur in $[K]$;

Abhängigkeit der Schallgeschwindigkeit von der Feuchte
Der Wasserdampfgehalt der Luft beeinflusst ebenfalls die Ausbreitungsgeschwindigkeit des Schalls. Da sich aufgrund der Luftfeuchte jedoch die Dichte der Luft ρ verkleinert und in geringerem Maß ebenso der Adiabatenkoeffizient κ, führt eine Erhöhung der Feuchte zu einer Vergrößerung der Schallgeschwindigkeit. Folgende Gleichung beschreibt den Zusammenhang von Schallgeschwindigkeit Temperatur, Luftdruck und Luftfeuchte [Kuttruff 1988].

$$v = 20,0553 \cdot \frac{T}{1 - 0,3378\frac{e}{p}}$$

mit:
e: Partialdruck des Wasserdampfes $[hPa]$;
p: Druck in $[hPa]$;

Tabelle 2 zeigt zusammenfassend den Einfluss der verschiedenen meteorologischen Zustandsgrößen auf die Schallgeschwindigkeit.

Tabelle 2: Meteorologischer Einfluss auf die Abstandsmessung ($20°C$, $1013hPa$, 50% rel. Luftfeuchte) [Ziegler 1996].

Temperatureinfluss $\Delta[10^{-6}/°C]$	-1800
Druckeinfluss $\Delta p[10^{-6}/hPa]$	+2,1
Feuchteeinfluss $\Delta e[10^{-6}/hPa]$	-190

Abhängigkeit der Schallgeschwindigkeit von Windeinflüssen

Da die Schallwellen im Gegensatz zu elektromagnetischen Wellen stets an ein stoffliches Medium gebunden sind, ergibt sich ein direkter Einfluss der Molekühlbewegung des Mediums auf die Schallausbreitungsgeschwindigkeit. Es lässt sich zeigen, dass im Fall kleiner Mediumsgeschwindigkeiten die Ausbreitungsgeschwindigkeit des Schalls als Summe der Schallgeschwindigkeit im ruhenden Medium und der Komponente der Geschwindigkeit des Mediums in Richtung der Schallausbreitung ergibt. Hieraus resultiert schon bei einer Windgeschwindigkeit von $0,3m/s$ in Richtung der gemessenen Strecke eine relative Verfälschung der Strecke von $1mm$ [Ziegler 1996].

Wie Tabelle 2 auf Seite 21 entnommen werden kann, hat eine Veränderung des Luftdrucks und der Feuchte nur einen geringen Einfluss auf die Ausbreitungsgeschwindigkeit des Schalls. Die Veränderung der Schallgeschwindigkeit durch Windeinflüsse kann ebenfalls vernachlässigt werden[9]. Die Temperaturabhängigkeit der Schallgeschwindigkeit muss dagegen berücksichtigt werden. Vor Messbeginn sollte ein Ultraschall basiertes Messsystem unbedingt auf die aktuelle Raumtemperatur kalibriert werden, um den Messfehler zu minimieren.

[9] Sofern die Messung in geschlossenen Räumen stattfindet.

3.4 Konzeption der Prozessabläufe

Die Entwicklung des diagnostischen Verfahrens zur Bestimmung der Gelenkachsen des Sprunggelenks erfordert eine sorgfältige Planung und Konzeption des gesamten Messablaufes. In Kapitel 3.1, auf Seite 13, wurden in einer ersten Analyse bereits die Anforderungen an das neu zu entwickelnde Verfahren formuliert. In der anschließenden Entwicklungsphase sind dann die Softwarearchitektur und damit auch die benötigten Einzelschritte des gesamten Messablaufes zu entwerfen. Ausgehend von einem ersten Entwurf der Messmethode müssen anschließend die Architektur der benötigten Programmfunktionen und -methoden sowie die mathematischen Algorithmen beschrieben und entwickelt werden.

3.4.1 Konzeption des Messablaufs

Die Programmstruktur bzw. der Messvorgang wird teilweise durch die in Kapitel 3.1 auf Seite 13 formulierten Anforderungen, hauptsächlich jedoch durch die funktionelle logische Abfolge einzelner Prozesse bestimmt.

1. Stammdatenerfassung.
2. Festlegen eines geeigneten Koordinatensystems anhand anatomischer Punkte. Die Messung sollte in einer möglichst neutralen Position, z. B. Hüftbreite, aufrechter Stand, durchgeführt werden.
3. Messung der oberen Sprunggelenkachse. Dazu muss die Rotation des Fußes aus einer Neutralstellung in eine maximale Dorsalflexion aufgezeichnet werden.
4. Messung der unteren Sprunggelenkachse. Aus den Daten einer wiederholten Inversions- und Eversionsbewegung bei maximaler Dorsalflexion. Durch eine Mittelwertbildung kann so der Fehler der Ergebnisse, der durch eine mögliche ungenaue Bewegungsausführung entstehen kann, minimiert werden.
5. Berechnung und Ausgabe aller erforderlichen Parameter.
6. Grafische Darstellung der Ergebnisse.
7. Speichern der Rohdaten, bzw. Export in ein universelles Dateiformat.

Abbildung 7 stellt die drei Hauptmodule der gesamten Methode dar. Die einzelnen Module können wiederum in verschiedene Submodule- bzw. routinen unterteilt werden. Ihr Aufbau bzw. ihre logische Struktur wird ausführlich in den jeweiligen Kapiteln beschrieben. Das Flussdiagramm, das die Struktur und logischen Zusammenhänge der für den gesamten Messablauf benötigten Einzelschritten abbildet, kann dem Anhang A auf Seite 123 entnommen werden.

Abbildung 7: Hauptmodule der gesamten Messmethode. Die Hauptmodule können in weitere Submodule- bzw. routinen unterteilt werden. Auf deren Struktur wird in den folgenden Kapitel bzw. dem Anhang näher eingegangen.

3.4.2 Konzeption der Berechnung der Rotationsachsen

Die logische Abfolge der zur Berechnung der Sprunggelenkachsen benötigten mathematischen Einzelschritte wird weitgehend von der Programmstruktur bzw. des praktisch durchzuführenden Messablaufs bei in vivo Messungen (Anhang A auf Seite 123) vorgegeben.
Der Prozess zur Berechnung der Sprunggelenkachsen gliedert sich dementsprechend in verschiedene Einzelschritte.

1. Berechnung eines *Tibiakoordinatensystems*. Die Ergebnisse der Sprunggelenkachsen wie z.B. Inklinations- und Deviationswinkel oder auch der minimale senkrechte Abstand der Achsen zum Kraftvektor der Achillessehne müssen, um eine bessere Vergleichsmöglichkeit zu gewährleisten, in einem relativ zu markanten anatomischen Punkten berechneten kartesischen Koordinatensystem ausgegeben werden (siehe Kapitel 3.4.3 auf Seite 25).
   Die zur Berechnung des *Tibiakoordinatensystems* benötigten markanten anatomischen Punkte werden in der „Neutral-Null-Stellung“[10] [Segesser und Nigg 1993] aufgezeichnet.

2. Aus den Markerkoordinatendaten, die die Berechnung des *Tibiakoordinatensystems* liefert, soll über den bekannten Abstand der Ultraschallsender 1 und 2 die Schallausbreitungsgeschwindigkeit bestimmt und ein Kalibrationsfaktor berechnet werden (Abhängigkeit der Schallgeschwindigkeit von der Temperatur, Kapitel 3.3.1 auf Seite 20).

3. Filtern der aufgezeichneten Ultraschallmarkerdaten. Berechnung der OSGA aus den Ultraschallmarkerdaten der Bewegung. Dabei erfolgt eine Bewegung

[10] Neutral-Null-Stellung [Segesser und Nigg 1993]: Aufrechter, hüftbreiter Stand des Probanden.

aus der Neutral-Null-Stellung in die maximale Dorsalflexion des Fußes. Für die Berechnung der unteren Sprunggelenkachse muss sichergestellt werden, dass es sich bei der aufgezeichneten Rotationsbewegung um eine reine Rotation um die Subtalarachse handelt, die nicht von einer Bewegung im oberen Sprunggelenk überlagert wird. Nach **Alt** [Alt 2001] ist dies bei einer maximalen Dorsalflexion gegeben. In Kapitel 3.7.3 auf Seite 76 soll dies in einer Voruntersuchung durch den Einsatz eines Magnet-Resonanz-Tomographen (MRT) betätigt werden. Transformation der Ergebnisse in das *Tibiakoordinatensystem*.

4. Filtern der aufgezeichneten Ultraschallmarkerdaten mit einem Butterworth-Lowpas-Filter. Berechung der unteren Sprunggelenkachse aus den Daten des Ultraschallmesssystems. Transformation der Ergebnisse in das *Tibiakoordinatensystem* und anschließende Rotation der USGA um die OSGA Achse mit dem gemessenen Winkel α (Winkel: Neutalposition-maximale Dorsalflexion).

5. Berechnung von Mittelwert und Standardabweichung der Ergebnisse.

Das Flussdiagramm (Abbildung 8) soll die notwendigen Abläufe der Berechnung visualisieren:

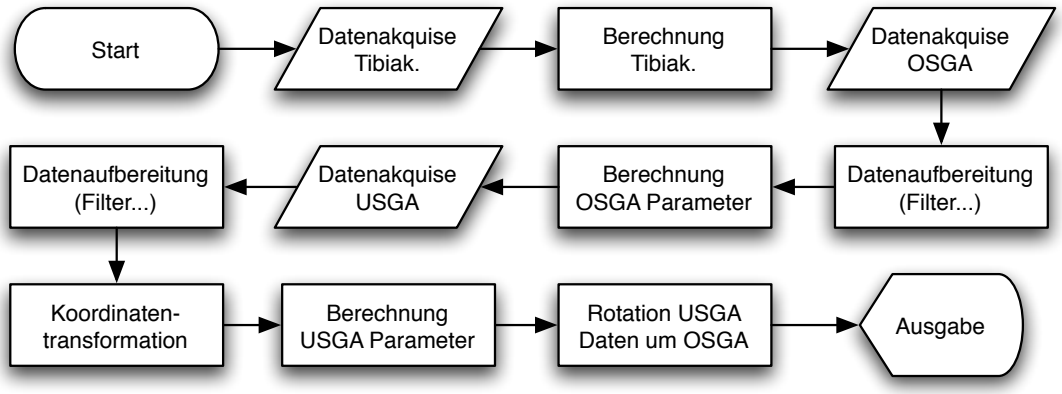

Abbildung 8: Prozessablauf zur Berechnung der Sprunggelenkachsen. 1. Datenakquisition; 2. Berechnung des *Tibiakoordinatensystems* und Temperaturkalibration des Ultraschallmesssystem; 3. Datenakquise der Daten zur Berechnung der OSGA; 4. Lowpassfiterung des Datenmaterials; 5. Koordinatentransformation in das *Tibiakoordinatensystem*; 6. Berechnung der Parameter der OSGA; 7. Datenakquise der Daten zur Berechnung der USGA; 8. Lowpassfilterung des USGA Datenmaterials; 9.Koordinatentransformation in das *Tibiakoordinatensystem*; 10. Berechnung der USGA Parameter; 11. Rotation der USGA um die OSGA mit dem Drehwinkel α (Neutralstellung-maximale Dorsalflexion); 12. Ausgabe und grafische Darstellung der Ergebnisse.

3.4.3 Koordinatensystem anatomischer Bezugspunkte

Wie bereits in Kapitel 3.4.2 auf Seite 23 beschrieben, wird für den interindividuellen Vergleich von verschiedenen Sprunggelenkachsenparametern ein geeignetes Koordinatensystem benötigt. Alle berechneten Parameter müssen vom Laborkoordinatensystem (gegeben durch die Koordinaten $x'y'z'$) in ein Koordinatensystem mit den Koordinaten xyz transferiert werden, das anhand von markanten anatomischen Punkten berechnet werden kann. Das Laborkoordinatensystem ist von der Hardware bzw. von der Anordnung und Befestigung der Ultraschallsensoren vorgegeben. Die Ultraschallsensoren sollen an der Tibiakante befestigt werden[11] (siehe Kapitel 3.6.1 auf Seite 62). Aufgrund der interindividuellen anatomischen Variabilität ist es nicht möglich, die Sensoren bei verschiedenen Probanden an identischen Stellen zu befestigen. Deshalb muss bei jedem Messvorgang eine Koordinatentransformation der Rohdaten in das sogenannte *Tibiakoordinatensystem* stattfinden. Alle Parameter müssen relativ zu diesem *Tibiakoordinatensystem* oder Koordinatensystem anatomischer Bezugspunkte angegeben werden, um so einen interindividuellen Vergleich der Achsparameter zu ermöglichen. Abbildung 35 auf Seite 63 zeigt den Ursprung wie auch die Koordinatenachsen des Zebris® *JMA* $11 \cdot 11$ Sensors. Dieser Sensor wird an der Tibiakante des Probanden angebracht (siehe Abbildung 37 auf Seite 66). Alle aufgezeichneten Daten müssen wie in Kapitel 3.4.4 beschrieben in das *Tibiakoordinatensystem* transformiert werden.

Um eine hohe Reliabilität der Messung zu gewährleisten, sollten möglichst markante anatomische Punkte gewählt werden. Da sich der *zufällige Gesamtmessfehler* ΔE aus den zufälligen Messfehlern der Messung ΔE_M und aus dem Fehler des Festlegens des *Tibiakoordinatensystems* ΔE_{xyz} ergibt, sollte schon bei der Bestimmung des *Tibiakoordinatensystems*, d. h. der Aufnahme dazu benötigten anatomischen Punkte, sehr sorgfältig gearbeitet werden. **Alt** [Alt 2001] hat für die Berechnung eines *Tibiakoordinatensystems* zwei Punkte a und b auf der Tibikante und einen Punkt c, der parallel zur Fußlängsachse ausgerichtet wurde, gewählt. Um den Fehler ΔE_{xyz} möglichst klein zu halten, wurde in dieser Arbeit der Punkt c auf der Achillessehnenmitte gewählt. Mögliche anatomische Besonderheiten des Vorfußes beeinflussen nicht die Berechnung des *Tibiakoordinatensystems*. Darüber hinaus erscheint das Markieren eines Punkes c auf der Achillessehnenmitte reproduzierbarer als das manuelle Ausrichten einer Strecke $\overline{ac}$ parallel zur Fußlängsachse. Die anatomischen Punkte werden mit einem speziell für diesen Zweck konstruierten Taststift erfasst (Siehe Kapitel 3.6.3 auf Seite 64).

[11] Der Sensor lässt sich relativ direkt auf der knöchernen Struktur der Tibia anbringen. Mögliche Messfehler verursacht durch Hautverschiebungen können so minimiert werden.

Abbildung 9: Festlegen des *Tibiakoordinatensystems* Schritt 1: Markieren des Punkts A auf der Tibiakante.

Mit Hilfe des Taststiftes wird ein erster Punkt A auf der Tibiakante markiert. Punkt A wird durch Ortsvektor $\vec{a}$ beschrieben. Dazu liegt der Ursprung des Laborkoordinatenssystems beim in vivo Versuch im Mittelpunkt der vom *JMA* 11·11 aufgespannten Fläche (siehe Abbildung 35 auf Seite 63) und wenige Zentimeter vor der Tibiakante, oberhalb des Punktes A.

$$A : \vec{a} = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} \tag{1}$$

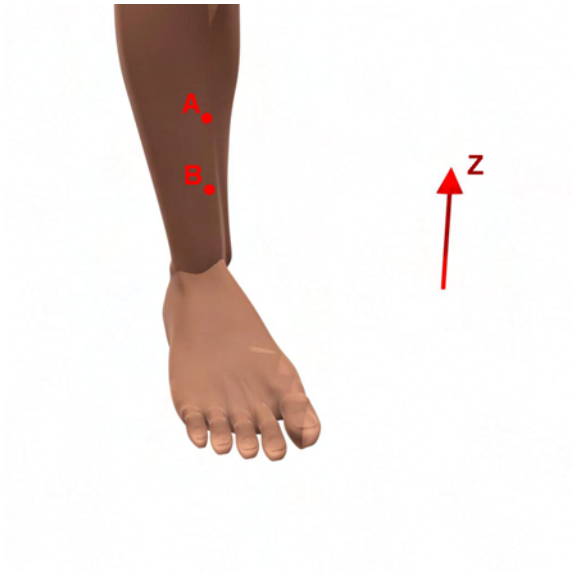

Abbildung 10: Festlegen des *Tibiakoordinatensystems* Schritt 2: Markieren des Punkts B auf der Tibiakante.

Punkt B liegt unterhalb von Punkt A auf der Tibiakante und wird durch Vektor $\vec{b}$ beschrieben. Vektor $\vec{z}$ kann aus den Vektoren $\vec{a}$ und $\vec{b}$ berechnet werden. Vektor $\vec{z}$ ist bereits parallel zur z-Achse des *Tibiakoordinatensystems*.

$$B : \vec{b} = \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} \tag{2}$$

$$\vec{z} = \vec{a} - \vec{b} \tag{3}$$

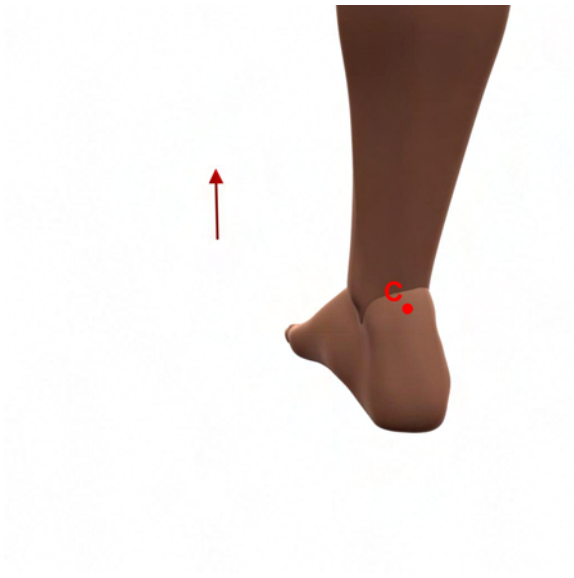

Abbildung 11: Festlegen des *Tibiakoordinatensystems* Schritt 3: Markieren des Punkts C auf der Achillessehnenmitte. Der Vektor $\vec{d}$ zeigt vom Punkt B durch den Unterschenkel auf den Punkt C.

Punkt C kann mit der zweiten Taststiftspitze (siehe dazu Abbildung 40 auf Seite 68) markiert werden. Punkt C wird durch Vektor $\vec{c}$ beschrieben. Der Vektor $\vec{d}$ kann darum mit Hilfe der Vektoren $\vec{b}$ und $\vec{c}$ der Punkte B und C berechnet werden, er zeigt vom Punkt B durch den Unterschenkel auf den Punkt C und wird in der Abbildung nicht dargestellt.

$$C : \vec{c} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \tag{4}$$

$$\vec{d} = \vec{c} - \vec{b} \tag{5}$$

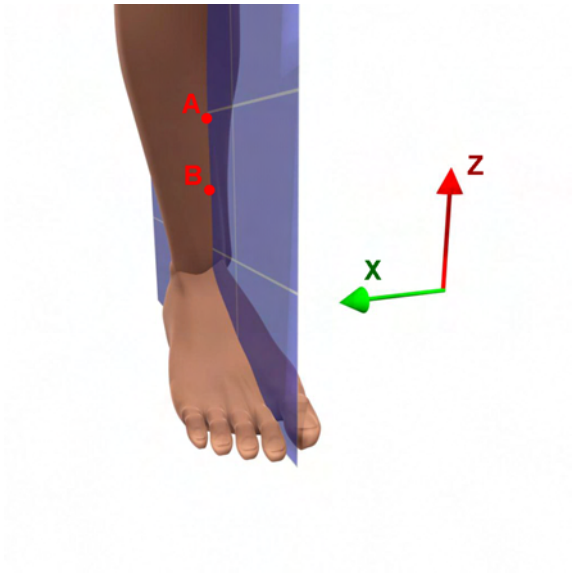

Abbildung 12: Festlegen des *Tibiakoordinatensystems* Schritt 4: Punkte A, B, C sowie die Vektoren $\vec{z}$ und $\vec{d}$ liegen in einer Ebene (Sagitalebene). Der Vektor $\vec{x}$ ist der Normalenvektor zu dieser Ebene.

Die Vektoren $\vec{d}$ und $\vec{z}$ liegen in einer Ebene (Sagitalebene). Aus deren Kreuzprodukt geht der Vektor $\vec{x}$ hervor der parallel zur X-Achse des *Tibiakoordinatensystems* ist.

$$\vec{x} = \vec{z} \times \vec{d} \tag{6}$$

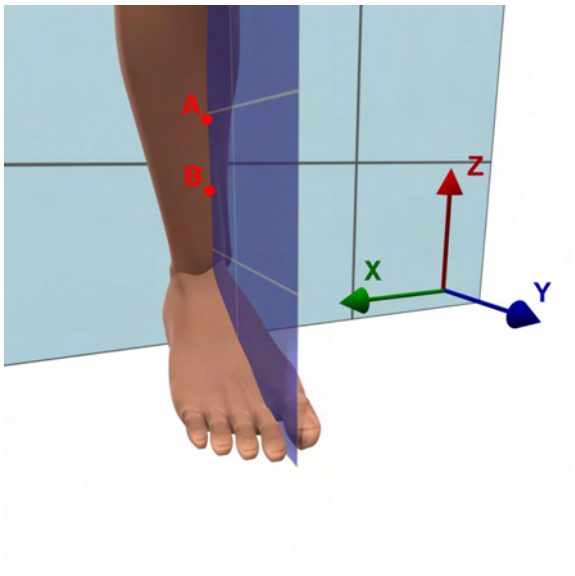

Abbildung 13: Festlegen des *Tibiakoordinatensystems* Schritt 5: Kreuzprodukt von $\vec{x}$ und $\vec{z}$ liefert den Normalenvektor $\vec{y}$ zur Frontalebene.

Vektor $\vec{z}$ und $\vec{x}$ stehen senkrecht aufeinander. Ein erneutes Kreuzprodukt von Vektor $\vec{z}$ und $\vec{x}$ liefert einen Vektor $\vec{y}$, der senkrecht zu den Vektoren $\vec{z}$ und $\vec{x}$ ist und parallel zur Y-Achse.

$$\vec{y} = \vec{z} \times \vec{x} \tag{7}$$

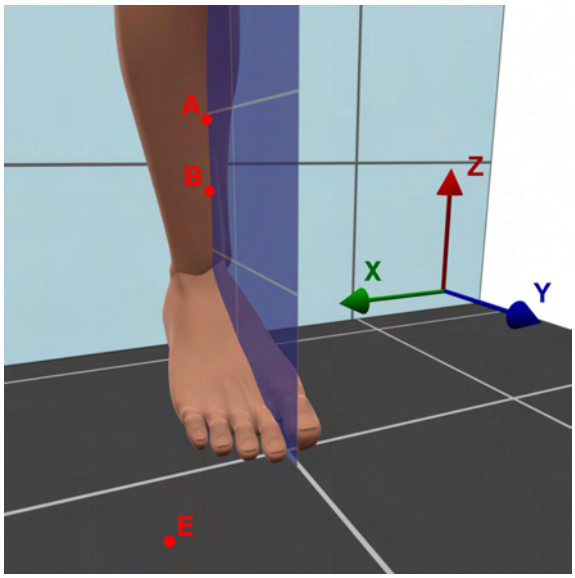

Abbildung 14: Festlegen des *Tibiakoordinatensystems* Schritt 6: Markieren eines beliebigen Punktes E auf der Standfläche des Probanden. Die Transversalebene ist durch die Vektoren $\vec{x}$ und $\vec{y}$ gegeben und wird in den Punkt E ,in die Standfläche, verschoben.

Als letzter Punkt wird Punkt E aufgezeichnet. E liegt in der Standfläche und ist durch Vektor $\vec{e}$ gegeben. Der Ursprung U des *Tibiakoordinatensystems* ist der in die Standfläche projizierte Punkt C (Punkt auf der Achillessehnenmitte, in der Abbildung nicht sichtbar).

$$E = \vec{e} = \begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix} \tag{8}$$

$$U = \vec{u} = \begin{pmatrix} x_c \\ y_c \\ z_e \end{pmatrix} \tag{9}$$

Damit sind alle nötigen Punkte und Vektoren des *Tibiakoordinatensystem*s berechnet:
U: Ursprung des *Tibiakoordinatensystems*
$\vec{x}$: X-Achse des *Tibiakoordinatensystems*
$\vec{y}$: Y-Achse des *Tibiakoordinatensystems*
$\vec{z}$: Z-Achse des *Tibiakoordinatensystems*

3.4.4 Koordinatentransformation

Alle vom System aufgenommenen Bewegungskoordinaten müssen vom Laborkoordionatensystem in das *Tibiakoordinatensystem* transformiert werden (siehe Kapitel 3.4.2 auf Seite 23. Um ein kartesisches Koordinatensystem mit der Basis $(x'y'z')$ in ein anderes kartesisches Koordinatensystem mit der Basis (xyz) zu transformieren, müssen folgende Schritte unternommen werden [Merziger und Wirth 1993]:
Aufstellen einer Abbildungsmatrix A mit Hilfe der Vektoren $\vec{x}$, $\vec{y}$, $\vec{z}$. Die Vektoren $\vec{x}$, $\vec{y}$, $\vec{z}$ repräsentieren die Koordinatenachsen des transformierten Koordinatensystems.

$$A = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix} \tag{10}$$

mit:

$$\vec{x}: \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}; \vec{y}: \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}; \vec{z}: \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}$$

Berechnen der Abbildungsmatrix B aus der inversen Matrix A.

$$B = A^{-1} = A^T \tag{11}$$

Um Punkte vom Laborkoordinatensystem in das transfomierte Koordinatenssytem zu überführen, müssen die Punkte in einem ersten Schritt um den Vektor $\vec{u}$, dem Vektor vom Ursprung des Laborkoordinatensystems zum Ursprung des transfomierten Koordinatenssytems, verschoben werden.

$$v' = v - \vec{u} \tag{12}$$

mit:

$\vec{v}$: der nicht transformierte Vektor im Laborkoordinatensystem
$\vec{v}'$: der transformierte Vektor (z.B. des *Tibiakoordinatensystem*s)
$\vec{u}$: Vektor vom Ursprung des Laborkkordinatensystem zum Ursprung des transformierten Koordinatensystems

Anschließend müssen diese verschobenen Punkte mit der Abbildungsmatrix B multipliziert werden.

$$v'' = B \cdot v' \tag{13}$$

3.4.5 Mathematische Methoden zur Bestimmung von Rotationsachsen

Es gibt verschiedene mathematische Methoden, um die Verschiebung und Verdrehung eines Starrkörpers zu beschreiben:

- ◇ **Euler/Kardan**-Winkel Rotation und Translation eines kartesischen Koordinatensystems, bei der die Drehung eines Körpers durch eine 3×3 Rotationsmatrix R beschrieben wird. Allgemeine Ortsveränderungen können so durch eine Translation mit einer anschließenden Sequenz von Rotationen beschrieben werden: $\vec{x} = R\vec{x}' + \vec{v}$. Soll die Rotationsachse berechnet werden, muss die Rotationsmatrix R bestimmt werden.

- ◇ **Helical Axis** („screw axis“, „Schraubungsachse“, „equivalent screw displacement“). Die Schraubungsachse wird durch vier Größen eindeutig beschrieben: Ihre Orientierung im Raum durch einen Vektor $\vec{n}$ und einen Ortsvektor $\vec{r}$, einen Drehwinkel ϕ , und der Ganghöhe t. Drei nichtlinear angeordnete Punkte sind für dieses Verfahren notwendig, aus deren x, y, z sowie den abgebildeten x', y', z' Koordinaten, Ortsvektor $\vec{n}$ und Richtungsvektor $\vec{r}$ berechnet werden können. Mithilfe des **Helical Axis** können Translationsbewegungen entlang der Rotationsachse berechnet werden. Berechnungen von reinen Translationsbewegungen oder auch Translationsbewegungen die nicht parallel zur Rotationsachse erfolgen, sind aus nur zwei Datensätzen der x, y, und z Koordinaten nicht möglich[12].

[12] Reine Translationsbewegungen oder Rotationen, überlagert von Translationsbewegungen, die nicht parallel zur Rotationsachse erfolgen, lassen sich durch eine Kombination aus mehreren Schraubenbewegungen abbilden (Theorem von Chasles [Beggs 1966]). Der tatsächliche Bewegungsablauf kann durch eine Anzahl von n einzelner Schraubenbewegungen angenähert werden. Für n gegen unendlich ergibt sich eine kontinuierliche Folge von unendlich vielen, verschwindend kleinen Schraubenbewegungen, welche den tatsächlichen Bewegungsablauf ohne Berücksichtigung der Zwischenpositionen, exakt wiedergeben.

Handelt es sich um eine reine Rotationsbewegung[13], wird durch diese holonome Zwangsbedingung, die Anzahl der Freiheitsgrade reduziert [Kuypers 1989][14].

Geht man von einer reinen Rotationsbewegung ohne translatorischen Anteil aus[15], kann die Bewegung auf eine Rotationsbewegung in der Ebene beschränkt werden. Dadurch verringert sich der Freiheitsgrad eines N-Punkt-massensystems ohne Zwang auf $2N$. Damit benötigt man zur Berechnung von Rotationsachsen lediglich einen einzigen Marker, im Gegensatz zu den auf Seite 30 beschriebenen Methoden 3.4.5 der **Rotationsmatrix** oder dem **Helical Axis** Verfahren. Durch das Verringern der Anzahl der Marker kann gegebenenfalls auch der Messfehler durch Verschiebungen bzw. Vibrationen der Marker verkleinert werden.

3.4.6 Berechnung von Rotationsachsen mit dem Einpunktverfahren

Wie bereits erwähnt wird bei der Berechnung von Rotationsachsen unter Zuhilfenahme von nur einem Markerpunkt von der Annahme ausgegangen, dass es sich bei der Rotationsbewegung um eine Rotation in der Ebene mit nur $2N$ Freiheitsgraden handelt, also keine translatorischen Anteile vorhanden sind. Nachfolgende Abbildungen und Gleichungen sollen die Vorgehensweise des Einpunktverfahrens zur Berechnung von Rotationsachsen erläutern[16].

[13] Wenn weder eine Translation in Richtung der Rotationsachse (**Helical Axis**) noch eine von einer Translationsbewegung überlagerten Rotation vorkommt.

[14] Als Zwangsbedingung wird in der klassischen Mechanik eine Einschränkung der Bewegungsfreiheit eines Systems aus N Teilmassen bezeichnet. Dadurch nimmt die Anzahl der Freiheitsgrade des Systems ab: k unabhängige holonome Nebenbedingungen reduzieren die Zahl der Freiheitsgrade auf $3N - k$. Holonome Zwangsbedingungen beschreiben eine Beschränkung der Bewegungsfreiheit auf Flächen. Eine gleichzeitige Beschränkung auf zwei unterschiedliche, sich schneidende Flächen ist gleichbedeutend mit einer Beschränkung auf die Kurve, in der sich die Flächen schneiden. [Kuypers 1989]

[15] In der Literatur wird das Vorhandensein einer **Helical Axis** kontorvers diskutiert (siehe Kapitel 2 auf Seite 8).

[16] Die Vektoren werden in allen nachfolgenden Schaubildern des Kapitels als Geraden dargestellt. Auf die Vektorpfeile wurde der besseren Übersichtlichkeit wegen verzichtet.

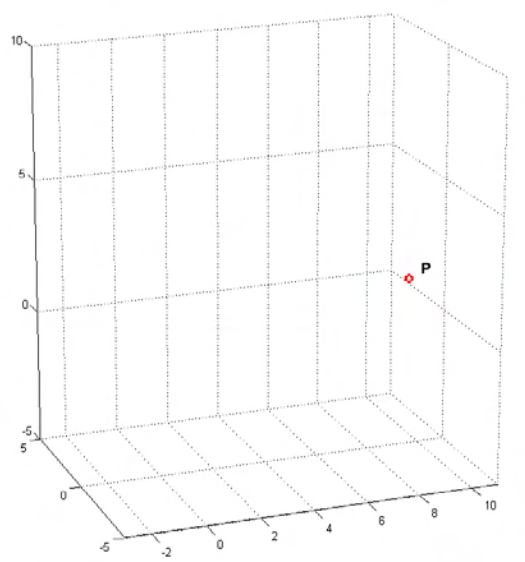

Abbildung 15: Einpunktverfahren Schritt 1, eine erste Momentaufnahme liefert den Punkt P.

Der Punkt P wird durch den Vektor $\vec{p}_1$ beschrieben:

$$P : \vec{p}_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{14}$$

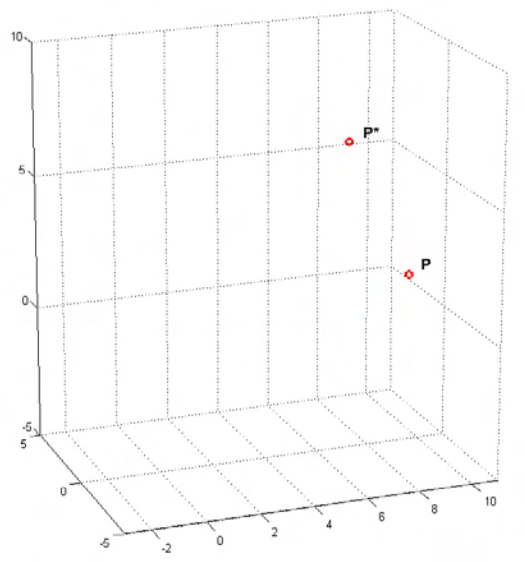

Abbildung 16: Einpunktverfahren Schritt 2, der Punkt P wird durch eine Rotation in den Punkt $P^\star$ abgebildet.

Der Punkt $P^\star$ wird durch den Vektor $\vec{p}_2$ beschrieben:

$$P^\star : \vec{p}_2 = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \tag{15}$$

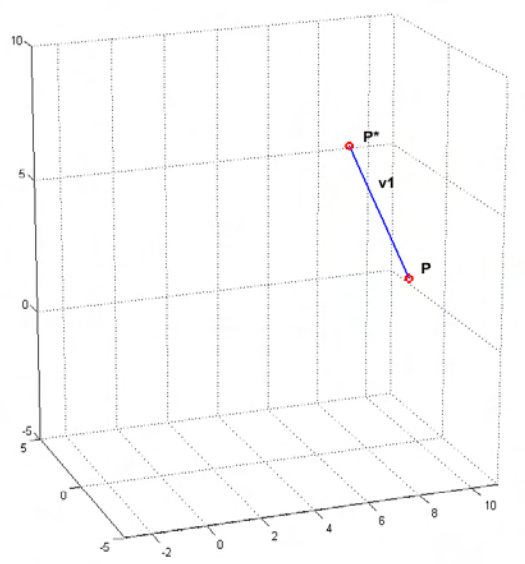

Abbildung 17: Einpunktverfahren Schritt 3, die Punkte P und $P^{\star}$ beschreiben den Vektor $\vec{v}_1$.

Der Vektor $\vec{v}_1$ kann durch die Koordinaten bzw. den Vektoren der Punkte P und $P^{\star}$ berechnet werden:

$$\vec{v}_1 = \vec{p}_2 - \vec{p}_1 \tag{16}$$

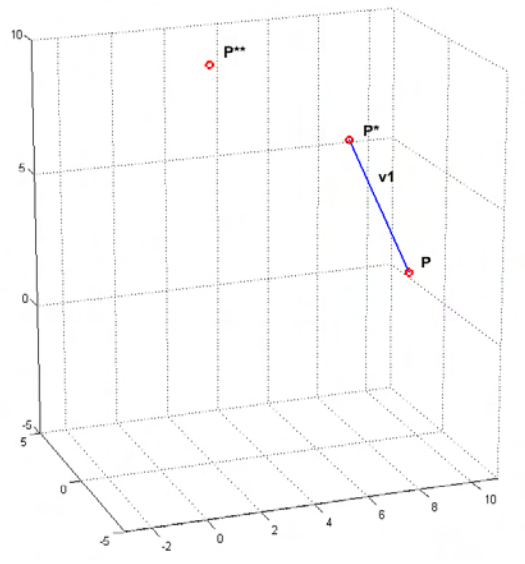

Abbildung 18: Einpunktverfahren Schritt 4, der Punkt $P^{\star\star}$ geht aus der Fortsetzung der Rotation des Punktes $P^{\star}$ hervor.

Der Punkt $P^{\star\star}$ wird durch den Vektor $\vec{p}_3$ beschrieben:

$$P^{\star\star} : \vec{p}_3 = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} \tag{17}$$

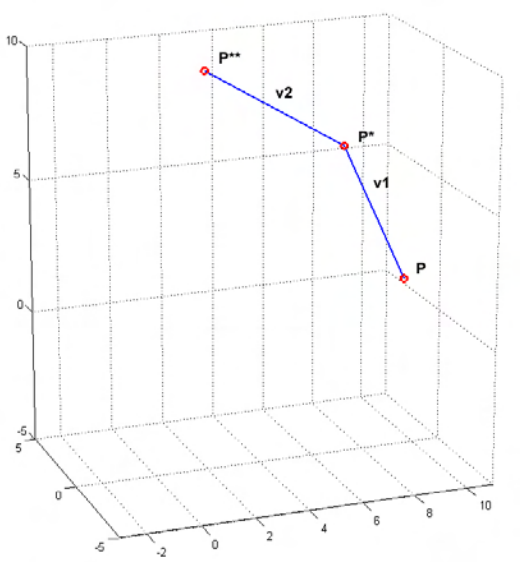

Abbildung 19: Einpunktverfahren Schritt 5, die Punkte $P^{\star}$ und $P^{\star\star}$ beschreiben den Vektor $\vec{v}_2$.

Der Vektor v_2 wird durch die Koordinaten bzw. den Vektoren der Punkte $P^{\star}$ und $P^{\star\star}$ berechnet:

$$\vec{v}_2 = \vec{p}_3 - \vec{p}_2 \tag{18}$$

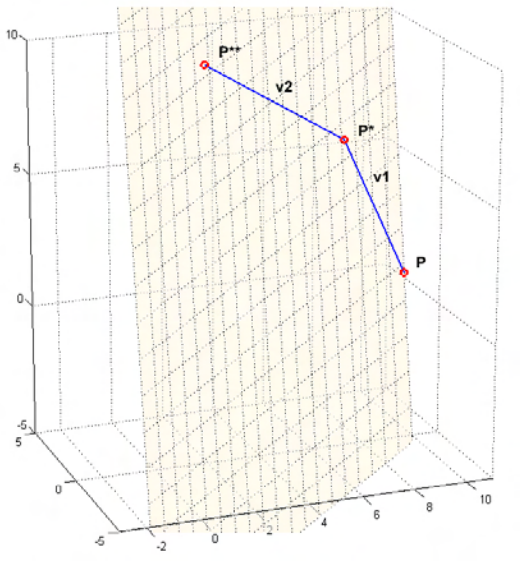

Abbildung 20: Einpunktverfahren Schritt 6, die Punkte P, $P^{\star}$ und $P^{\star\star}$ sowie die Vektoren $\vec{v}_1$ und $\vec{v}_2$ liegen in der Ebene E.

Da es sich um eine Rotationsbewegung handelt, müssen die Punkte P, $P^{\star}$ und $P^{\star\star}$ sowie die Vektoren $\vec{v}_1$ und $\vec{v}_2$ in der Ebene E liegen.

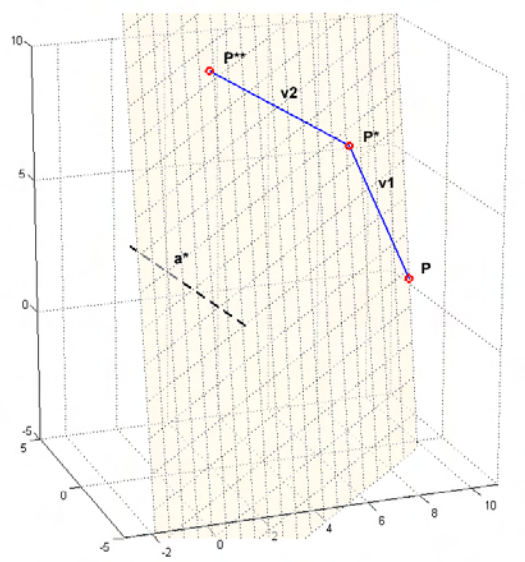

Abbildung 21: Einpunktverfahren Schritt 7, der Normalenvektor $\vec{a}^{\star}$ der Ebene E ist eine Parallele zur Rotationsachse.

Der Normalenvektor $\vec{a}^{\star}$ der Ebene E wird durch das Kreuzprodukt aus den Vektoren $\vec{v}_1$ und $\vec{v}2$ berechnet [Merziger und Wirth 1993]. Dieser Normalenvektor ist bereits parallel zur gesuchten Rotationsachse, d.h. die räumliche Orientierung ist schon bekannt, nicht aber die Position.

$$\vec{a}^{\star} = \frac{\vec{v}_1 \times \vec{v}_2}{|\vec{v}_1 \times \vec{v}_2|} \tag{19}$$

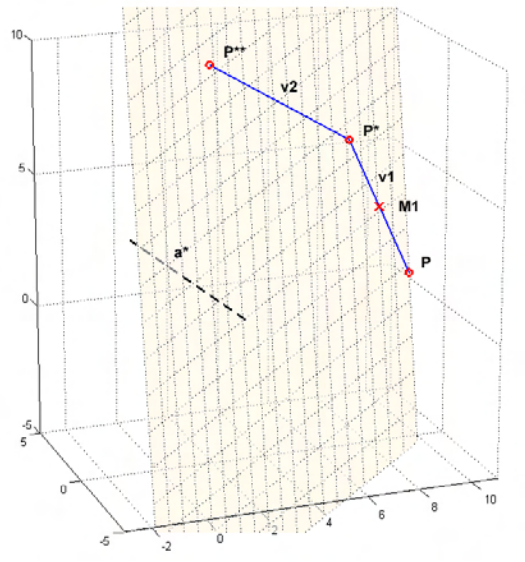

Abbildung 22: Einpunktverfahren Schritt 8, der Mittelpunkt M_1 liegt auf Vektor $\vec{v}_1$ und halbiert die Strecke zwischen den Punkten P und $P\star$.

Der Mittelpunkt M_1 auf Vektor $\vec{v}_1$ zwischen den Punkten P und $P^{\star}$ wird wie folgt berechnet:

$$M_1 : \vec{m}_1 = \vec{p}_1 + \frac{1}{2} \cdot \vec{v}_1 \tag{20}$$

mit:
$\vec{p}_1$: Vekor zum Punkt P

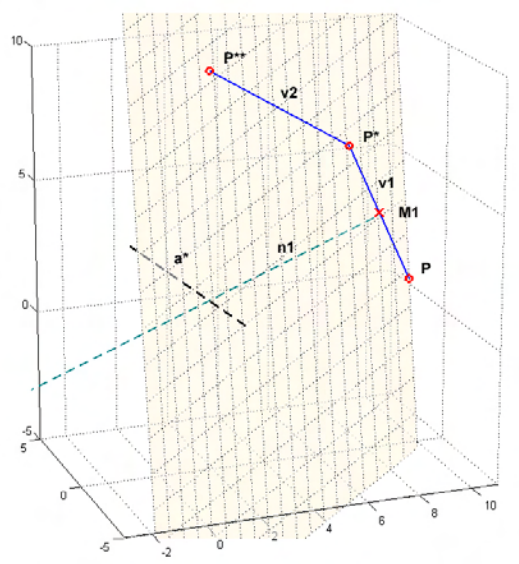

Abbildung 23: Einpunktverfahren Schritt 9, die Gerade n_1 liegt in der Ebene E, geht durch den Punkt M_1 auf Vektor $\vec{v}_1$ und ist senkrecht zu Vektor $\vec{v}_1$.

Da es sich um eine Rotationsbewegung handelt, muss der Abstand von Punkt P zur Rotationsachse gleich dem von Punkt $P^\star$ sein. Damit muss das Rotationszentrum der Punkte P und $P^\star$ auf einer Geraden liegen, die in der Ebene E liegt und durch den Punkt M_1 senkrecht zu Vektor $\vec{v}_1$ geht. Der Richtungsvektor $\vec{r}_{n1}$ der Geraden n_1 wird aus einem Kreuzprodukt von Vektor $\vec{v}_1$ und $\vec{a}^\star$ berechnet:

$$\vec{r}_{n1} = \vec{v}_1 \times \vec{a}^\star \tag{21}$$

Mit Punkt M_1 als Aufpunkt erhält man als vollständige Geradengleichung für n_1:

$$n_1 : \vec{x} = \vec{m}_1 + \lambda \cdot \vec{r}_{n1} \tag{22}$$

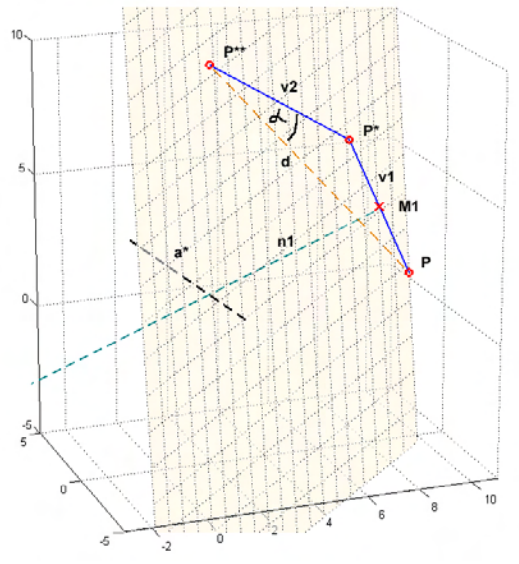

Abbildung 24: Einpunktverfahren Schritt 10, der Vektor $\vec{v}_2$ und der Vektor $\vec{d}$ schließen den Winkel α ein.

Der Vektor $\vec{d}$ kann aus den Koordinaten der Punkte P und $P^{\star\star}$ berechnet werden:

$$\vec{d} = \vec{p}_2 - \vec{p}_1 \tag{23}$$

mit:
$\vec{p}_1$: Vektor zum Punkt P
$\vec{p}_2$: Vektor zum Punkt $P^{\star\star}$
Das Skalarprodukt der Vektoren $\vec{d}$ und $\vec{v}_2$ liefert den Winkel α [Merziger und Wirth 1993]:

$$\cos\alpha = \frac{\vec{v}_2 \cdot \vec{d}}{|\vec{v}_2| \cdot |\vec{d}|} \tag{24}$$

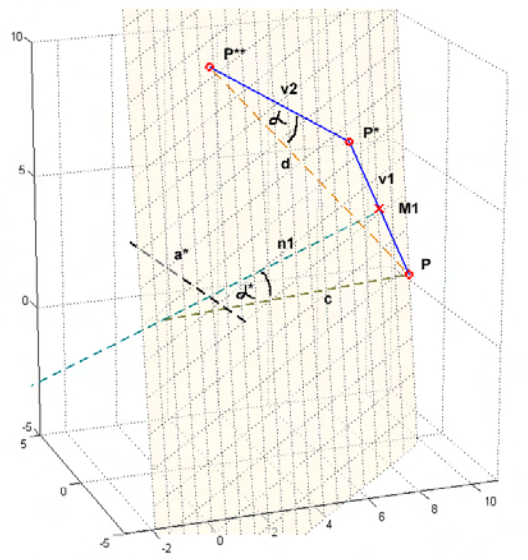

Abbildung 25: Einpunktverfahren Schritt 11, die Gerade n_1 und c schließen den Winkel $\alpha^\star$ ein.

Die Gerade c ist gegeben durch die Vektoren $\vec{p}_1$ und $\vec{c}$:

$$c : \vec{x} = \vec{p}_1 + \gamma \cdot \vec{c} \tag{25}$$

Der Winkel $\alpha^\star$, der von der Geraden n_1 und c eingeschlossen wird, ist nach dem *Satz vom Umfangswinkel* gleich dem Winkel α (siehe Anhang B auf Seite 124):

$$\alpha^\star = \alpha \tag{26}$$

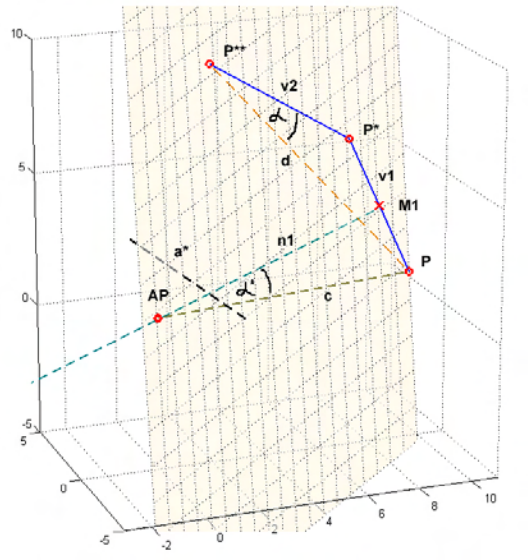

Abbildung 26: Einpunktverfahren Schritt 12, die Gerade n_1 und v_1 schließen einen rechten Winkel ein.

Die Gerade v_1 kann mithilfe der Vektoren $\vec{p}_1$ und $\vec{v}_1$ berechnet werden:

$$v_1 : \vec{x} = \vec{p}_1 + \tau \cdot \vec{v}_1 \tag{27}$$

Im rechtwinkligen Dreieck AP, M_1, P kann mithilfe der Vektoren $\vec{r}_{n1}$, $\vec{c}$ und $\vec{v}_1/2$, dem Winkel $\alpha^\star$ und einfachen trigonometrischen Funktionen die Länge l der Strecke $\overline{M_1 AP}$ vom Punkt M_1 zum Punkt AP berechnet werden :

$$\tan\alpha = \frac{|\vec{v}_1|}{2 \cdot l}$$

$$l = \frac{|\vec{v}_1|}{2 \cdot \tan\alpha} \tag{28}$$

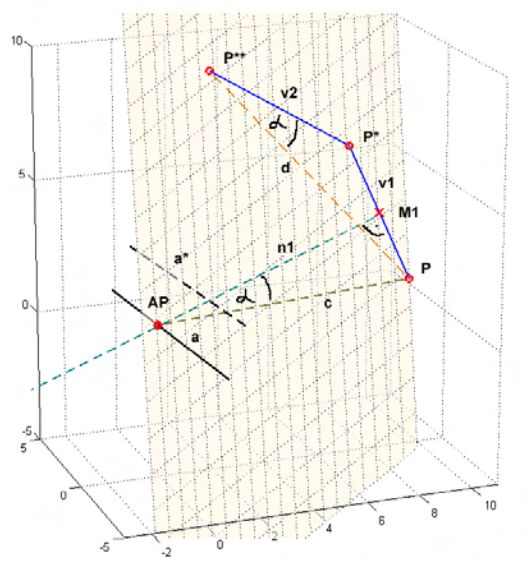

Abbildung 27: Einpunktverfahren Schritt 14, die Rotationsachse g geht durch den Punkt AP und ist senkrecht zur Ebene E.

Die Länge l der Strecke $\overline{M_1 AP}$ ist nun bekannt. Damit kann der Punkt AP mit Hilfe des normierten Vektors $\vec{n}_{n1}$, dem Vektor $\vec{v}_1$ und $\vec{p}_1$ konstruiert werden:

$$\vec{n}_{n1} = \frac{\vec{n}_1}{|\vec{n}_1|} \tag{29}$$

$$\vec{ap} = \vec{p}_1 + \frac{1}{2} \cdot \vec{v}_1 + l \cdot \vec{n}_{n1} \tag{30}$$

Die Geradengleichung der Rotationsachse g ist demnach:

$$g : \vec{r} = \vec{ap} + \lambda \cdot \vec{a} \tag{31}$$

mit:
λ: Streckungsfaktor

3.4.7 Rotationsmatrix

Da die Messung der unteren Sprunggelenkachse in einer maximalen Dorsalflexion durchgeführt wird, müssen alle berechneten Ergebnisse um den Winkel der maximalen Dorsalflexion zurück rotiert werden (wie in Kapitel 3.4 auf Seite 22 beschrieben). Als Rotationsachse soll dazu die obere Sprunggelenkachse verwendet werden. Für die Berechnung einer Rotationsmatrix $R_{(\delta)}$ werden zwei Punkte P_1 und P_2 der Rotationsachse und der Drehwinkel α benötigt. Schritt 1, Berechnung des Vektors $\vec{v}$ aus den Punkten P_1 und P_2:

$$\vec{v} = P_2 - P_1 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} \tag{32}$$

Die Länge des Vektors $\vec{v}$ lautet:

$$|\vec{v}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{33}$$

Die Komponenten des zugehörigen Einheitsvektors:

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{34}$$

lauten daher:

$$a = \frac{x_2 - x_1}{|\vec{v}|}; b = \frac{y_2 - y_1}{|\vec{v}|}; c = \frac{z_2 - z_1}{|\vec{v}|} \tag{35}$$

Die Rotationsmatrix R wird wie folgt berechnet [Merziger und Wirth 1993]:

$$R_{(\delta)} = (1-\cos\alpha)\cdot \begin{pmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{pmatrix} + \cos\alpha \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \sin\alpha \cdot \begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix} \tag{36}$$

Folgende Schritte sind notwendig, um die Koordinatentransformation einzelner Punkte durchzuführen. Der Punkt U muss um einen Vektor $\vec{p_1}$, in diesem Fall um den Vektor zum Aufpunkt $P1$ der Rotationsachse, verschoben werden.

$$U^\star = U - \vec{p_1} \tag{37}$$

Der Punkt $U^{\star\star}$ geht aus einer Multiplikation des Punktes $U^\star$ mit der Rotationsmatrix $R_{(\delta)}$ hervor.

$$U^{\star\star} = R_{(\delta)} \cdot U^\star \tag{38}$$

Der rotierte Punkt $U^{\star\star}$ muss in einem letzten Schritt wieder um den Vektor $\vec{p_1}$ zurück verschoben werden.

$$U^{\star\star\star} = U^{\star\star} + \vec{p_1} \tag{39}$$

$U^{\star\star\star}$ ist der um die Rotationsachse mit dem Winkel α rotierte Punkt. Die Gleichungen 37, 38 und 39 können zu einer Gleichung 40 zusammengefasst werden. Jeder Punkt W kann so in einen Punkt $W^\star$ transformiert werden.

$$W^\star = ((W - \vec{p_1}) \cdot R_{(\delta)}) + \vec{p_1} \tag{40}$$

mit:

$R_{(\delta)}$:	In Gleichung 36 berechnete Rotationsmatrix.
W:	Zu transformierende Punkt.
$W^{\star}$:	Transformierter Punkt
$\vec{p}_1$:	Vektor zum Aufpunkt P_1 der Rotationsachse

3.5 Softwareentwicklung

Die Software zur Bestimmung der Sprunggelenkachsen wurde mit *Microsoft Visual C++®* entwickelt. Bei C++ handelt es sich um eine effiziente, maschinennahe Programmiersprache, die geeignet ist, zeitkritische Abläufe und Prozesse zu bearbeiten. Die objektorientierte Programmierung von C++ erlaubt es außerdem, eigene Klassen zu entwickeln. Vorteile sind die höhere Modularisierung des Codes, somit eine höhere Wartbarkeit[17] und Wiederverwendbarkeit der Einzelmodule sowie eine höhere Flexibilität des gesamten Programms. Zusätzlich zu den bereits in *Microsoft Visual C++®* enthaltenen Standardpaketen wurden noch folgende Klassen von Drittanbietern eingebunden:

- ◇ Die kommerzielle Klasse bzw. DLL ZEBSDK der Firman Zebris®. Diese Sammlung an C++ Klassen und DLLs übernimmt die Ansteuerung sowie den Dateneinzug verschiedener Hardwarekomponenten, wie z. B. dem *CMS* [Zebris Medizintechnik 2004].

- ◇ Die Open Source Klasse XMath [Dougherty 2002]. Diese Klasse übernimmt alle benötigten Vektor- und Matrizenrechnungen.

- ◇ Die Opensource Klasse NTGraph3d [Teofilov 2003]. Diese Klasse ist in der Lage, dreidimensionale Flächen und Funktionen darzustellen. Die Klasse NTGrpah3D wurde modifiziert, um sowohl dreidimensionale Funktionen als auch OBJ Dateien[18] darzustellen.

Alle weiteren Klassen und Programmdialoge wie auch die Benutzeroberfläche (GUI) mussten selbständig entwickelt werden. Mit den Klassen *calcAxis* und *zebClass* wurden zwei Klassen entwickelt, die ohne weiteres in fremde Programme eingebunden werden können. Alle Eigenschaften und Methoden stehen dann diesen Programmen zur Verfügung. Die Entwicklung neuer Programme, die sich mit einer ähnlichen Problematik beschäftigen bzw. dieselbe Hardware ansteuern müssen, kann so erheblich vereinfacht werden. Deshalb werden diese Klassen mit ihren Eigenschaften und Methoden nachfolgend erläutert.

- ◇ Die Klasse zebClass übernimmt die Hardwareansteuerung und Datenakquise. Greift auf die ZEBSDK.DLL zu.

[17] Kriterium der Softwareentwicklung, gibt an, mit welchem Aufwand und welcher Qualität Änderungen an einem bestehenden Softwaresystem durchgeführt werden können.

[18] OBJ Dateien repräsentieren ein gängiges Dateiformat, um dreidimensionale Körper darzustellen.

- ◇ Die Klasse calcAxis berechnet die Gelenkachsen des oberen- und untern Sprungelenks. Sie führt alle dazu notwendigen Rechenoperationen wie Koordinatentransfomationen, Rotationen, Eichfaktorberechung usw. durch. Nutzt die Klasse XMath für alle Vektor und Matrizenrechnungen.

- ◇ Vom Hauptprogramm USG werden alle Teilprogramme und Klassen zusammengefasst. Alle Dialoge zur Koordinatenachsenberechnung, Sprunggelenkachsenbestimmung, Probandendaten aber auch Teile der Datenverarbeitung werden von diesem Programm durchgeführt.

3.5.1 Die Klasse zebClass

Die Ansteuerung der Hardware erfolgt über die Klasse *zebClass*. Diese Klasse bindet die von der Firma Zebris® zur Verfügung gestellten DLL ein. Mit diesem Befehlssatz ist es möglich, das CMS 20, aber auch andere Ultraschall-Laufzeit-Bewegungsanalyse-Systeme der Firma Zebris®anzusteuern. Die *zebClass* fasst verschiedene Schritte bzw. Befehle, die zur Ansteuerung der Hardware notwendig sind, zu einzelnen Methoden zusammen. Der Programmcode der C++ Klasse *zebClass* kann dem Anhang auf Seite 139 entnommen werden.

3.5.1.1 Methoden der Klasse zebClass

Die Funktion *zebConf* öffnet den Konfigurationsdialog der Zebris®Hardware. Hier können verschiedene Hardwaretypen, Schnittstellen oder auch Sensorentypen definiert werden. Bisher werden das CMS 20 und JMA20 unterstützt. Der Anschluss kann sowohl über die RS232 als auch die USB Schnittstelle erfolgen. Als Sensorentypen stehen die WinJaw Sensoren JMA11x11 und JMA8x8 zur Verfügung, wobei der JMA8x8 wegen seiner geringeren Abmessungen und damit verbundenen geringeren mechanischen Störanfälligkeit zu präferieren ist.

```
zebConf()
```

zebHdwInit initialisiert die Hardware. Hierbei wird ein Port zur Hardware geöffnet und verschiedene Parameter wie z. B. die Einzugsfrequenz, Anzahl der einzuziehenden Analog- oder Digitalkanäle und Name des Sensortyps übergeben. Dieser Befehl muss beim erstmaligen Start des Programms ausgeführt werden.

```
zebHdwInit()
```

zebStart startet die Hardware, d. h. schaltet die Ultraschallsender an. Zu hören ist dann das typische Geräusch der gepulsten Ultraschallsignale.

```
zebStart()
```

zebGetSingle liest einzelne Messwerte aus dem Hardwarebuffer aus und weisst diese den Variablen *zebX*1 bis *zebZ*4 der Analogkanäle sowie den Variablen $ON1$ und $ON2$ der zwei Digitalkanälen zu.

```
zebGetSingle()
```

zebStop() stoppt die Hardware, d. h. schaltet die Ultraschallsender aus. Der Port zu Hardware bleibt aber geöffnet, das bedeutet, der Befehl *zebHdwInit* kann nicht erneut ausgeführt werden. Erst nach dem Befehl *zebHdwRelease* kann die Hardware erneut initialisiert werden.

```
zebStop()
```

zebHdwRelease schließt den Port zur Hardware. Soll eine Veränderung der Konfiguration vorgenommen werden, muss der Befehl *zebHdwrelease* ausgeführt werden, erst dann ist einer Reinitialisierung möglich. *zebStatus*

```
zebHdwRelease()
```

3.5.1.2 Eigenschaften der Klasse zebClass

Die public Variable *zebStatus* beinhaltet den aktuellen Status der Hardware. Mögliche Inhalte sind z.B. „Hardware wird initialisiert“ oder „Hardware wird gestartet“[19]. *zebStatus*

```
CString zebStatus
```

Nachfolgende Variablen enthalten die X, Y und Z Koordinaten der vier Analogkanäle.

```
double zebX1;
double zebX2;
double zebX3;
double zebX4;
double zebY1;
double zebY2;
double zebY3;
double zebY4;
double zebZ1;
double zebZ2;
```

[19] Der aktuelle Status der Harware wird allerdings nur für Debuggingzwecke benötigt und ist für den Betrieb der Messsoftware nicht von Bedeutung.

```
double zebZ3;
double zebZ4;
```

Die zwei Variablen $ON1$ und $ON2$ enthalten den Schaltzustand der zwei Digitalkanäle.

```
bool ON1;
bool ON2;
```

Um eine fehlerfreie Funktion der Hardware zu gewährleisten muss eine bestimmte Sequenz der Befehle eingehalten werden (siehe Abbildung 28 auf Seite 44).

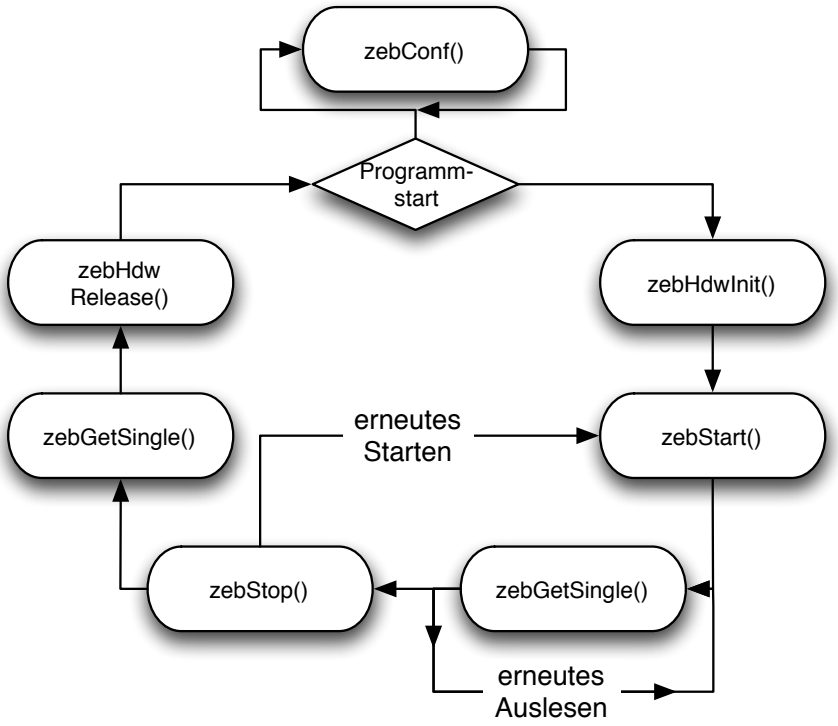

Abbildung 28: Befehlssequenz der Klasse *zebClass*. Bei einem ersten Start der Software muss die Hardware mittels der Funktion *zebHdwInit* initiiert werden. Nach dem Start der Hardware durch den Befehl *zebStart* können die erfassten Koordinatendaten den Variablen mittels der Methode *zebGetSingle* zugewiesen werden. Dieser Vorgang kann beliebig oft wiederholt werden. Nach einem Stoppen durch *zebStop* ist ein erneutes Starten ohne erneute Initialisierung der Hardware möglich. Der Aufruf von *zebConf* ist optional. Die Hardwarekonfiguration wird von dieser Funktion in einer Textdatei gespeichert, die von *zebHdwInit* bei jeder Initialisierung aufgerufen wird. Wird an dieser Konfiguration nichts geändert, muss *zebConf* nicht erneut aufgerufen werden. Die Klasse *zebClass* ermöglicht das Ansteuern und Auslesen jeder Zebris® -Hardware durch den Einsatz von nur drei Befehlen.

Die *zebSDK.dll* stellt noch weitere Befehle zur Ansteuerung verschiedener Hardware der Firma Zebris®zur Verfügung, auf die hier nicht weiter eingegangen werden soll [Zebris Medizintechnik 2004].

3.5.2 Die Klasse calcAxis

Die Klasse *calcAxis* verfügt über alle Eigenschaften und Methoden, die notwendig sind, um Rotationsachsen zu berechnen. Bei der Entwicklung wurde darauf geachtet dass die Klasse universal einsetzbar ist, das bedeutet, die Klasse kann nicht nur für das Problem der Sprunggelenkachsenbestimmung eingesetzt werden, vielmehr ist sie für die Bestimmung von Gelenkachsen bzw. Rotationsachsen allgemein geeignet. Die Klasse *calcAxis* unterscheidet dabei zwei Typen von Achsen (dazu Kapitel 3.4.7 auf Seite 38 sowie Abbildung 8 auf Seite 24).

- ◇ Rotationsachsen, die direkt aus den Koordinatendaten berechnet werden.
- ◇ Rotationsachsen die anschließend eine Koordinatentransformation durchlaufen. Im Fall der USG Achse wird diese nach Bestimmung ihrer räumlichen Position und Lage um die ermittelte OSG Achse rotiert.

Optional kann ein alternatives Koordinatensystem wie z. B. das *Tibiakoordinatensystem*, festgelegt werden. Wird ein alternatives Koordinatensystem festgelegt, werden alle Koordinatendaten in dieses Koordinatensystem transformiert (beschrieben in Kapitel 3.4.7 auf Seite 38). Die Berechnung der Schallausbreitungsgeschwindigkeit über den bekannten Abstand zweier Punkte P_1 und P_2 ist ebenfalls optional.
Im nachfolgenden Abschnitt werden die wichtigsten Eigenschaften und Methoden der Klasse *calcAxis* beschrieben. Der vollständige Quellcode der Klasse *calcAxis* sowie die Syntax der einzelen Methoden kann dem Anhang C auf Seite 125 entnommen werden.

3.5.2.1 Methoden der Klasse calcAxis

calcAxis initiiert eine Variable vom Typ *calcAxis*. Dieser Variablen vom Typ *calcAxis* stehen alle Methoden und Eigenschaften der Klasse zur Verfügung.

```
calcAxis()
```

calcTemp berechnet den Faktor f_{temp} „Temperaturabhängikeit der Schallgeschwindigkeit“ aus dem tatsächlichen Abstand $l = 50mm$ und dem gemessenen Abstand s

der Marker M_1 und M_2 des Taststiftes.

$$f_{temp} = \frac{50}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}} \tag{41}$$

mit den Koordinaten der Marker M_1: $\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$ und M_2: $\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$

Der Methode *calcTemp* müssen die x, y und z Koordinaten beider Marker M_1 und M_2 übergeben werden.

```
calcTemp(double x1, double y1, double z1,
         double x2, double y2, double z2)
```

coordTibia berechnet ein alternatives Koordinatensystem[20]. Übergeben werden alle zur Berechnung benötigten Punkte (wie beschrieben in Kapitel 3.4.3 auf Seite 25). Übergeben werden die x, y und z Koordinaten der Marker M_1 und M_2 von vier Punkten.

```
coordTibia(double x1, double y1, double z1,
           double x2, double y2, double z2,
           double x3, double y3, double z3,
           double x4, double y4, double z4,
           double x5, double y5, double z5,
           double x6, double y6, double z6,
           double x7, double y7, double z7,
           double x8, double y8, double z8)
```

transform wird nur aufgerufen, wenn ein alternatives Koordinatensystem festgelegt wurde (Transformation wie beschrieben in Kapitel 3.4.4 auf Seite 29). Übergeben wird der Zeiger auf den zu transformierenden Vektor der Klasse *XMath*.

```
transform(UX_VECTOR3 *vector)
```

calcEV berechnet Aufpunkt und Richtungsvektor sowie den Drehwinkel der Rotationsachse. Übergeben werden die Vektoren dreier Punkte sowie die Zeiger[21] auf die Vektoren des Aufpunktes, des Richtungsvektors und des Drehwinkels (Berechnung wie in Kapitel 3.4.6 auf Seite 31 beschrieben). Die Variable *from* gibt an, aus welcher Methode *calcEV* aufgerufen wurde. Ein Aufruf aus der Methode *calcOSG* bedeutet

[20] Diese Methode ist optional. Wird kein alternatives Koordinatensystem definiert, findet auch keine Koordinatentransfomation statt.

[21] Als Zeiger oder Pointer bezeichnet man in der Informatik eine Klasse von Variablen, die auf einen Speicherbereich verweisen.

$from = 0$ und damit kein Aufruf der Methode *rotBack*[22]. Variable $from = 1$ bezeichnet einen Aufruf aus *calcUSG* bzw. *rotBack*.

```
calcEV(UX_VECTOR3 vec1, UX_VECTOR3 vec2, UX_VECTOR3 vec1,
       UX_VECTOR3 *Aufpunkt, UX_VECTOR3 *Richtungsvektor,
       double *Dehwinkel, int from)
```

calcOSG berechnet eine Rotationsachse, die nicht transformiert wird. Übergeben werden die x, y und z Koordinaten von drei Punkten (siehe Kapitel 3.4.6 auf Seite 31).

```
calcOSG(double x1, double y1, double z1,
        double x2, double y2, double z2,
        double x3, double y3, double z3)
```

calcUSG berechnet eine zu transformierende Rotationsachse. Übergeben werden ebenfalls die x, y und z Koordinaten von drei Punkten. *calcUSG* ruft die Methode *rotBack* auf.

```
calcUSG(double x1, double y1, double z1,
        double x2, double y2, double z2,
        double x3, double y3, double z3)
```

rotBack rotiert Aufpunkt und Richtungsvektor einer Drehachse mit einem bestimmten Drehwinkel um eine gegeben Rotationsachse. Übergeben werden die Zeiger von Aufpunkt und Richtungsvektor beider Achsen.

```
rotBack(UX_VECTOR3 *ap_osg, UX_VECTOR3 *rv_osg,
        UX_VECTOR3 *ap_usg, UX_VECTOR3 *rv_usg)
```

calcDevUsg berechnet den Winkel α zwischen y-Achse des Koordinatensystems und der auf die xy-Ebene projizierte Rotationsachse (USGA). Im Fall der Sprunggelenkachsenbestimmung ist der Deviationswinkel der Winkel zwischen der auf die Transversalebene projizierten USGA und der sagittotransversal-Achse des *Tibiakoordinatensystems*. Die Deviation wird berechnet mit

$$\alpha = \arctan \frac{x}{y} \tag{42}$$

und dem Richtungsvektor $\vec{rv}_{usga}$ der USGA: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

[22] *rotBack* lässt die berechnete Achse mit einem bestimmten Drehwinkel um ein Rotationsachse rotieren.

Übergeben werden die Zeiger des Richtungsvektors der Rotationsachse und des Deviationswinkels.

```
calcDevUsg(UX_VECTOR3 *rv_usg, double *angle)
```

calcIncUsg berechnet den Winkel β zwischen der y-Achse des Koordinatensystems und der auf die yz-Ebene projizierte Rotationsachse. Im Falle der USGA ist das der eingeschlossene Winkel zwischen der Projektion der USGA auf die Sagitalebene und der Sagittotransversal-Achse. Die Inklination wird berechnet aus

$$\beta = \arctan \frac{z}{y} \tag{43}$$

und dem Richtungsvektor $\vec{rv}_{usga}$ der USGA: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Übergeben werden ebenfalls die Zeiger des Richtungsvektors der Rotationsachse und des Inklinationswinkels.

```
calcIncUsg(UX_VECTOR3 *rv_usg, double *angle)
```

calcDevOsg berechnet den Winkel α zwischen der y-Achse des Koordinatensystems und der auf die xy-Ebene projizierten Rotationsachse. Entspricht dem von auf die Transversalebene projizierten OSGA und der Sagittotransversal-Achse eingeschlossenem Winkel. Die Deviation der OSG-Achse wird berechnet aus

$$\alpha = 90 - \arctan \frac{x}{y} \tag{44}$$

und dem Richtungsvektor $\vec{rv}_{osga}$ der OSGA: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Es werden ebenfalls nur die Zeiger des Richtungsvektors der Rotationsachse und des Deviationswinkels übergeben.

```
calcDevOsg(UX_VECTOR3 *rv_usg, double *angle)
```

calcIncOsg berechnet den Winkel zwischen z-Achse des Koordinatensystems und der auf die xz-Ebene projezierten Rotationsachse. Die Inklination der OSG-Achse wird berechnet aus

$$\beta = 90 - \arctan \frac{z}{x} \tag{45}$$

und dem Richtungsvektor $\vec{rv}_{osga}$ der OSGA: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Analog zum Funktionsaufruf der Inklinations- und Deviationwinkelbestimmung bei der USGA werden wiederum nur die Zeiger des Richtungsvektors der Rotationsachse und des Inklinationswinkels übergeben.

```
calcIncOsg(UX_VECTOR3 *rv_usg, double *angle)
```

3.5.2.2 Eigenschaften der Klasse calcAxis

Verschiedene Eigenschaften der Klasse *calcAxis* können direkt abgefragt werden bzw. müssen gesetzt werden. Nachfolgend werden Variablentyp und -name angegeben.

Die Variable *Leg* kann die Werte *L* für Linkssystem oder *R* für Rechtssystem annehmen. Da bei Gelenkachsen zwischen den Achsen der linken und rechten Extremität unterschieden werden kann, wird die Bewegung und damit auch die Berechnung in einem Links- bzw. Rechtssystem durchgeführt. Standardmäßig ist *Leg = 'R'* definiert.

```
char Leg
```

Alle nachfolgenden Eigenschaften repräsentieren Ergebnisse der Berechnung.

Richtungsvektor der OSGA:

```
UX_VECTOR3 OSG_RV
```

Aufpunkt der OSGA:

```
UX_VECTOR3 OSG_AP
```

Deviationswinkel der OSGA:

```
double OSG_Dev
```

Inklinationswinkel der OSGA:

```
double OSG_Inc
```

Drehwinkel bzw. Bewegungsamplitude der Bewegung um die OSGA:

```
double OSG_Winkel
```

Senkrechter (minimaler) Abstand der OSGA zum Ansatzpunkt des Kraftvektors der Achillessehne:

```
double Lever_OSGA
```

Richtungsvektor der USGA:

UX_VECTOR3 USG_RV

Aufpunkt der USGA:

UX_VECTOR3 USG_RV

Deviationswinkel der USGA:

double USG_Dev

Inklinationswinkel der USGA:

double USG_Inc

Drehwinkel bzw. Bewegungsamplitude der Bewegung um die USGA:

double USG_Winkel

Senkrechter Abstand der USGA zum Kraftvektor der Achillessehne:

double Lever_USGA

Senkrechter Abstand der OSGA zur USGA:

double Dist

Ein Beispiel, wie die Klasse *calcAxis* in eigenen Programmen angewendet werden kann, ist dem Anhang D auf Seite 138 zu entnehmen. Die meisten ab Seite 45 beschriebenen Methoden werden von Funktionen innerhalb der Klasse *calcAxis* aufgerufen (siehe Abbildung 29 auf Seite 51).

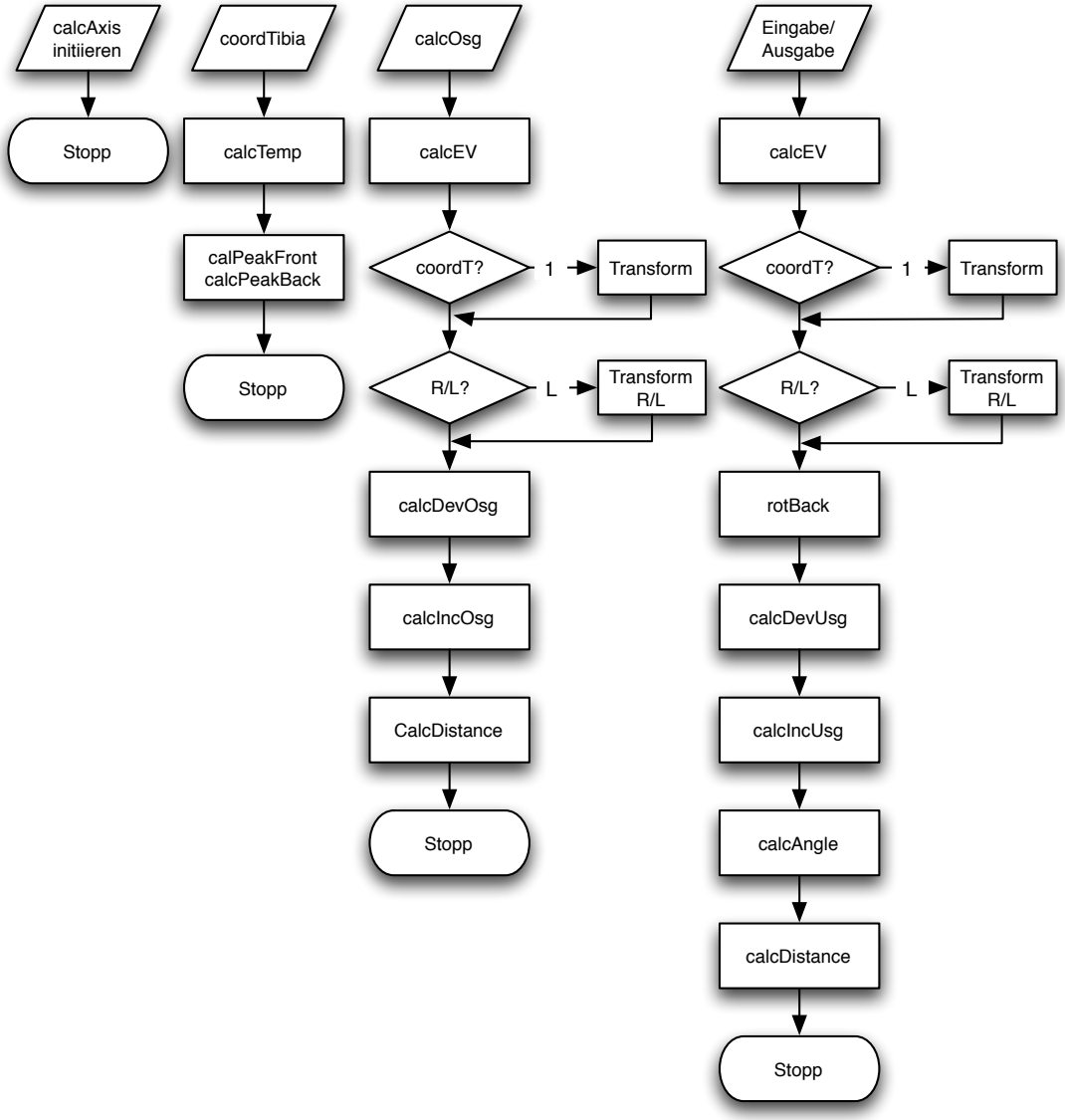

Abbildung 29: Schematischer Aufbau der Klasse *calcAxis*. Nach dem Initiieren durch die Methode *calcAxis*, stehen die Funktionen *coordTibia()*, *calcOsg()* sowie *calcUsg()* zur Verfügung (siehe Beispiel auf Seite 138). Die Abbildung zeigt die Programmstruktur und Beziehung aller Funktionen bzw. Methoden der Klasse *calcAxis*.

3.5.3 Die Klasse USG

Die Klasse *USG* ist gleichzeitig das Kernstück der Software. Alle vorangegangenen beschriebenen Klassen werden von diesem Programmteil zusammengefasst. Hier werden alle Dialogklassen definiert sowie die logischen Abfolgen der für einen Messdurchlauf benötigten Funktionen festgelegt. Die Benutzeroberfläche dieser Klasse dient zum einen der Steuerung sowie Konfiguration der kompletten Software, stellt aber auch die numerischen wie auch grafisch aufbereiteten Ergebnisse einer Messung dar. Selbstverständlich verfügt dieser Programmteil der Software auch über die Möglichkeit, Rohdaten zu speichern bzw. zu öffnen oder die berechneten Ergebnisse als *csv*-Datei[23] zu exportieren. Werden Rohdaten erneut geöffnet, werden alle Ergebnisparameter abermals berechnet. Sollten an den Berechnungsroutinen der Software Veränderungen vorgenommen werden, können alte Datensätze erneut geladen werden. Alle Ergebnisparameter werden erneut, aber mit den veränderten Routinen berechnet. Da das Messverfahren schon in einer frühen Prototypenphase zum Einsatz kam, war diese Funktion besonders wichtig. Alle weiteren Funktionen und die Bedienung der GUI[24] werden im Kapitel 3.5.4 auf Seite 59 behandelt.
Weitere wichtige Funktionen der Klasse *USG* sind:

- ◇ Aufzeichnen der Koordinatendaten für die Berechnung des alternative *Tibiakoordinatensystems*.
- ◇ Erfassen der von der Klasse *zebClass* gelieferten Rohdaten.
- ◇ Filtern der Rohdaten um den Fehler durch Rauschen- bzw. Störgeräusche zu minimieren.
- ◇ Ermitteln geeigneter Datensätze zur Berechnung der gewünschten Parameter der Rotationsachsen.

3.5.3.1 Routine zur Erfassung der Koordinatendaten

Wie in Kapitel 3.4.3 auf Seite 25 bereits erwähnt werden für die Berechnung eines *Tibiakoordinatensystems* vier anatomische Punkte benötigt. Da darüber hinaus auch noch jeweils ein Punkt auf dem Malleolus lateralis und Malleolus medialis aufgezeichnet werden soll, müssen insgesamt sechs Punkte von dieser Routine aufgezeichnet und berechnet werden. Die Berechnung der Punkte aus den X, Y und Z Koordinaten der Ultraschallsender der Kanäle 1 und 2 übernimmt die in Kapitel 3.5.2 ab Seite 45 beschriebene Klasse *calcAxis*. Vom *GUI* müssen für die Erfassung der Koordinatendaten verschiedene Prozesse bzw. Dialoge gestartet werden:

[23] Von Excel lesbares Dateiformat.
[24] Abkürzung für 'General User Interface', gleichbedeutend mit Benutzeroberfläche.

- ◇ Ein *Arbeitsthread*[25] der ununterbrochen mit Hilfe von Methoden der Klasse *zebClass* den Buffer der Hardware, also die Koordinatendaten, ausliest. Dieser Arbeitsthread wird dabei mit weiteren Threads bzw. Prozessen synchronisiert und kommuniziert über bestimmte Schnittstellen, in diesem Fall mit verschiedenen Benutzeroberflächenthreads[26] [Microsoft 2005].

- ◇ Insgesamt werden vom Hauptprogramm oder *Parentdialog*[27] nacheinander sechs *Childdialoge* erzeugt. Jeder dieser erzeugten Dialoge dient der Erfassung eines anatomischen Punktes zur Berechnung des *Tibiakoordinatensystems*.

- ◇ Gleichzeitig wird in diesen Childdialogen ein Benutzeroberflächenthread gestartet, der mit dem Arbeitsthread des Parentdialogs kommuniziert. Nur wenn ein Ultraschallsignal von den Sensoren empfangen wird, werden die entsprechenden Schaltflächen des *GUI* der Childdialoge freigegeben und es kann eine Aufzeichnung der Koordinatendaten stattfinden. Die aufgenommenen Datensätze werden in einem Array[28] des Parentdialogs gespeichert. Anschließend wird der Childdialog geschlossen und der nächste vom Parentdialog erzeugt.

Die exakte Vorgehensweise und Programmierung kann dem Anhang F ab Seite 144 entnommen werden. Abbildung 30 auf Seite 54 bildet die prozeduralen Zusammenhänge des Vorgangs zum Festlegen des *Tibiakoordinatensystems* ab. Wurden alle sechs anatomischen Punkte erfolgreich aufgezeichnet, wird das *Tibiakoordinatensystem* sowie auch der Korrekturfaktor für die Ausbreitungsgeschwindigkeit des Ultraschalls berechnet.

3.5.3.2 Routine zur Aufzeichnung der Rohdaten

Im Gegensatz zur Berechnung des *Tibiakoordinatensystem*s, bei der nur Momentaufnahmen der Ultraschallmarker gemacht werden müssen, wird zur Bestimmung der Parameter der USGA und OSGA ein stetiges Signal des Ultraschallmarkers gebraucht. Alle vom CMS 20 gelieferten Koordinatendaten werden von einem Arbeitsthread in einem Array erfasst. Abbildung 31 auf Seite 55 zeigt die Rohdaten der

[25] Unter einem Thread versteht man in der Software-Architektur einen Ausführungsstrang oder eine Ausführungsreihenfolge innerhalb eines Prozesses, der nebenläufig zu anderen Threads ausgeführt werden kann. In diesem Beispiel läuft der Arbeitsthread zum Auslesen der Hardware parallel und autonom ab, ohne dabei andere Prozesse bzw. Threads zu blockieren.

[26] Ein Benutzeroberflächenthread wird normalerweise verwendet, um unabhängig von Threads, die derzeit andere Teile der Anwendung ausführen, Benutzereingaben zu behandeln und um auf vom Benutzer erzeugte Ereignisse zu reagieren.

[27] Eine Anwendung kann verschiedenste Dialoge oder Fenster gleichzeitig verwenden. Child- oder Kinderdialoge werden dabei von Parent- bzw. Elterndialoge erzeugt.

[28] Mit Hilfe eines Arrays können Daten eines einheitlichen Datentyps geordnet so im Speicher eines Computers abgelegt werden, dass ein Zugriff auf die Daten über einen Index möglich wird.

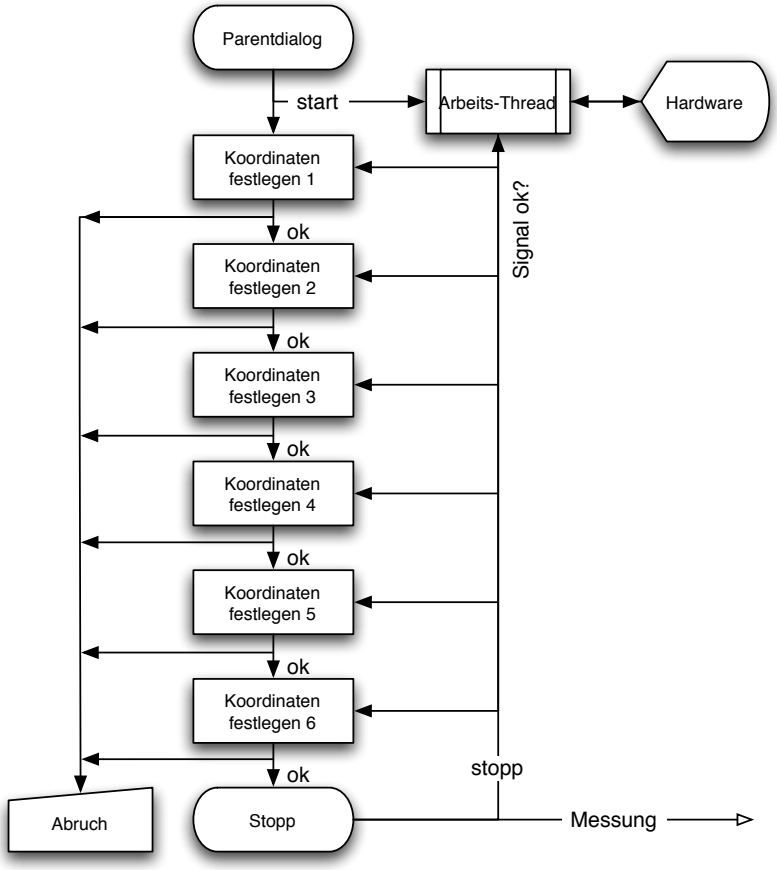

Abbildung 30: Ablauf der Routine zur Erfassung der anatomischen Bezugspunkte. Gleichzeitig mit dem Erzeugen des ersten Childdialogs wird der Arbeitsthread zum Auslesen des Hardwarebuffers, gestartet. Die Benutzeroberflächenthreads der Childdialoge kommunizieren mit dem Arbeitsthread des Parentdialog. Nur wenn ein Ultraschallsignal von beiden Markern P_1 und P_2 empfangen wird, werden die entsprechenden Schaltflächen des Childdialogs zum Aufzeichnen der Koordinatendaten freigegeben. Wurden die Datensätze korrekt aufgezeichnet, wird der Childdialog geschlossen und der nächste erzeugt, anderenfalls kommt es zum Abbruch. Wurden alle sechs Datensätze erfolgreich aufgezeichnet, wird im Parentdialog die Schaltfläche zur Messung freigegeben.

X-Koordinate eines Messdurchlaufs. Es handelt sich dabei um ein rechtes Sprunggelenk. Die Eversions- bzw. Inversionsbewegung wurde fünf mal wiederholt. Da sich die Software jeweils zwei Extrempunkte und einen zugehörigen Mittelpunkt zur Berechnung der Rotationsachsen sucht, muss vermieden werden, dass fälschlicherweise ein Messfehler bzw. Ausreißer als Extrempunkt erkannt wird. Da solche Ausreißer immer eine hochfrequente Veränderung im Signalverlauf verursachen, kann dieser Fehler durch den Einsatz eines Lowpassfilters vermieden werden.

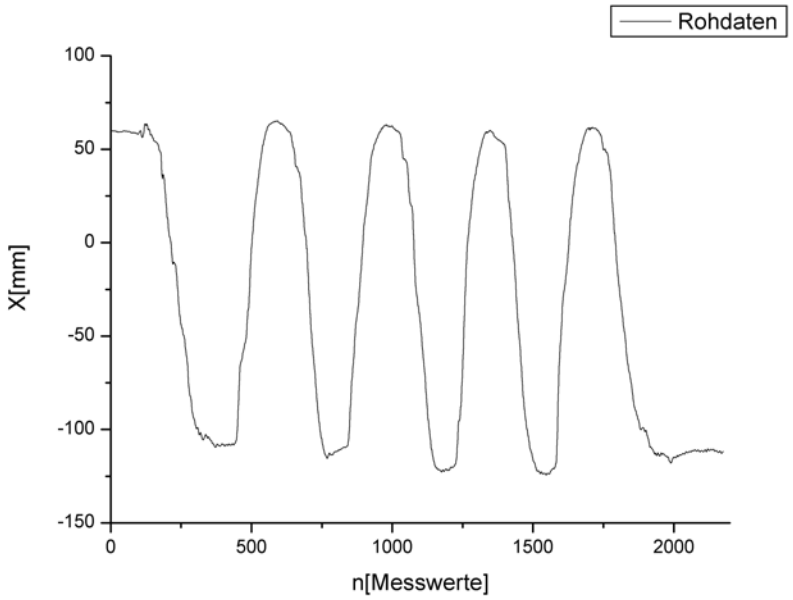

Abbildung 31: Rohdaten der X Koordinaten eines Messdurchlaufs (rechtes Sprunggelenk). Die erste abfallende Flanke der Signalkurve kennzeichnet die Inversionsbewegung. Die Eversions- bzw. Inversionbewegung wurde fünfmal wiederholt.

3.5.3.3 Filterroutine

Da es sich bei der auszuführenden Bewegung um die Sprunggelenkachsen um eine sehr niederfrequente Bewegung handelt, wurde ein *Butterworth Lowpassfilter* vierter Ordnung[29] mit einer Grenzfrequenz von $f_g = 1Hz$ eingesetzt. Die Methode *callFilter* der Klasse *USG* liest alle Rohdaten aus, filtert diese und schreibt die gefilterten Datensätze in ein neues Array. Die Rohdaten bleiben erhalten, d. h. sollten die Filterkoeffizienten verändert werden, kann die Berechnung jederzeit erneut auf das Rohdatenmaterial angewandt werden. Abbildung 32 auf Seite 56 zeigt den Signalverlauf des gefilterten und ungefilterten Datenmaterials[30]. Die gefilterte Signalkurve zeigt dabei eine leichte Phasenverschiebung. Da es sich bei der Bewegung um keinen zeitkritischen Vorgang handelt, kann dieser Sachverhalt außer Acht bleiben.

[29] Durch das Hintereinanderschalten von mehreren Tiefpässen kann man dessen Ordnung erhöhen. Zwei hintereinander geschaltete Tiefpässe 2. Ordnung bilden einen Tiefpass 4. Ordnung. Die Dämpfung nimmt also oberhalb der Grenzfrequenz mit $4 \cdot 20dB/Dekade = 80dB/Dekade$ zu.

[30] Die als Textfile gespeicherten Rohdaten einer Messung enthalten zusätzlich die gefilterten Daten. Diese gefilterten Daten werden vom Programm später zwar nicht mehr verwendet, waren aber während der Entwicklungsphase für die Einstellung der Filterkoeffizienten sehr hilfreich.

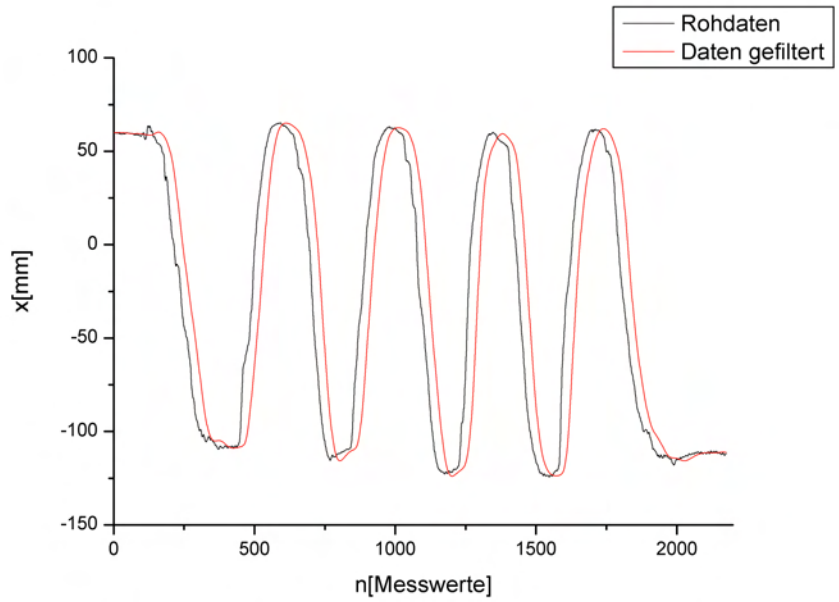

Abbildung 32: Rohdaten und gefilterte Daten der X Koordinate eines Messdurchlaufs. Die rote Signalkurve zeigt den gefilterten Datensatz. Zu beobachten ist eine leichte Phasenverschiebung $\Delta\varphi$ gegenüber dem nicht gefilterten Signalverlauf. Da es sich um einen nicht zeitkritischen Vorgang handelt, ist diese Phasenverschiebung nicht weiter von Bedeutung. Ausreißer oder Störsignale wie sie von der schwarzen Kurve z.B. um den Messwert 200 gezeigt werden, werden weitgehend herausgefiltert.

3.5.3.4 Routine zur Ermittlung geeigneter Datensätze für die Parameterberechnung

Aus den Daten der Bewegung um die OSGA oder USGA werden anschließend die gesuchten Parameter der Rotationsachsen berechnet. Dazu werden von einer Funktion des Hauptprogramms jeweils Hoch- und Tiefpunkte des Signalverlaufs der aufgezeichneten x-Bewegungskoordinaten bestimmt. Sind die Extrempunkte ermittelt, wird anschließend der Mittelpunkt zu diesen Punkten gesucht. Aus diesen drei Punkten kann die Software die gewünschten Parameter mit Hilfe der Klasse *calcAxis* berechnen. Tabelle 3 auf Seite 57 enthält Indexnummern des Arrays, der die X, Y und Z Koordinaten der Messung speichert. Die Indexnummern für die Punkte P_1 und P_3 repräsentieren dabei die Extrempunkte des Signalverlaufs, Punkt $P2$ den dazugehörigen Mittelpunkt. Während der Entwicklungsphase wurden zusätzliche Funktionen zur Ausgabe dieser Datentriplets in den Programmcode eingefügt. Mit Hilfe der Rohdaten der Messung und den ausgegebenen Indexnummer des Array konnte die Routine zum Auffinden von Extremstellen und Mittelpunkten überprüft werden.

Abbildung 33 auf Seite 58 zeigt das gefilterte Datenmaterial und die von der Routine gefundenen Datentriplets.

Tabelle 3: Datentriplets zur Berechnung der Rotationsachsen. Der Array enthält an den Stellen P_1 und P_2 die Extremstellen der Koordinatendaten des Signalverlaufs, an der Stelle P_2 den dazugehörigen Mittelpunkt. In Abbildung 33 auf Seite 58 sind diese Arraystellen als schwarze Punkte im Schaubild markiert.

Array Indexnummer		
P_1	P_2	P_3
1	262	424
425	518	612
613	736	803
804	916	1012
1013	1125	1201
1202	1292	1380
1381	1473	1571
1572	1641	1739
1740	1847	2027

Wie bereits im Kapitel 3.4 auf Seite 22 beschrieben, handelt es sich bei der Entwicklung des Messsystems zu einem betriebsfähigen Prototypen um einen iterativen Prozess. Um den Messfehler möglichst zu minimieren, wurden deshalb die Berechnungsroutinen der Software ständig verbessert. Dabei wurden sowohl die Routinen zur Bestimmung geeigneter Datensätze zur Berechnung der Rotationsachsenparameter als auch die Berechnungsroutinen verändert. Die Verbesserung durchlief verschiedene Phasen:

1. In der ersten Version wurden instantane Rotationsachsen berechnet. Das Programm lieferte sehr gute Ergebnisse an ersten Messungen am mechanischen Modell. Bei ersten Versuchen in vivo waren die Ergebnisse allerdings mit einem sehr großen Fehler ($\Delta E > 25°$) behaftet. Bedingt durch die geringen Punktabstände der zur Berechnung benutzten Koordinatendaten ($s < 5mm$ bei instantaner Rotationsachsen) hatten schon kleine Messfehler einen großen Einfluss auf das berechnete Ergebnis.

2. Ein Lowpassfilter eliminierte mögliche Störsignale, der Messfehler wurde dadurch weiter minimiert.

3. Bei der Berechnung von finiten Achsen haben kleine Fehler der X, Y und Z Koordinaten der Ultraschallsender, bedingt durch die günstigeren geometrischen Verhältnisse, weniger Einfluss auf das zu berechnende Ergebnis. Es

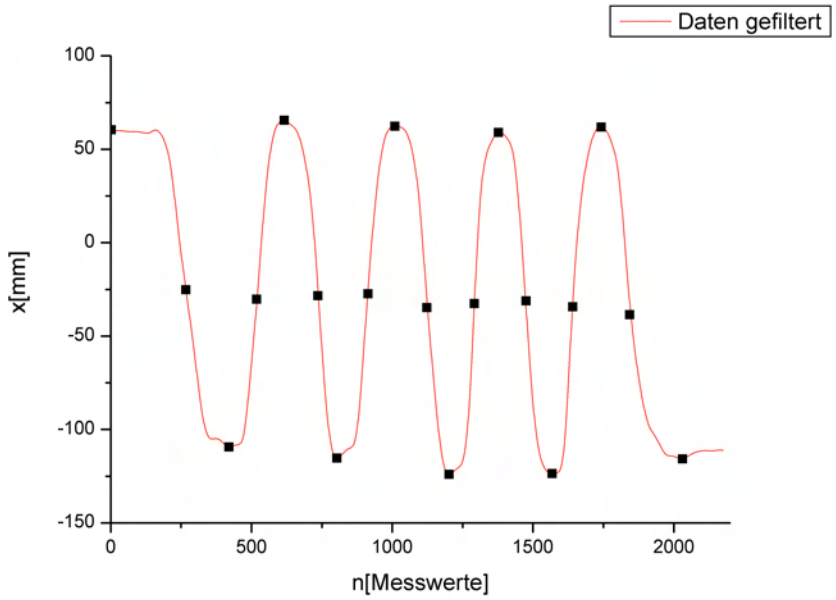

Abbildung 33: Gefiltertes Signal der X-Koordinate mit den von der Routine gefundenen Extremstellen und Mittelpunkten. Die schwarzen Punkte kennzeichnen die Punkte der maximalen Eversion (Hochpunkte), maximalen Inversion (Tiefpunkte) und einem Mittelpunkt zu den beiden Extremstellen. Aus diesem Datensatz werden neun Rotationsachsen berechnet.

werden nur noch drei Koordinatensätze benötigt. Benutzt werden die Koordinatendaten der Anfangs-, End- und einer Mittelstellung (siehe Abbildung 33 auf Seite 58).

4. Um den Messfehler weiter zu minimieren, wurden die Methoden des Programms erneut verändert. Es werden die Rotationsachsen von mehreren Bewegungszyklen. also mehreren aufeinander folgenden Inversions- und Eversionsbewegungen berechnet. Aus diesen Ergebnissen werden dann Mittelwert und Standardabweichung berechnet.

5. Ergebnisse, die außerhalb der zweifachen Standardabweichung liegen, werden als Ausreißer behandelt. Mittelwert und Standardabweichung werden erneut berechnet, ohne die Ausreißer einzubeziehen.

Durch diese Modifikationen konnte der Fehler, der zwangsläufig bei in vivo Messungen z.B. durch ungenaue Bewegungsausführung, Hautverschiebungen usw. auftritt, minimiert werden.

3.5.4 Programmoberflächen - GUI

Von den Benutzeroberflächen (GUI) lassen sich alle Prozesse der Messprogramms steuern. Bei der Entwicklung dieser Oberflächen wurde deshalb ein besonderes Augenmerk auf die intuitive Bedienbarkeit gelegt. Desweiteren sollten Fehlbedienungen des Programms und damit Fehlermessungen ausgeschlossen werden, der gesamte Messprozess sollte möglichst objektivbar gestaltet sein. Das Programm setzt sich aus verschiedenen Dialogen zur Programmsteuerung, Datenerfassung und Datenausgabe zusammen:

- ◇ Dialog der Klasse USG: Alle Prozesse und weiteren Dialoge werden von dieser Oberfläche aus gestartet. Dieser Dialog stellt sozusagen das „Hauptprogramm“ dar.
- ◇ Dialog - Probandendaten: Dient zur Erfassung der Stammdaten von Probanden.
- ◇ Dialog - Koordinaten festlegen: Besteht genau genommen aus sechs Dialogen, die zur Erfassung der markanten anatomischen Punkte zur Berechnung des *Tibiakoordinatensystems* dienen.
- ◇ Dialog - Messung: Setzt sich aus einem Dialog zur Messung der OSGA und einem weiteren Dialog zur Messung der USGA zusammen.

Nachfolgend soll lediglich das GUI der Klasse USG, sozusagen die Hauptbenutzeroberfläche und damit Steuereinheit des Messprogramms, besprochen werden. Alle weiteren Dialoge werden im Anhang G ab Seite 187 beschrieben.

3.5.4.1 Dialog der Klasse USG

Alle Dialoge werden von dieser Benutzeroberfläche aus geöffnet. Sämtliche wichtigen Funktionen lassen sich von auf der Dialogoberfäche angebrachten Schaltflächen direkt ansteuern. Über eine Windows-typische Menüleiste lassen sich alle weiteren Funktionen anprechen. Die wichtigsten Ergebnisse werden sowohl numerisch als auch grafisch dargestellt. Abbildung 34 auf Seite 61 zeigt alle Schalt- und Ausgabeflächen der GUI.

Funktionen der Programmsteuerung:

1 Probanden Daten: Öffnet den Dialog Probanden Daten.

2 Koordinaten festlegen: Ruft den Dialog zum Festlegen eines *Tibiakoordinatensystem*s auf.

3 Messung starten: Diese Schaltfläche wird erst aktiviert nachdem ein *Tibiakoordinatensystem* festgelegt wurde.

4 Daten speichern: Öffnet einen Windowsdialog zum Speichern der Rohdaten.

5 Programm zurücksetzen: Führt einen Programmreset durch. Alle Messungen und Ergebnisse werden gelöscht, alle Schalt- und Ausgabeflächen zurückgesetzt.

6 Programm beenden.

Felder der numerischen Ergebnisdarstellung:

7 Inklinationswinkel der OSGA.

8 Drehwinkel der OSGA.

9 Deviationswinkel der OSGA.

10 Hebel, der senkrechte minimale Abstand der OSGA zur Z Achse. Entspricht dem Angriffspunkt des Kraftvektors der Achillessehne.

11 Inklinationswinkel der USGA.

12 Drehwinkel der USGA.

13 Deviationswinkel der USGA.

14 Hebel, der senkrechte minimale Abstand der USGA zur Z Achse.

Die Felder 11 bis 14 verändern zudem noch je Größe der Standardabweichung ihre Hintergrundfarbe. Dunkelgrün bedeutet eine Standardabweichung $S < 5°$, hellgrün $5° < S < 10°$, hellblau $10° < S < 15°$ und rot $S > 15°$. Somit können die Messergebnisse direkt nach einer Messung beurteilt und die Messung gegebenenfalls wiederholt werden.

Schaltflächen zur Anpassung der grafischen Darstellung:

15 Schaltet die 3D-Darstellung des Fußes zwischen den Modi Fläche, Fläche 2D XY, Fläche 2D YZ und Gittenetz um.

16 Wechselt zwischen einer orthografischen und perspektivischen Darstellung des Diagramms.

17 Kopiert die aktuelle 3D Darstellung des Fußes in die Zwischenablage.

18 Diese drei Schaltflächen rotieren das Diagram in die XZ, XY oder YZ Ebenen.

19 Schaltet zwischen den Funktionen Rotieren, Zoomen, und Pannen um.

20 Öffnet den Dialog Diagrammeigenschaften.

Feld für die grafische Darstellung der Ergebnisse:

21 Die Achsen werden in einem dreidimensionalen kartesischen Koordinatensystem dargestellt. Die USGA in der Farbe blau, die OSGA wird in rot eingezeichnet. Durch einen Mausklick in das Diagramm kann durch entsprechende Bewegungen der Maus das Schaubild rotiert, gezoomt oder gepannt werden.

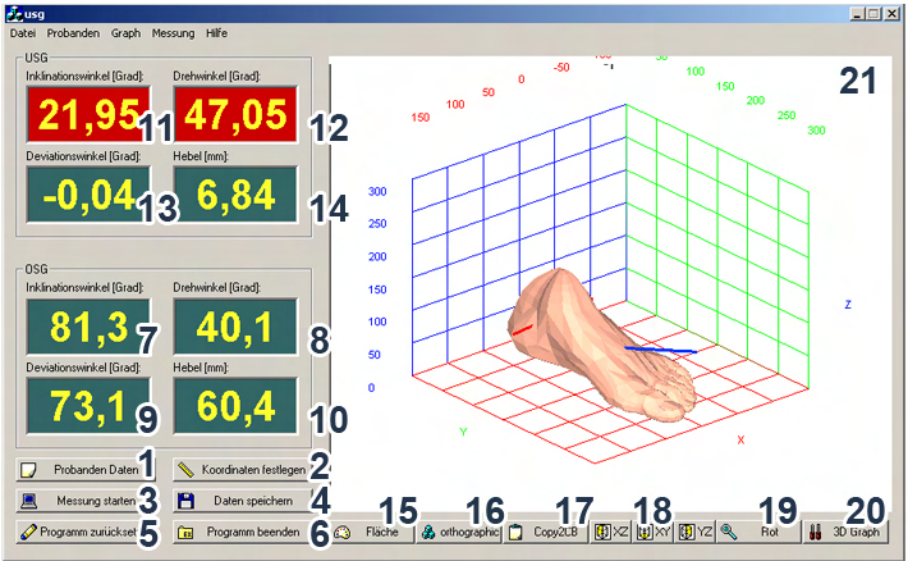

Abbildung 34: Benutzeroberfläche des Hauptprogramms. Über die Schaltflächen 1-6 können alle Hauptfunktionen des Programms aufgerufen werden. Die Felder 7-14 geben die wichtigsten Ergebnisse numerisch aus. Über die Schaltflächen 15-20 lassen sich die Parameter der grafischen Darstellung anpassen. Feld 21 stellt die Sprungelenkachsparameter grafisch dar.

3.6 Hardwaremodifikation und -entwicklung

Das 3D Messystem CMS 20 der Firma Zebris® ist grundsätzlich für eine Bewegungsanalyse des Sprunggelenkkomplexes geeignet (siehe Kapitel 3.3 auf Seite 19). Um es zur Bestimmung der Sprunggelenkachsen einsetzen zu können, müssen an der Hardware allerdings verschiedene Modifikationen vorgenommen werden. Da für die Bewegungsanalyse lediglich eine Bewegung des Calcaneus relativ zum Unterschenkel von Bedeutung ist, müssen sowohl für den Unterschenkel als auch für den Calcaneus verschiedene Vorrichtungen gebaut werden die eine Befestigung von Ultraschallmarkern- und Sensoren erlauben.

3.6.1 Befestigung der Ultraschallsensoren

Von der Firma Zebris® wurde ein spezieller Sensor- und Markersatz für das *JMA* Mess-System zur Kiefergelenkanalyse entwickelt. Zwei *JMA* Sensoren sind auf dem Markt erhältlich: Der *JMA* $11 \cdot 11$ und der etwas kleinere *JMA* $8 \cdot 8$ Sensor. Diese Sensoren wurden für Bewegungsanalysen im Nahbereich, $d < 50cm$, konzipiert und besitzen eine höhere Genauigkeit als die Standardsensoren des CMS 20. Da *JMA* Sensoren für die Analyse des Kiefergelenks am Unterkiefer angebracht werden müssen, zeichnen sie sich durch eine relativ geringe Masse aus und sind somit geeignet, am Unterschenkel befestigt zu werden. In dieser Arbeit wurde der*JMA* $11 \cdot 11$ Sensor verwendet. Dieser Sensor besteht aus vier Ultraschall-Miniatur-Mikrofonen, die eine quadratische Fläche mit einer Kantenlänge von $11cm$ aufspannen. Der Koordinatenursprung des *JMA* $11 \cdot 11$ liegt im Schnittpunkt der beiden Diagonalen des von den Ultraschallsensoren aufgespannten Quadrats (siehe Abbildung 35 auf Seite 63).

Zur Befestigung der Sensoren am Unterschenkel erweist sich die Tibia als geeignet. **Alt** hatte für die Anbringung der Infrarot Marker ein ähnliches System verwendet [Alt 2001]. Diese Haltevorrichtung wurde modifiziert, sodass der*JMA* Sensor unkompliziert an der Tibia angebracht werden kann. Die gesamte Vorrichtung ist so konstruiert, dass es möglich ist die Sensoren, unabhängig von der interindividuellen Beingeometrie, optimal auf die Ultraschallsender auszurichten. Abbildung 36 auf Seite 64 zeigt einen Prototypen der Haltevorrichtung des *JMA* Sensors. Abbildung 37 auf Seite 66 zeigt den an der Tibia applizierten *JMA* $11 \cdot 11$Sensor.

3.6.2 Befestigung der Ultraschallmarker

Damit die Sensoren ein störungsfreies Ultraschallsignal empfangen können, müssen die Ultraschallsender möglichst unterhalb der Sensoren angebracht werden. Für die Berechnung der OSG- bzw. USG-Achse muss eine Bewegung oder Rotation im

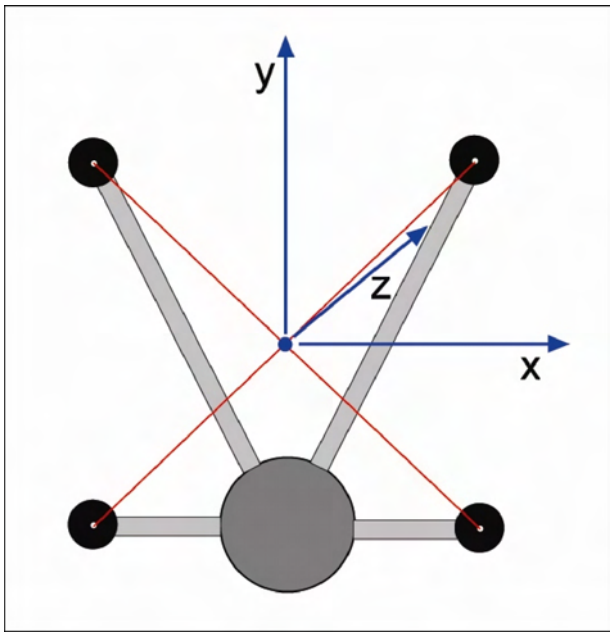

Abbildung 35: Skizze *JMA* $11 \cdot 11$ Sensor. Der Koordinatenursprung liegt im Schnittpunkt der Diagonalen der Eckpunkte des Sensors. Alle vom Sensor erfassten X-, Y- und Z-Koordinatendaten werden in diesem Koordinatensystem angegeben.

talokruralen- bzw. im subtalaren Gelenk aufgezeichnet werden. Dies ist nur durch eine indirekte Befestigung der Ultraschallmarker am Calcaneus möglich. Nur dadurch können Bewegungen des Vorfusses ausgeschlossen werden. Abbildung 38 auf Seite 66 zeigt, wie die Markerbefestigung mit Hilfe eines Fussballschuhes realisiert wurde. Der vordere Teil des Fussballschuhes wurde entfernt, anstelle der Stollen wurde ein Aluminiumbügel angeschraubt. Auf diesem Aluminiumbügel können die Ultraschallmarker befestigt werden. Der Schuh kann mit Hilfe eines Klettbandes am Fuß befestigt werden (siehe Abbildung 39 auf Seite 67). Für die Berechnung der Parameter der OSGA und USGA wurden dabei zwei verschiedene Marker verwendet. Für die Bestimmung der OSGA-Parameter wurden die Koordinatendaten des Ultraschallmarkers verwendet, der näherungsweise dicht an der USGA befestigt wurde. Es muss dabei berücksichtigt werden, dass die aufgezeichneten Daten der Dorsalflexion im OSG von einer Bewegung im USG überlagert sein können[31]. Aufgrund günstiger geometrischer Verhältnisse, sprich dem kleinen Abstand des zur Berechnung verwendeten Markers zur Rotationsachse, kann dieser Fehler minimiert werden. Für die Bestimmung der USGA soll der Radius bzw. Abstand des Ultraschallmarkers zur vermuteten Rotationsachse möglichst groß gewählt werden. Sollte nun wiederum das Signal der Inversions bzw. Eversionsbewegung von einer Bewegung im talokruralen

[31] Das subtalare Gelenk kann aufgrund der anatomischen Situation des Talus nicht festgesetzt werden.

Abbildung 36: Prototyp der Ultraschallsensorbefestigung. Die Sensorbefestigung verfügt über zwei Gelenke. Der Sensor kann so optimal unabhängig von interindividuellen Unterschieden an der Tibia des Probanden angebracht werden. Das Gesamtgewicht beträgt $m = 50g$. Die Haltevorrichtung besteht aus einem leicht flexiblen Kunststoffband, das mit Hilfe von doppelseitigem Klebeband auf die Tibia geklebt werden kann. Am Kunstoffband wiederum ist ein Metallstift befestigt, an dem über ein Gelenk die magnetische Halterung für den *JMA* Sensor angebracht ist. Der Sensor kann so unkompliziert auf der Halterung befestigt werden. Die gesamte Vorrichtung ist so konstruiert, dass es möglich ist, die Sensoren unabhängig von der interindividuellen Beingeometrie optimal auf die Ultraschallsender auszurichten.

Gelenk überlagt sein, kann so trotzdem ein hinreichend genaues Ergebnis erzielt werden.

3.6.3 Entwicklung des Taststiftes zur Erfassung anatomischer Punkte

Um markante anatomische Punkte aufzeichnen und daraus das *Tibiakoordinatensystem* berechnen zu können, wurde ein spezieller Taststift entwickelt. Aus der bekannten Geometrie des Taststiftes (siehe Anhang H auf Seite 191) kann über einfache vektorielle Beziehungen der Zeiger auf den jeweilig markierten Punkt berechnet werden. Für die Taststiftspitze gilt somit:

$$P = \vec{m}_1 + l_1 \cdot \frac{\vec{m}_1 - \vec{m}_2}{|\vec{m}_1 - \vec{m}_2|} \tag{46}$$

mit:

P: Zu markierenden anatomischen Punkt.
$\vec{m}_1$: Vektor zum Ultraschallmarker M_1.
$\vec{m}_2$: Vektor zum Ultraschallmarker M_2.
l_1: $35mm$, Abstand vom Marker M_1 zur Taststiftspitze

Für die Spitze an der Taststiftrückseite gilt:

$$P = \vec{m}_1 - l_2 \cdot \frac{\vec{m}_1 - \vec{m}_2}{|\vec{m}_1 - \vec{m}_2|} \tag{47}$$

mit:

P: Zu markierenden anatomischen Punkt.
$\vec{m}_1$: Vektor zum Ultraschallmarker M_1.
$\vec{m}_2$: Vektor zum Ultraschallmarker M_2.
l_2: $210mm$, Abstand vom Marker M_1 zur Spitze an der Taststiftrückseite

Damit die Ultraschallsensoren ein störungsfreies Signal von den Markern M_1 und M_2 empfangen zu können, muss sich der Taststift unterhalb der an der Tibia angebrachten Sensoren befinden. Die Spitze an der Taststiftrückseite wurde für Punkte an der Unterschenkelrückseite konstruiert. Die Ultraschallsender M_1 und M_2 werden vom CMS 20 auf Kanal 1 und 2 eingezogen. Der Prototyp des Taststiftes verfügt zusätzlich über zwei Taster, die den digitalen Kanal des CMS 20 ansteuern. Über diese digitalen Kanäle kann das Programm gesteuert werden. Durch Betätigen des Taster 1 wird ein Ultraschallsignal aufgezeichnet, Taster 2 schaltet die Routine zur Bestimmung des *Tibiakoordinatensystem*s einen Schritt weiter. Befindet sich das Programm in der Routine *Bestimmung der Sprunggelenkachsen*, dient der Taster 1 zum Starten und Stoppen der Aufzeichnung, Taster 2 schaltet die Routine wiederum weiter. Ein exakter Konstruktionsplan des Taststiftes befindet sich im Anhang H auf Seite 191.

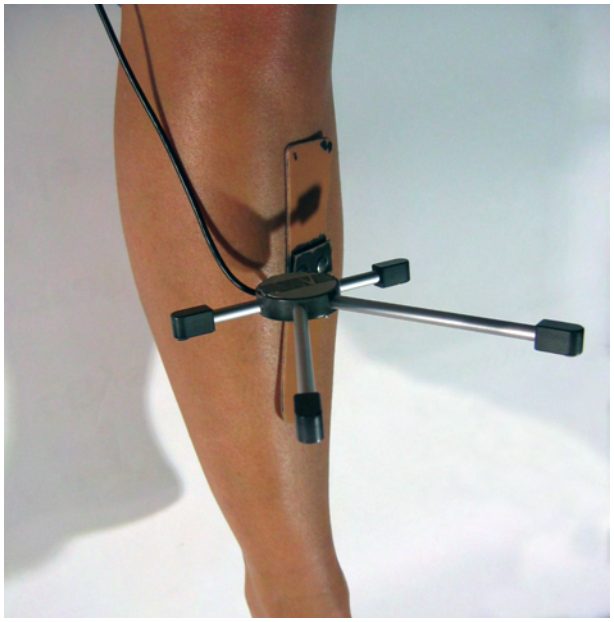

Abbildung 37: An der Tibia applizierter *JMA* Sensor. Durch diese Befestigung werden lediglich Bewegungen im OSG und USG relativ zur Tibia aufgezeichnet.

Abbildung 38: Prototyp der Ultraschallmarkerbefestigung. Das Klettband dient zur Befestigung des Schuhs am Probandenfuß. Der in der Mitte des Aluminiumbügels angebrachte Marker dient zur Bestimmung der OSGA Parameter und wird vom CMS 20 auf Kanal 3 eingezogen. Der rechte Ultraschallmarker liefert die zur Berechnung der USGA Parameter benötigten Koordinatendaten und wird vom CMS 20 auf Kanal 4 eingezogen.

Abbildung 39: Prototyp der Ultraschallmarkerbefestigung, angebracht am Probandenfuß. Zu sehen ist, dass der Vorfuß nicht vom Schuh mit eingeschlossen wird. Das Klettband drückt die hintere Schuhschale gegen das Fersenbein. Bei einer Bewegung folgen die Marker der Bewegung des Calcaneus.

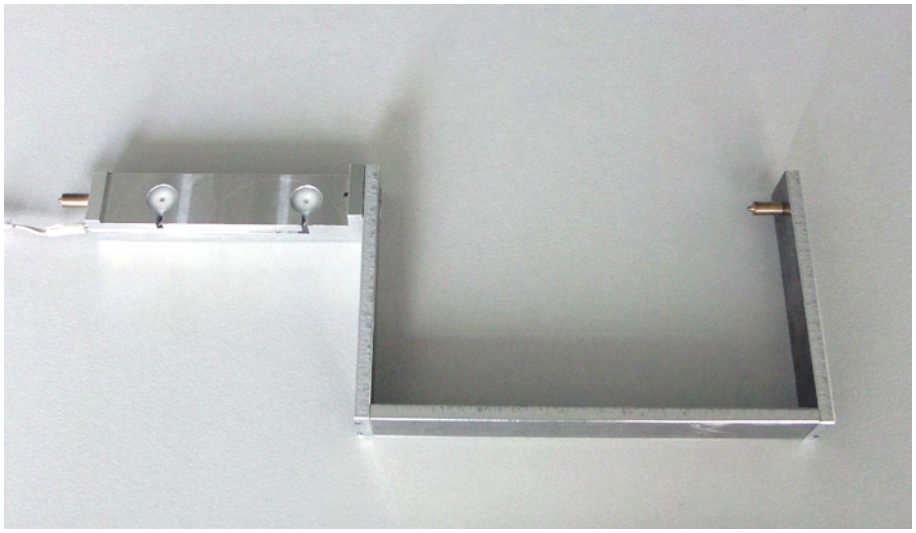

Abbildung 40: Prototyp Tatstift zur Erfassung markanter anatomischer Punkte. Die Ultraschallmarker wurden so weit im Griff versenkt, dass sie genau in der Verbindungsachse der beiden Taststiftspitzen liegen. Ein Konstruktionsplan des Taststiftes befindet sich im Anhang H auf Seite 191.

3.7 Validierung der Messmethode

Der modulare Aufbau des entwickelten diagnostischen Verfahrens erlaubt eine Unterteilung in die Komponenten der Software, Hardwareschnittstelle, Hardware sowie in vivo Versuchsdurchführung. Die Validierung des Gesamtmesssystems erfolgte durch eine Validierung der einzelnen Module. Ergebnisse dieser Validierungen wurden in den darauffolgenden Konzeptions- und Entwicklungsphasen berücksichtigt. Die Kapitel 3.7.1 und 3.7.2 beschriebenen Validierungsversuche am virtuellen und mechanischen Modell dienten der Überprüfung der Soft- und Hardware des diagnostischen Verfahrens. Da die Bestimmung der USGA in vivo auf einer isolierten Bewegungsanalyse des USG basiert, musste in einem weiteren Versuch überprüft werden, ob das OSG und damit der Talus in einer maximalen Dorsalflexion in der Tibia-Fibula-Knochengabel als festgesetzt betrachtet werden kann. Dieser Versuch ist in Kapitel 3.7.3 erläutert.

3.7.1 Validierung am virtuellen Modell

Beim virtuellen Modell handelt es sich um ein Programm, das Rotationen um zwei miteinander gekoppelte Rotationsachsen simuliert und die Koordinatendaten von virtuellen Markern ausgibt. Dabei wurde, analog zum Versuch am Sprunggelenk, zuerst eine Rotation um eine der Achsen durchgeführt. Da die Achsen miteinander gekoppelt sind, wird durch diese Rotation die räumliche Position und Orientierung der zweiten Rotationsachse verändert. Anschließend wurde eine weitere Rotation um die zweite Rotationsachse ausgeführt. Mithilfe der Koordinatendaten der virtuellen Marker können die beiden Rotationsachsen berechnet werden. Aufpunkt und Richtungsvektoren der beiden Rotationsachsen waren dabei bekannt, die von der Klasse *calcAxis* berechneten Ergebnisse der Rotationsachsen konnten daher mit den gegebenen Rotationsachsen verglichen werden. Dieser Versuch diente zum einen der Überprüfung der Klasse *calcAxis*, die alle notwendigen Berechnungen übernimmt, zum anderen dem Festlegen von Mindestabständen der Marker zur Rotationsachse bzw. Mindestdrehwinkel zwischen den drei Aufnahmen des rotierenden Markers[32]. Im Versuch wurden sowohl Drehwinkel als auch der Abstand der Marker zur Rotationsachse variiert.

[32] Es ist davon auszugehen, dass bei Unterschreitung dieser Mindestmaße, bedingt durch Rundungsfehler, ein mit einem sehr großen Fehler behaftetes Ergebnis berechnet wird.

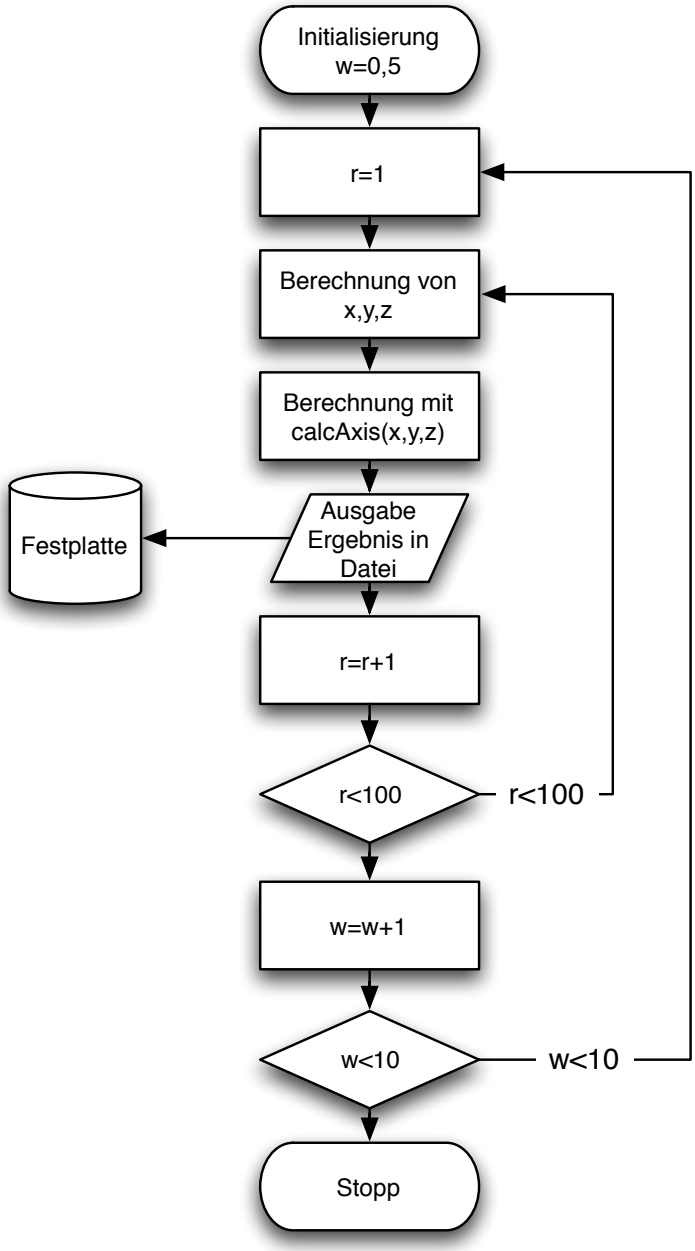

Abbildung 41: Flussdiagramm zum Programm Modell virtuell. In einer ersten Programmschleife wird der Winkel α zwischen $0,5° < \alpha < 10°$ in $\Delta\alpha = 0,5°$ Schritten vergrößert. Innerhalb dieser ersten Schleife wird eine zweite Schleife durchlaufen die ebenfalls schrittweise den Abstand r zur Rotationsachse zwischen $1mm < r < 100mm$ mit $\Delta r = 1mm$ vergrößert. Zu jedem dieser Wertepaare von r und α werden mithilfe der bekannten Rotationsachsen und einfacher trigonometrischer Funktionen die Koordinatendaten von drei Punkten P_1, P_2 sowie P_3 berechnet und diese dann an die Klasse *calcAxis* übergeben. Die Klasse *calcAxis* berechnet aus diesen Daten die Parameter der Rotationsachsen und speichert die Ergebnisse in einer Textdatei.

3.7.1.1 Versuchsdurchführung

Nach Programmstart wird vom Programm *Modell virtuell* eine Programmschleife durchlaufen, die den Drehwinkel α zwischen $0,5° < \alpha < 10°$ in $\Delta\alpha = 0,5°$ Schritten vergrößert. Innerhalb dieser ersten Programmschleife wird dann in einer zweiten Schleife der Abstand r des virtuellen Markers zur Rotationsachse zwischen $1mm < r < 100mm$ in $\Delta r = 1mm$ Schritten variiert. Die X, Y und Z Koordinaten der drei Punkte P_1, P_2 und P_3 werden mittels trigonometrischer Funktionen und den gegebenen Rotationsachsen berechnet. Diese Daten werden dann an die Klasse *calcAxis* übergeben, die wiederum aus den Koordinatenpunkten die Rotationsachsen ermittelt. Insgesamt werden von dieser Simulation 1900 Rotationsachsen aus möglichen Kombinationen von α und r berechnet und in einer Textdatei gespeichert. In Anhang I auf Seite 192 befindet sich ein Auszug des Programmcodes. Es ist zu erwarten, dass bei der Unterschreitung eines minimalen Drehwinkels bzw. eines minimalen Abstands der Marker von der Rotationsachse, die Berechnungsroutinen der Klasse *calcAxis*, bedingt durch die mathematischen Rundungen, einen großen Fehler liefern. Diese minimalen Grenzen galt es zu bestimmen.

3.7.1.2 Versuchsauswertung

Alle Parameter wie Aufpunkt und Richtungsvektor der Rotationsachsen, Drehwinkel und minimaler senkrechter Abstand der Rotationsachsen wurden von der Klasse *calcAxis* korrekt berechnet. Tabelle 4 auf Seite 71 enthält die Grenzwerte des minimalen Abstand r_{min} bei bestimmten Drehwinkeln α für der Fehler $\Delta E_m = \alpha - \alpha^\star < 0,001°$.

Tabelle 4: Ergebnis der virtuellen Validierung. Die Tabelle enthält die Grenzwerte des minimalen Abstandes der Marker zur Rotationsachse r_{min} für diskreten Drehwinkel α, für die gilt $\Delta E_m < 0,001°$. Wie zu erwarten nimmt der minimale Abstand r_{min} für kleinere Drehwinkel α zu.

Drehwinkel $\alpha[°]$	minimaler Abstand $r_{min}[mm]$
1	12,1
2	5,1
3	2,1
4	1,1
5	1,1
6	1,1
...	1,1
20	1,1

Wie zu erwarten war, nimmt der minimale Abstand der Marker zur Rotationsachse r_{min} für kleine Drehwinkel α zu. Bei Drehwinkel $\alpha > 4°$ kann r_{min} nicht weiter

verkleinert werden. Da für die Berechnung mit dem *Einpunktverfahren* Datensätze von jeweils drei Koordinatenpunkten benötigt werden, darf der minimale Drehwinkel α_1 zwischen P_1 und P_2 sowie der Winkel α_2 zwischen P_2 und P_3, zwei Grad nicht unterschreiten. Der Fehler $\Delta E_m < 0,001°$ ist für reale Messungen am mechanischen Modell oder auch in vivo Messungen sicherlich zu vernachlässigen.

3.7.2 Validierung am mechanischen Modell

Um die Messmethode weiter überprüfen zu können, wurden verschiedene Messungen an einem mechanischen Fußmodell[33] durchgeführt. Bei dem mechanischen Modell handelt es sich um eine Konstruktion, die zwei miteinander gekoppelte Gelenkachsen besitzt (siehe Abbidlung 42 auf Seite 73). Ähnlich wie beim menschlichen Fuß können damit kombinierte Bewegungen, z. B. Plantar- und Dorsalflexion oder Inversions- und Eversionsbewegungen um die OSG- bzw. USG Achse durchgeführt werden. Da beide mechanische Achsen des Modells bekannt sind, beziehungsweise mit einfachen mechanischen Verfahren bestimmt werden können, lassen sich die direkt gemessenen Achsen mit den berechneten Achsen der neuen Messmethode vergleichen. Das mechanische Modell bietet verschiedene Vorrichtungen zur Ultraschallsensor und -marker Befestigung. Die Stellung der „USG"-Achse des Modells kann variiert werden, die „OSG"-Achse hingegen ist fest vorgegeben.

3.7.2.1 Versuchsdurchführung

Der erste Schritt bestand in der Durchführung der *direkten* Messung. Dazu wurde das mechanische Fußmodell in eine Neutralstellung gebracht. In dieser Neutralstellung wurden mithilfe des Ulraschalltaststiftes jeweils zwei Punkte auf der OSGA und USGA aufgezeichnet. Anschließend wurde von OSGA und USGA jeweils Aufpunkt und Richtungsvektor berechnet.
In einem zweiten Schritt wurden die Gelenkachsen des Modell mittels der neuen Messmethode bestimmt. Dazu musste zunächst das *Tibiakoordinatensystem* festgelegt werden. Im Anschluss erfolgte eine Rotation um die OSGA, eine Bewegung aus der Neutralstellung in die maximale Dorsalflexion. Der Fuß des mechanischen Modells wurde in dieser Position fixiert, so dass nachfolgende Bewegung im USG nicht von einer Bewegung im OSG überlagert wurde. Es folgten mehrere sich wiederholende Inversions- und Eversionsbewegungen. Um die Ergebnisse der beiden Messungen miteinander vergleichen zu können, mussten die Vektoren aus der *direkten* Messung ebenfalls in das *Tibiakoordinatensystem* transformiert werden. Die

[33] Das Modell wurde an der Universität Freiburg entwickelt und gebaut. Dieses Modell war eines der wichtigsten Werkzeuge während der Entwicklung der neuen Messmethode.

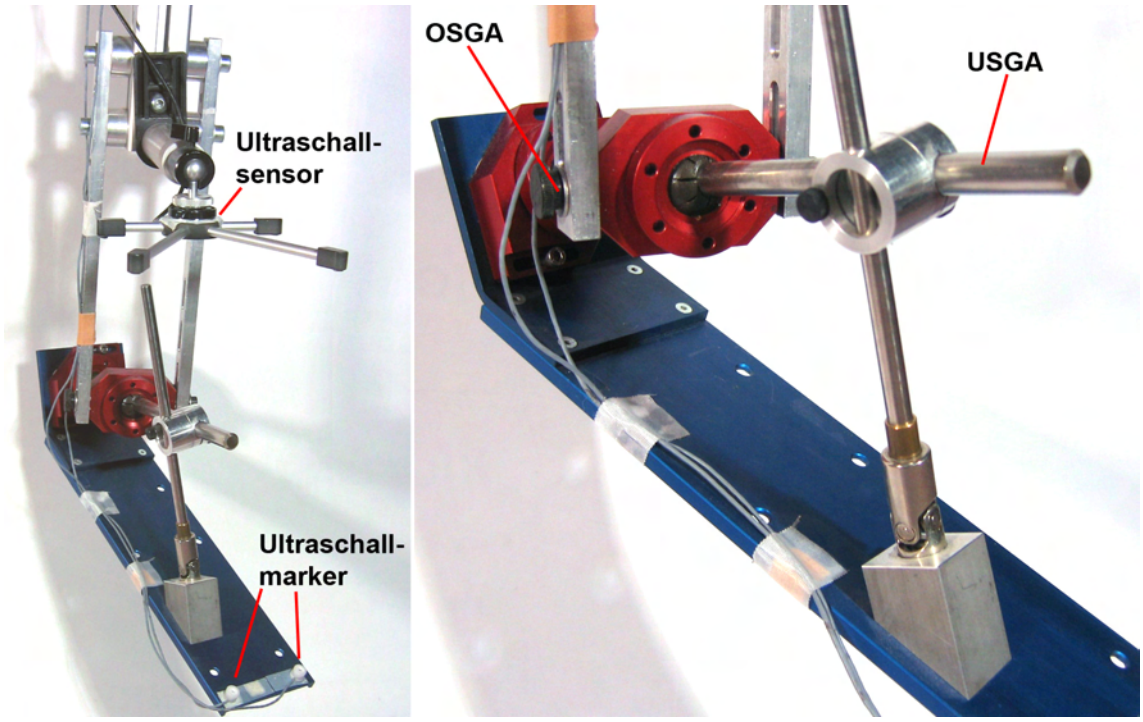

Abbildung 42: Mechanisches Fußmodell. Das linke Bild zeigt die Ultraschallsensor und -marker Befestigung am mechanischen Modell. Der Zebris® *JMA* $11 \cdot 11$ kann an die magnetische Haltevorrichtung angebracht werden. Die Ultraschallmarker werden mit einem doppelseitigen Klebeband am vorderen Teil des Modellfußes befestigt. Das Bild rechts zeigt eine Vergrößerung des Modells. Zu sehen sind beide Gelenkachsen. Die Stellung der als OSGA bezeichnete Rotationsachse kann nicht verändert werden. Die räumliche Orientierung der USGA wird, wie beim anatomischen Vorbild, durch eine Rotation der OSGA verändert. Die Stellung der USGA kann variiert werden.

Bewegungsamplituden der Drehungen um die OSGA bzw. USGA wurden kleiner als $\alpha < 40°$ gehalten. Ähnliche Bewegungsamplituden sind auch bei in vivo Messungen zu erwarten.

3.7.2.2 Versuchsauswertung

Tabelle 5 stellt die direkt gemessenen bzw. mit Hilfe der neu entwickelten Methode bestimmten Ergebnisse der Inklinations- und Deviationswinkel dar. Die Fehler der Winkel und Vektoren betragen (vgl. Tabelle 6):

$E_{Inc} \leq \pm 0,85°$	: Fehler des Inkinationswinkel.
$E_{Dev} \leq \pm 1,34°$	: Fehler des Deviationswinkel.
$\vec{AP}_{X,Y,Z} \leq \pm 2,7mm$	: Fehler des Vektors des Aufpunkts.
$\vec{RV}_{X,Y,Z} \leq \pm 0,005mm$	: Fehler des Richtungsvektors.

Der Gesamtfehler ΔE setzte sich aus dem Fehler der direkten Bestimmung der

Rotationsachsen ΔE_d und dem Fehler ΔE_b der indirekt gemessenen Achse zusammen:

$$\Delta E = \Delta E_d + \Delta E_b$$

Die Abbildungen 43 und 44 auf den Seiten 75 bzw. 75 zeigen weitere Ergebnisse des Validierungsversuchs. Die in den Schaubildern abgebildete blaue Gerade entspricht der *direkt* gemessenen Achse. Die rote Gerade stellt die mithilfe der neuen Messmethode bestimmte Rotationsachse dar. Die mit der entwickelten Messmethode bestimmten Parameter der Rotationsachse stimmen zu einem hohen Maß mit den Ergebnissen der direkt gemessenen Achse überein. Man kann also davon ausgehen, dass die Methode alle relevanten Parameter der Rotationsachsen korrekt wiedergibt.

Tabelle 5: Ergebnisse der Inklinations- und Deviationswinkel des Validierungsversuch am mechanischen Modell. Die mit *direkt* bezeichnete Zeile enthält die mit dem Taststift gemessenen Ergebnisse. Die Zeile *berechnet* enthält die mit der neu entwickelten Methode bestimmten Achsparameter.

	Inklinations- und Deviationswinkel	
	Inc [°]	Dev [°]
direkt	11,65	-4,42
berechnet	12,5	-5,75
ΔE	0,85	-1,34

Tabelle 6: Ergebnisse des Validierungsversuchs am mechanischen Modell, Ergebnisse der Aufpunkte und Richtungsvektoren. Gegenübergestellt sind die Ergebnisse der *direkt* gemessene und der *berechneten* Rotationsachse. Die Zeile ΔE gibt die Differenz der Zeilen *direkt* und *berechnet* an.

	Vektoreinträge der Aufpunkt und Richtungsvektoren					
	AP_x[mm]	AP_y[mm]	AP_z[mm]	RV_x[mm]	RV_y[mm]	RV_z[mm]
direkt	-87,8	212,1	25,8	-0,0382	0,9957	0,0842
berechnet	-87,2	212,7	26,5	-0,0418	0,9958	0,0817
ΔE	2,0	-2,6	1,1	0,0036	-0,0001	0,0025

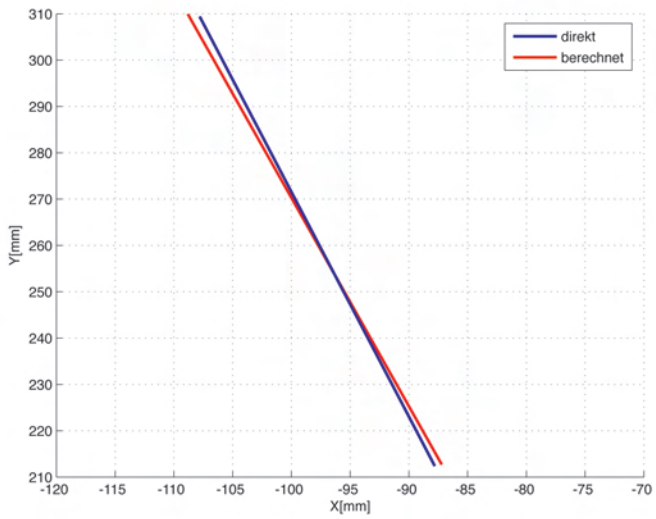

Abbildung 43: Validierungsversuch am mechanischen Modell, Abbildung der direkt gemessenen (blau) und berechneten (rot) Rotationsachsen in der XY Ebene. Die XY Ebene entspricht der Transversalebene, der zwischen der Y-Achse und den Geraden eingeschlossene Winkel dem Deviationswinkel. Um die Unterschiede zwischen direkt gemessener und berechneter Rotationsachse stärker zu verdeutlichen, wurden für die X und Y Achsen verschiedene Skalierungen gewählt.

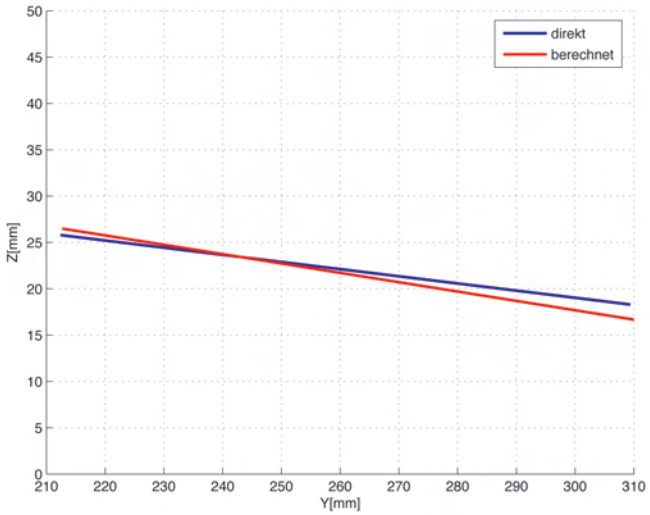

Abbildung 44: Validierungsversuch am mechanischen Modell, Abbildung der direkt gemessenen (blau) und berechneten (rot) Rotationsachsen in der YZ Ebene. Die YZ Ebene entspricht der Sagitalebene, der zwischen der Y-Achse und den Geraden eingeschlossene Winkel dem Inklinationswinkel. Für die Y und Z Achsen wurden ebenfalls verschiedene Skalierungen gewählt, d. h. auch hier sind die Winkel verzerrt.

3.7.3 Validierungsversuch in vivo, MRT-Voruntersuchung

Das entscheidende Problem bei der Bestimmung der Achse des unteren Sprunggelenkes ist die besondere anatomische Situation des Talus. Durch die Lage zwischen Calcaneus und Malleolengabel ist die räumliche Lage des Talus mit nicht invasiven Verfahren nicht objektivierbar. Daraus folgt, dass Relativbewegungen zwischen Talus und Calcaneus nicht gemessen werden können, vielmehr muss für die in vivo Untersuchung eine Situation geschaffen werden für die gilt:

- ◇ Die Relativbewegung zwischen Talus und Tibia/Fibula ist quasi Null.
- ◇ Die Bewegung des Calcaneus relativ zur Tibia/Fibula ist damit nicht von Talus-Bewegungen überlagert.
- ◇ Die Bewegungen in allen anderen Gelenken des Fußes bleiben unberücksichtigt.

Um diese Voraussetzung zu schaffen, wird die Bewegung des Fußes im unteren Sprunggelenk in maximaler Dorsalflexion ausgeführt. Die Form der Trochlea tali wird weitgehend übereinstimmend dahingehend beschrieben, dass der ventrale Teil breiter ist im Vergleich zum dorsalen Teil. Eine Verbreiterung des Abstandes zwischen Tibia und Fibula als Folge der vom Talus übertragenen Momente ist eher unwahrscheinlich. **Fick** geht davon aus, dass es bei einer maximalen Dorsalflexion zu einer Auseinanderdrängung der beiden Unterschenkelknochen von $2-3mm$ kommt. Eine weitere Auseinanderdrängung wird aber durch die Syndesmose verhindert. Der Autor geht davon aus, dass in dieser Stellung „... bei festgestelltem Sprungbein das Kahn- und Würfelbein und mit ihnen der ganze Vorfuß dem Fersenbein im wesentlichen folgen“ [Fick 1911]. Auch **Alt** zeigte in verschiedenen in vitro Versuchen, bei denen er das obere Sprunggelenk mittels Kirschnerdrähten fixierte und die berechneten Ergebnissen aus der Bewegungsanalyse des unteren Sprunggelenks mit den Ergebnisse des durch eine maximale Dorsalflexion fixierten oberen Sprunggelenks verglich, dass bei dieser Stellung das obere Sprunggelenk als festgesetzt betrachtet werden kann [Alt 2001].

In einer in vivo Voruntersuchung sollte überprüft werden, ob bei einer maximalen aktiven Dorsalflexion des Fußes bei gleichzeitiger maximalen Eversion- und Inverisonsbewegung der Talus tatsächlich als festgesetzt betrachtet werden kann. Der Unterschenkel des Probanden wurde dazu in einer speziell für diesen Versuch angefertigten Vorrichtung fixiert. So konnte gewährleistet werden, dass der Unterschenkel sich während der MRT Aufnahmen nicht bewegte (siehe Abbildung 45 auf Seite 77). Die Vorrichtung ermöglichte außerdem, dass der Unterschenkel in einem Winkel von ca. 30° in der MRT-Spule gehalten wurde. Nur so war eine maximale Dorsalflexion des Fußes möglich.

Abbildung 45: Vorrichtung zur Fixierung des Unterschenkels in der MRT Spule. Durch den Winkel von ca. 30° der Vorrichtung ist eine Eversions- und Inversionsbewegung bei maximaler Dorsalflexion des Fußes, ohne Einschränkung durch die MRT-Spule möglich.

3.7.3.1 Versuchsdurchführung

Es wurden jeweils Momentaufnahmen der Sagittalebene und der Koronarebene des Talus in drei verschiedenen Sprunggelenkstellungen (maximale Eversion, neutral Stellung, maximale Inversion) bei maximaler Dorsalflexion gemacht. Durch Aufnahmen in zwei verschiedenen Ebenen kann sichergestellt werden, dass weder ein Absenken in der Sagittalebene noch ein Abkippen in der Transversalebene des Talus stattfindet. Die Dicke l der einzelnen MRT Schichten betrug dabei $l = 6mm$. Die Aufnahmezeit betrug $t = 24s$. Durch die Fixationsvorrichtung konnte auf ein erneutes Kalibrieren des MRT zwischen den einzelnen Aufnahmen verzichtet werden. Dadurch verkürzte sich die Aufnahmezeit. Die Aufnahmen wurden wie folgt durchgeführt:

1. Der Proband sollte aktiv eine maximale Eversion bei gleichzeitiger maximaler Dorsalflexion einnehmen. Der Fuß wurde in dieser Stellung mit Hilfe von Klebebändern fixiert, anschließend die MRT Aufnahmen gemacht. Das Fixieren des Fußes war unbedingt nötig, da es nicht möglich ist, bei Aufnahmezeiten von ca $20s$, den Fuß bewegungslos in einer maximalen Dorsalflexion zu halten.

2. Aus dieser maximalen Eversion sollte der Proband eine Neutralstellung (zwischen maximaler Eversion- und Inversion) bei maximaler Dorsalflexion einnehmen. Der Fuß wurde wiederum in dieser Stellung fixiert. Wieder erfolgten dann die MRT Aufnahmen.

3. In einem letzten Schritt sollte der Proband den Fuß aus der eingenommenen Neutralstellung in eine Stellung der maximalen Inversion bewegen und halten. Der Fuß wurde erneut fixiert und es folgten die abschließenden MRT Aufnahmen.

3.7.3.2 Versuchsauswertung

Die Abbildung 46 auf Seite 80 zeigen ausgewählte Schnittebenen der MRT Aufnahmen des Sprunggelenks. Um eine Bewegung eines dreidimensionalen Körpers registrieren zu können, muss der Körpers in mindestens zwei verschiedenen Schnittebenen beobachtet werden. Um Bewegungen des Talus erfassen zu können, wurden als Schnittebenen die Sagittal- und Transversalebene gewählt. Abbildung 47 auf Seite 81 zeigt die Stellungen des Talus bei einer Supinationsbewegung. Dazu wurden mithilfe eines Grafikbearbeitungsprogramms manuell verschiedenfarbige Geraden durch anatomisch markante Punkte des Talus, der einzelnen Schnitte gelegt gelegt. Während die Geraden der maximalen Eversion und Neutralstellung annähend deckungsgleich sind, kann man bei der Stellung der maximalen Inversion ein minimales Abkippen des Talus nach lateral-kaudal beobachten. Abbildung 48 auf Seite 81 zeigt ein ähnliches Verhalten. Auch in dieser Ebene sind die ersten zwei Geraden annähern kongruent während die dritte Gerade eine Absenkung plantar des Talus zeigt.

Die Erklärung liefert die für diese Bewegung benötigte Muskulatur. Für die Eversion bei maximaler Dorsalflexion sind der Musculus Peroneus und der Musculus Tibialis anterior zu nennen. Um eine Inversionstellung bei maximaler Dorsalflexion einnehmen zu können, wird ebenfalls der Musculus Tibialis anterior, aber auch der Musculus Tibialis posterior benötigt. Der Tibialis posterior übernimmt aber auch gleichzeitig die Funktionen der Plantarflexion. Das bedeutet, bei dem Versuch eine maximale Inversionsstellung einzunehmen, besteht immer die Gefahr, den Fuß plantar abzusenken. Betrachtet man die Ultraschallmarker-Koordinatendaten eines in vivo Versuchs, wird diese Annahme bestätigt. Der auf die XY Ebene (Transversalebene) projizierte Signalverlauf zeigt kaum Abweichungen der einzelnen Bewegungssequenzen (Abbildung 49 auf Seite 82). Die Projektion auf die XZ Ebene (Frontalebene) zeigt ein anderes Bild (Abbildung 50 auf Seite 82). Hier sind Unterschiede der einzelnen Bewegungszyklen der Inversions- und Eversionsbewegung im Signalverlauf zu erkennen. Vor allem in der maximalen Inversions-Stellung (im

Schaubild in Richtung kleinerer X Werte) nehmen die Abweichungen die Z Koordinate betreffend zu.

Insgesamt sind die Bewegungen des Talus relativ gering, so dass er annähernd als festgesetzt betrachtet werden kann. Supinations-, bzw. Pronationsbewegungen bei einer maximalen Dorsalflexion sind dementsprechend Bewegungen des Subtalargelenks. Diese Versuchsanordnung ist damit für die Bestimmung der unteren Sprunggelenkachse in vivo geeignet.

Abbildung 46: MRT Aufnahmen eines rechten Sprunggelenks, die linken drei Bilder zeigen den Schnitt durch den Talus in der Frontalebene jeweils bei maximaler Dorsalflexion. Bild 1 links, max. Eversion; Bild 2 links, Neutrallstellung; Bild 3 links, max. Inversion. Die rechte Bildreihe zeigen das Sprunggelenk in der Sagittalebene. Bild 1 rechts, max. Eversion; Bild 2 rechts, Neutralstellung; Bild 3 rechts, max. Inversion.

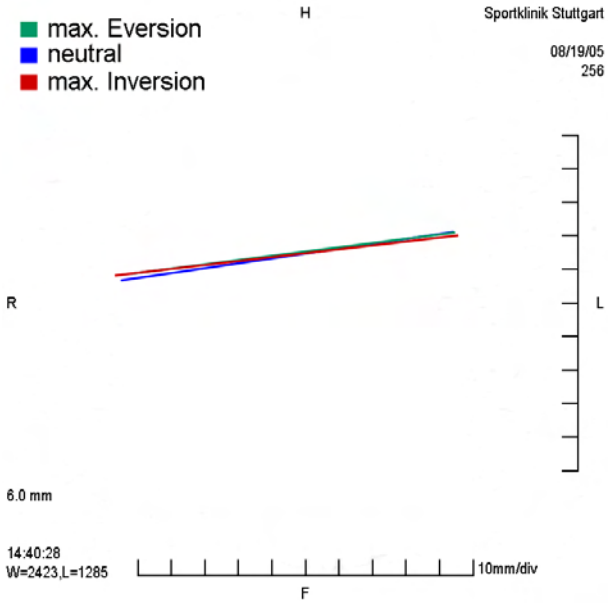

Abbildung 47: Talus Schnitt mit der Frontalebene. Um Bewegungen des Talus zu verdeutlichen, wurden mithilfe eines Grafikbearbeitungsprogramms manuell Geraden durch markante anatomische Punkte des Talus der einzelnen Schnitte gelegt. Die Bilder der MRT Schnitte wurden anschließend ausgeblendet.

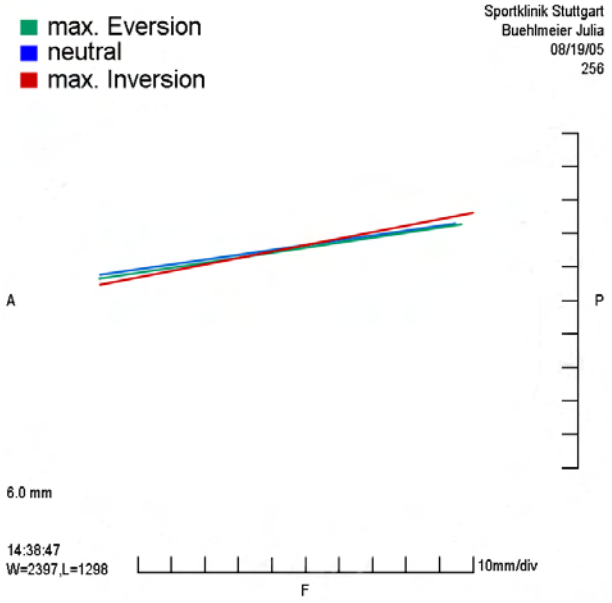

Abbildung 48: Schnitt des Talus mit der Sagittalebene. Um Bewegungen des Talus zu verdeutlichen, wurden ebenfalls Geraden durch anatomisch markante Punkte des Talus gelegt.

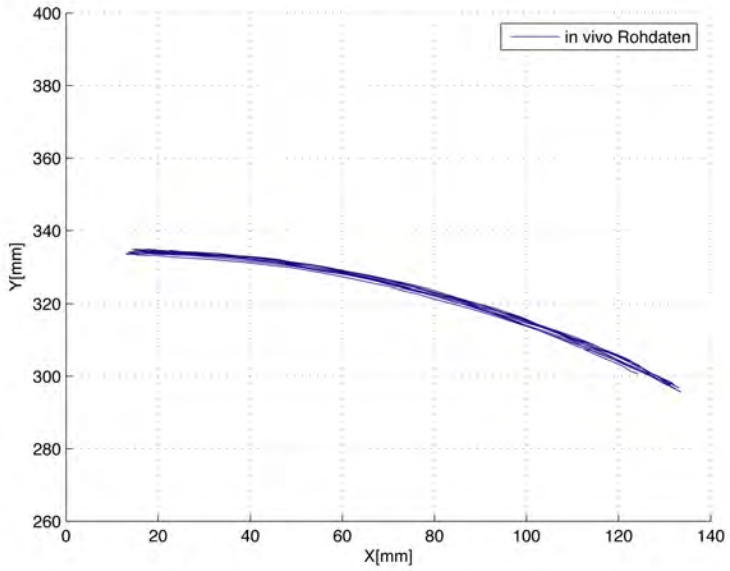

Abbildung 49: XY Koordinatendaten eines in vivo Versuchs. Die Abbildung entspricht der Projektion des Signalverkaufs auf die Transversalebene. Es handelt sich hierbei um den Versuch an einem rechten Sprunggelenk.

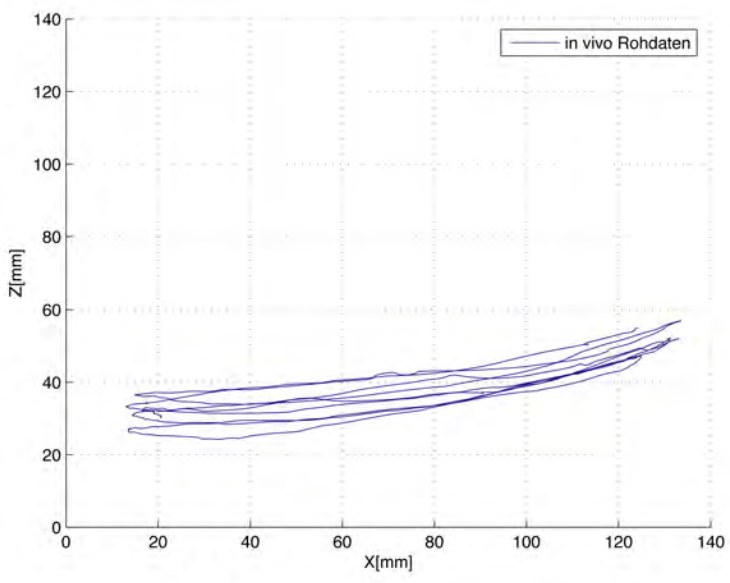

Abbildung 50: XZ Koordinatendaten eines in vivo Versuchs. Zu sehen ist die Projektion des Signalverlaufs auf die Frontalebene. Es handelt sich um den Versuch an einem rechten Sprunggelenk. Die Eversionsstellung liegt bei den maximalen X Koordinaten, die Inversionstellung bei den minimalen. Zu beobachten ist, dass die Abweichungen der einzelnen Kurvenverläufe bei zunehmender Inversionsstellung stärker werden.

3.8 Reliabilitätstest

Um Aussagen über die Zuverlässigkeit bzw. Reliabilität der Methode zur Bestimmung der Sprunggelenkachsen machen zu können, wurde eine Reihe von Versuchen wiederholt durchgeführt. Da Bewegungen am mechanischen Modell gut reproduzierbar sind, wurden zuerst verschiedene Tests am Modell durchgeführt. Mögliche Messungenauigkeiten, die eventuell durch unsaubere Bewegungsausführungen bei in vivo Messungen auftreten könnten, können bei diesen Tests ausgeschlossen werden.

3.8.1 Reliabilitätstest am am mechanischen Modell

Der gesamte Messvorgang zur Bestimmung der Sprunggelenkachsen besteht genau genommen aus zwei Einzelmessungen. In einem ersten Schritt muss das *Tibiakoordinatensystem* aus den Daten der anatomisch markanten Punkte festgelegt werden, anschließend erfolgt die Berechnung der Rotationsachsen aus den aufgezeichneten Bewegungsdaten. Daher ist der nachfolgend beschriebene Versuch in zwei Teile gegliedert.

3.8.1.1 Versuchsdurchführung

Beim ersten Teilversuch wurde das *Tibiakoordinatensystem* lediglich ein einziges Mal festgelegt und anschließend zehnmal die Rotationsachsen des mechanischen Fußmodells bestimmt. So konnten Aussagen über die Messgenauigkeit dieses Teils der Messmethode gemacht werden. Da zu erwarten ist, dass ein Fehler, der bei der Festlegung des *Tibiakoordinatensystems* gemacht wird, ebenfalls das Gesamtergebnis beeinträchtigt, wurde in einem zweiten Teilversuch vor jeder Messung das *Tibiakoordinatensystem* erneut festgelegt. Auch bei diesem Teilversuch wurden insgesamt zehnmal die Gelenkachsen des Modells bestimmt. Um die Rotationsachsen des mechanischen Modells zu ermitteln, wurde wie bereits in Kapitel 3.7.2 auf Seite 72 beschrieben vorgegangen. Der Modellfuß wurde dazu aus der Neutral-Null-Stellung in eine maximale Dorsalflexion bewegt, gefolgt von mehreren aufeinander folgenden Inversions und Eversions Bewegungen um die USGA des Modellfußes. Die aus der Bewegungsanalyse berechneten Achsdaten wurden nach jedem Durchgang gespeichert, anschließend Mittelwert und Standardabweichung berechnet.

3.8.1.2 Versuchsauswertung

Sowohl in Teilversuch 1 als auch in Teilversuch 2 wurden die Koordinaten der Aufpunkte und Richtungsvektoren der USGA berechnet. Außerdem wurden die Inklination, die Deviation, der Hebel und der maximale Drehwinkel bestimmt. Um mögliche

Unterschiede aus den beiden Teilversuchen zu verdeutlichen wurden für die Darstellung der Ergebnisse gleich skalierte Schaubilder verwendet.

Teilversuch 1: Die Richtungsvektoren der gemessenen USGA konnten mit sehr hoher Genauigkeit bestimmt werden. Die Standardabweichung σ_{RV} war für alle drei Vektoreinträge RV_x, RV_y und RV_z immer $\sigma_{RV} < 0,02mm$ (Tabelle 8 auf Seite 85). Demzufolge war ebenfalls eine kleine Standardabweichung der Inklinations- und Deviationswinkel der zehn Einzelmessungen von $\sigma_{1...10} < 1°$ zu erwarten (Tabelle 7 auf Seite 84). Die Standardabweichung σ_{AP} der Aufpunkte war mit $\sigma_{AP} < 2,5mm$ bereits größer. Dieser Umstand verursachte eine ebenfalls größere Standardabweichung $\sigma_L < 0,7mm$ für L, den senkrechten minimalen Abstand der USGA zur Z-Achse, also den Kraftangriffspunkt der Achillessehne. Die Standardabweichung σ_α für den maximalen Drehwinkel um die USGA ist mit $\sigma_\alpha < 3,4°$ relativ hoch. Die Messgenauigkeit wird aufgrund dieser Tatsache allerdings nicht herabgesetzt. Ein großer Wert für σ_α bedeutet lediglich, dass die Bewegungsamplitude der Rotationen um die USGA für die zehn Einzelversuche leicht variiert wurde. Die Bewegungsamplitude variiert in Teilversuch 1 des Reliabilitätstests zwischen einen minimalen Drehwinkel von $\alpha_{min} = 32,88°$ und einem maximalen Drehwinkel von $\alpha_{max} = 41,62°$. Die Ergebnisse des Teilversuchs 1, der Bestimmung der Achsparameter am mechanischen Modell bei einmaligem Festlegen des *Tibiakoordinatensystems*, sind in den Tabellen 7, 8 sowie den Abbildungen 51 und 52 dargestellt.

Tabelle 7: Ergebnisse des Reliabilitätstests Teilversuch 1 am mechanischen Modell. Abgebildet sind die berechneten Werte des Inklinations-, Deviations- und des Drehwinkels, die Strecke L (der minimale senkrechte Abstand der USGA zur Z Achse) der zehn Einzelversuche sowie die daraus berechnete Mittelwerte MW und Standardabweichungen $STABWN$.

	Parameter der USGA			
Nr.	Inc[°]	Dev[°]	L[mm]	α[°]
1	23,85	16,85	21,79	41,62
2	21,32	16,24	22,34	41,56
3	22,7	16,38	22,0	32,88
4	22,9	16,49	21,49	36,81
5	22,8	16,49	22,02	38,11
6	22,53	16,42	21,98	33,68
7	23,06	16,54	22,05	35,2
8	22,94	16,54	21,5	38,03
9	22,76	16,56	21,83	34,39
10	20,79	15,88	22,93	39,57
MW	22,41	16,43	21,80	36,63
STABWN	0,81	0,15	0,64	3,34

Tabelle 8: Ergebnisse des Reliabilitätstests Teilversuch 1, am mechanischen Modell. X, Y und Z Koordinaten der Aufpunkte und Richtungsvektoren der Rotationsachsen. Insgesamt wurde 10 mal die USGA am mechanischen Modell bestimmt. Anschließend Mittelwert und Standardabweichung berechnet. Auf den Seiten 85 und 86 sind diese Daten durch die Abbildungen 51 und 52 grafisch dargestellt.

Aufpunkt und Richtungsvektoren der bestimmten USGA						
Nr.	AP_x[mm]	AP_y[mm]	AP_z[mm]	RV_x[mm]	RV_y[mm]	RV_z[mm]
1	-79,29	187,63	77,66	-0,26	0,87	0,41
2	-79,44	192,2	71,75	-0,26	0,9	0,35
3	-78,86	189,95	73,22	-0,26	0,89	0,38
4	-79,02	190,59	72,57	-0,26	0,89	0,38
5	-79,53	191,34	74,43	-0,26	0,89	0,37
6	-79,48	191,68	74,17	-0,26	0,89	0,37
7	-79,87	190,5	77,14	-0,26	0,89	0,38
8	-79,36	190,97	74,86	-0,26	0,89	0,38
9	-79,45	190,64	76,54	-0,26	0,89	0,37
10	-79,45	194,32	69,1	-0,26	0,91	0,33
MW	-79,37	190,98	74,14	-0,26	0,89	0,372
STABWN	0,26	1,61	2,49	$5{,}5{\cdot}10^{-17}$	0,009	0,019

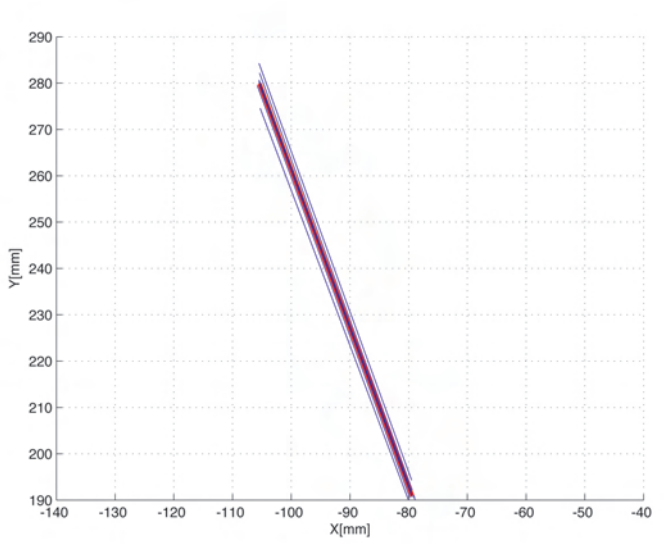

Abbildung 51: Reliabilitätstest Teilversuch 1 am mechanischen Modell. Abbildung XY Ebene der Rotationsachsen bei einmaligem Festlegen des *Tibiakoordinatensystems*. Die XY Ebene entspricht der Transversalebene, der zwischen den Geraden und der Y Achse eingeschlossene Winkel α entspricht dem Deviationswinkel. Die rote Gerade repräsentiert den Mittelwert aus allen zehn Einzelversuchen. Die zehn blauen Geraden stellen jeweils einen der zehn Einzelversuche dar.

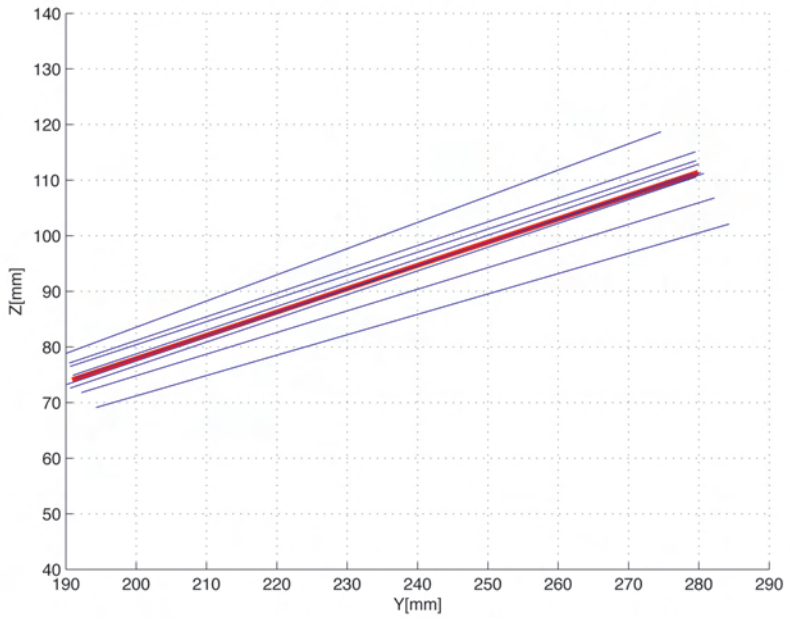

Abbildung 52: Reliabilitätstest Teilversuch 1 am mechanischen Modell. Abbildung YZ Ebene der Rotationsachsen bei einmaligem Festlegen des *Tibiakoordinatensystems*. Die YZ Ebene entspricht der Sagitalebene, der zwischen den Geraden und der Y Achse eingeschlossene Winkel repräsentiert den Inklinationswinkel. Die rote Gerade bildet den Mittelwert aus den zehn Einzelversuchen (blaue Geraden) ab.

Teilversuch 2: Auffällig ist, dass die in den zehn Einzelversuchen ermittelten Rotationsachsen nahezu parallel sind. Dementsprechend fällt die Standardabweichung $\sigma_{\vec{RV}}$für die Richtungsvektoren $\vec{RV}$ mit $\sigma_{\vec{RV}} \leq 0,03mm$ der USGA relativ klein aus. Da sowohl Inklinations- als auch Deviationswinkel direkt aus den Ergebnissen der Richtungsvektoren berechnet werden, erhält man auch hierfür kleine Standardabweichungen, von $\sigma_{Inc} \leq 0,65°$ und $\sigma_{Dev} \leq 0,84°$.
Die Standardabweichung des Inklinations- und Deviationswinkel der OSGA betragen $\sigma_{Inc} \leq 0,64°$ und $\sigma_{Dev} \leq 0,89°$ und besitzen damit eine ähnliche Dimension wie die berechneten Standardabweichungen der Winkel der USGA.
Die Standardabweichung des Vektors des Aufpunktes $\vec{AP}$ liegt bei $\sigma_{\vec{AP}} \leq 2,45mm$ (Tabelle 10 auf Seite 88). Der aus dem Aufpunkt berechnete Hebel L besitzt eine Standardabweichung von $\sigma_L \leq 0,79mm$. $\sigma_\alpha \leq 1,8°$ sagt wiederum lediglich etwas über die Differenz der durchschnittlichen Bewegungsamplituden der zehn Einzelversuche aus.Die Tabelle 10 auf Seite 88 beinhaltet die aus den zehn Einzelmessungen bestimmten Aufpunkte und Richtungsvektoren der OSG- und USG Achsen. In Tabelle 9 auf Seite 88 sind die daraus berechneten Inklinations-, Deviations, Drehwinkel und der Abstand L sowie deren Mittelwerte MW und Standardabweichungen $STABWN$ abgebildet. Die Abbildungen 53 und 54 auf den Seiten 89 und90 stellen diese Ergebnisse grafisch dar.

Der Vergleich aus Teilversuch 1 und Teilversuch 2 des Reliabiltätstests lässt keine nennenswerte Unterschiede betreffend der Größenordnungen der Standardabweichungen erkennen. Das erneute Festlegen des *Tibiakoordinatensystems* wirkte sich nicht negativ auf die Messgenauigkeit der Einzelversuche aus. Der berechnete Messfehler bzw. die daraus resultierende Standardabweichung ist demzufolge das Ergebnis einer leichten Variation der Bewegungsausführung der zehn Einzelversuche. Das Schaubild 55 auf Seite 90 zeigt die Rohdaten einer Messung am mechanischen Modell. Die Daten sind gefiltert aber nicht in das *Tibiakoordinatensystem* transformiert. Die Bewegung bestand aus jeweils fünf Inversions- und fünf Eversionsbewegungen. Zu beobachten sind leichte Abweichungen der Signalkurven der fünf Bewegungszyklen. Bei dem gewählten Datensatz lagen die Standardabweichungen der Winkel $\sigma_{Inc} = \sigma_{Dev} \leq 0.6°$. Die berechneten Standardabweichung aus den zehn Einzelversuchen sind damit nur unwesentlich größer als die eines einzelnen Messdurchgangs.
Das System liefert für Messungen am mechanischen Modell sehr gut reproduzierbare Ergebnisse und ist damit hoch reliabel.

Tabelle 9: Ergebnisse des Reliabilitätstest Teilversuch 2 am mechanischen Modell. Abgebildet sind die Ergebnisse des Teilversuchs 2 des Reliabilitätstests sowie die aus den gesamten Messdurchläufen berechneten Mittelwerte *MW* und Standardabweichungen *STABWN*. In insgesamt zehn Einzelversuchen wurden jeweils die Parameter der OSG- und USG-Achse bestimmt. Dabei wurde vor jedem Einzelversuch das *Tibiakoordinatensystem* erneut festgelegt.

	Parameter der OSGA				Parameter der USGA			
Nr.	Inc.[°]	Dev.[°]	L[mm]	α[°]	Inc.[°]	Dev.[°]	L[mm]	α[°]
1	92,3	87,1	44	20	21,32	16,24	22,34	41,56
2	93,4	84,3	43	18,2	23,27	14,07	22,93	41,55
3	93,9	85,3	43,1	17,6	23,52	15,67	21,76	42,98
4	91,9	84	44	19,9	22,13	13,92	24,82	39,55
5	93,2	83,8	44	18,7	22,6	13,22	23,6	41,46
6	92,3	84,7	44	18,9	22,42	15,02	22,48	38,66
7	92,9	85,6	42	18,2	21,92	14,99	22,85	39,78
8	91,9	84,8	42	17,8	23,23	14,53	23,21	41,03
9	92,4	85,3	43	18,9	22,16	15,12	23,26	40,94
10	92,2	84,8	42	19,2	22,21	14,42	23,57	36,25
MW	92,64	84,97	43,11	18,74	22,478	14,72	23,082	40,376
STABWN	0,64	0,89	0,83	0,77	0,65	0,84	0,79	1,8

Tabelle 10: Ergebnisse des Reliabilitätstest Teilversuch 2 am mechanischen Modell. Berechnete X, Y und Z Koordinaten der Aufpunkte und Richtungsvektoren der Rotationsachsen des Teilversuchs 2 des Relibilitätstest. Insgesamt wurde zehnmal die USGA am mechanischen Modell bestimmt. Dabei wurde vor jeder Messung auch das *Tibiakoordinatensystems* berechnet und anschließend Mittelwert *MW* und Standardabweichung *STABWN* berechnet. Auf den Seiten 85 und 86 sind diese Daten durch die Abbildungen 51 und 52 grafisch dargestellt.

Aufpunkt und Richtungsvektoren der bestimmten USGA						
Nr.	AP_x[mm]	AP_y[mm]	AP_z[mm]	RV_x[mm]	RV_y[mm]	RV_z[mm]
1	-70,15	195,05	72,62	-0,21	0,9	0,38
2	-73,08	191,97	70,22	-0,22	0,9	0,37
3	-76,13	190,7	75,46	-0,25	0,89	0,39
4	-71,94	191,21	74,97	-0,22	0,9	0,39
5	-73,92	192,83	70,38	-0,23	0,9	0,37
6	-76,41	192,23	75,39	-0,24	0,9	0,37
7	-73,84	192,57	73,42	-0,23	0,89	0,38
8	-75,38	192,74	73,71	-0,24	0,9	0,36
9	-74,96	192,17	73,78	-0,24	0,9	0,37
0	-79,44	192,2	71,75	-0,26	0,9	0,35
MW	-74,52	192,3	73,17	-0,23	0,89	0,37
STABW	2,44	1,09	1,81	0,014	0,004	0,012

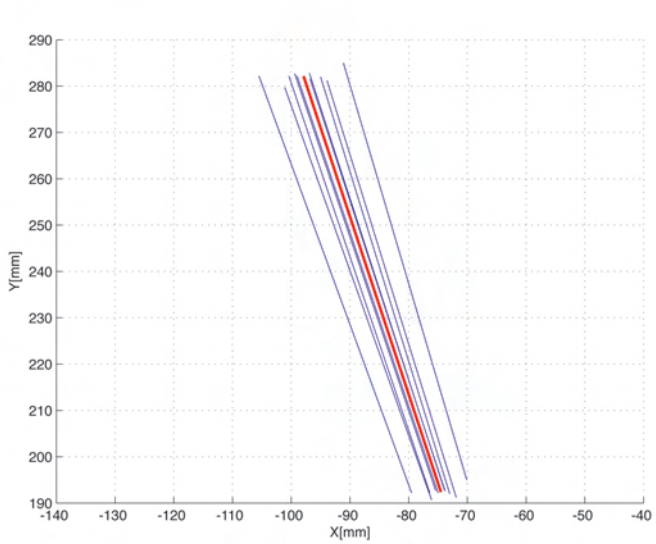

Abbildung 53: Reliabilitätstest Teilversuch 2 am mechanischen Modell. XY Ebene des Teilversuchs 2 zur Bestimmung der Reliabilität der Messmethode, bei erneutem Festlegen des *Tibiakoordinatensystems*. Der zwischen den Geraden und der Y Achse eingeschlossene Winkel repräsentiert den Deviationswinkel. Die X,Y Ebene stellt die Transversalebene dar. Die blauen Geraden bilden die Ergebnisse der zehn Einzelversuche ab, die rote Gerade den Mittelwert. Auffällig ist, dass die Richtungen der Geraden sehr gut übereinstimmen, d.h. die berechneten Inklinations- und Deviationswinkel sind mit einem sehr kleinen Fehler behaftet.

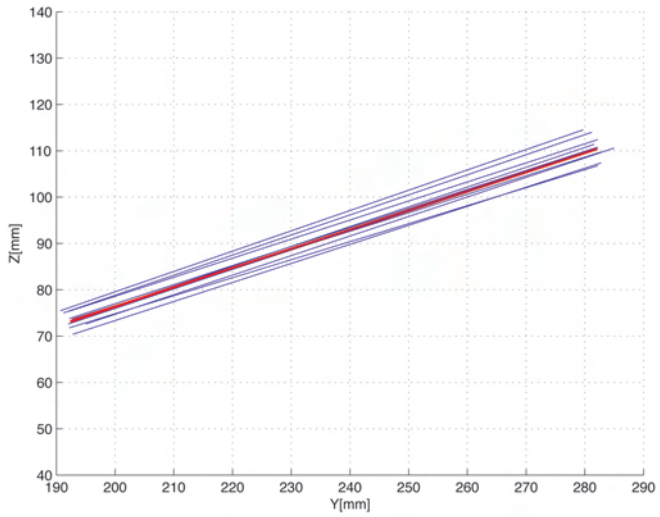

Abbildung 54: Reliabilitätstest Teilversuch 2 am mechanischen Modell. YZ Ebene des Teilversuchs 2 zur Bestimmung der Reliabilität bei erneutem Festlegen des *Tibiakoordinatensystems*. Die XY Ebene entspricht der Sagitalebene. Der zwischen den Geraden und der Y Achse eingeschlossene Winkel repräsentiert den Inklinationswinkel. Auch in der Projektion auf die Sagitalebene sind die abgebildeten Geraden nahezu parallel. Die zehn blauen Geraden bilden die Ergebnisse der zehn Einzelmessung aus Tabelle 10 auf Seite 88 ab. Die rote Gerade entspricht dem daraus berechneten Mittelwert.

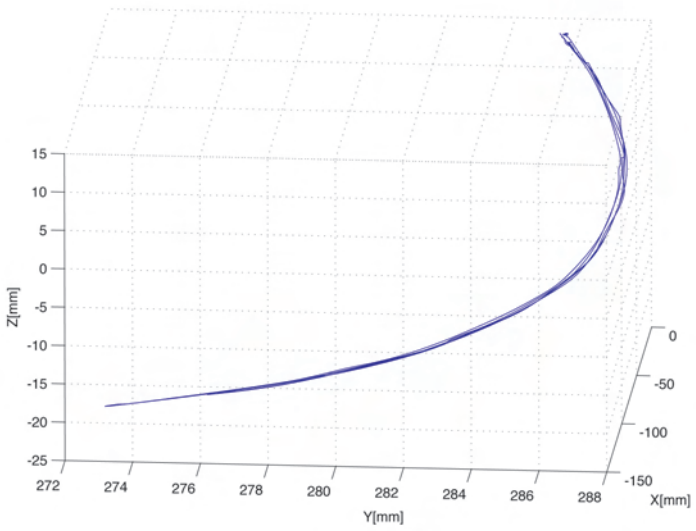

Abbildung 55: Grafische Darstellung der Rohdaten einer Messung am mechanischen Modell. Die Daten wurden gefiltert, aber nicht in das *Tibiakoordinatensystem* transformiert. Der Signalverlauf zeigt jeweils fünf Inversions- und Eversionsbewegungen. Zu sehen sind minimale Abweichungen der einzelnen Bewegungsausführungen.

3.8.2 Reliabilitätstest in vivo

Es ist davon auszugehen, dass bei in vivo Versuchen Schwierigkeiten auftreten, die am mechanischen Modell keine oder nur eine untergeordnete Rolle spielen.

- ◇ Sowohl die Ultraschallmarker als auch die Sensoren könnten durch mögliche Hautverschiebungen einen zusätzlichen Fehler verursachen.
- ◇ Das Aufzeichnen markanter anatomischer Punkte und damit das Festlegen des *Tibiakoordinatensystems* lässt sich im Unterschied zum mechanischen Modell nicht absolut reproduzierbar festlegen.
- ◇ Im Gegensatz zum Modell, lässt sich die OSGA nicht fixieren, das bedeutet wiederum, dass die Bewegungszyklen im USG einer größeren Variation unterworfen sein könnten.

Um Aussagen machen zu können, wie der gesamte Messaufbau die Messgenauigkeit beeinflusst, mussten daher verschiedene in vivo Versuche durchgeführt werden.

3.8.2.1 Versuchsdurchführung

Analog zum Versuch am mechanischen Modell wurden beim in vivo Versuch ebenfalls die Achsen des oberen und unteren Sprunggelenks mehrmals hintereinander bestimmt. Dazu wurden die Sensoren mit dem auf Seite 66 in Abbildung 37 gezeigten Prototypen an der Tibia befestigt. Die Ultraschallmarker wurden wie in Abbildung 38 auf Seite 66 gezeigt am Probandenfuß fixiert. Anschließend wurde das *Tibiakoordinatensystem* in der Neutral-Null-Stellung aufgezeichnet. Aus dieser Neutral-Null-Stellung folgte eine Bewegung des Fußes in die maximale Dorsalflexion. Anschließend sollte der Proband mehrere Inversions- und Eversionsbewegungen in der maximalen Dorsalflexion ausführen. Dieser komplette Vorgang wurde wiederum zehnmal hintereinander wiederholt. Aus den Ergebnissen der Einzelversuche wurden die Mittelwerte und Standardabweichungen der Parameter der OSG und USG Achsen berechnet.

3.8.2.2 Versuchsauswertung

Wie zu erwarten war, zeigten die im in vivo Versuch ermittelten Rotationsachsen eine wesentlich höhere Streuung als die im vergleichbaren mechanischen Experiment bestimmten. Tabelle 11 und 12 auf den Seiten 92 und 93 enthalten die berechneten

Parameter sowie die Aufpunkte und Richtungsvektoren der bestimmten Rotationsachsen. Die Standardabweichung des Richtungsvektors $\sigma_{\vec{RV}}$ ist mit $\sigma_{\vec{RV}} \leq 0,03mm$ annähernd doppelt so groß wie die Standardabweichung der Richtungsvektoren aus dem Versuch am mechanischen Modell. Dasselbe gilt für die Standardabweichung der Vektoren des Aufpunktes $\sigma_{\vec{AP}} \leq 4.8$. Demzufolge sind die Standardabweichungen der Inklinations- und Deviationswinkel $\sigma_{Inc} = \sigma_{Dev} \leq 2,35°$ fast viermal so groß wie die des Versuchs am Modell. Die Abbildungen 56 und 57 zeigen die Projektion der Rotationsachsen auf die Transversal- bzw. Sagitalebene. Die Schaubilder lassen eine wesentlich höhere Streuung der Gelenkachsen um ihren berechneten Mittelwert erkennen. Das Diagramm 58 auf Seite 94 zeigt die nicht transformierten Rohdaten eines Einzelversuchs. Verglichen mit den im Schaubild 55 auf Seite 90 abgebildeten Rohdaten des mechanischen Versuchs, lassen sich beim in vivo Test doch erhebliche Unterschiede in den einzelnen Bewegungszyklen erkennen. Berechnete Ergebnisse, die mit der doppelten Standardabweichung vom Mittelwert abweichen, werden wie schon erwähnt als Ausreißer behandelt und bei einer erneuten Berechnung von Mittelwert und Standardabweichung nicht miteinbezogen. Die größere Standardabweichung bei allen berechneten Parametern lässt sich durch die ungenauere Bewegungsausführung beim in vivo Versuch begründen.

Tabelle 11: Berechnete Parameter des Reliabilitätstests in vivo. Abgebildet sind die Ergbnisse der Einzelmessungen sowie die Mittelwert- und Standardabweichungen der Inklinations- , Deviations- und Drehwinkel der USGA, desweiteren der minimale senkrechte Abstand L der USGA zur Z Achse des Koordinatensystems. Spalte α beinhaltet die Drehwinkel um die OSGA. Spalte S enthält die minimalen Abstände der OSGA zur USGA.

	OSGA	Parameter der USGA				
Nr.	α[°]	Inc[°]	Dev[°]	L[mm]	β[°]	S[mm]
1	46,9	8,48	-9,32	20,66	33,25	41,25
2	53,8	9,1	-3,64	8,66	39,13	40,76
3	48,2	8,9	-10,91	14,12	34,32	44,21
4	44,8	8,59	-8,9	9,31	38,78	41,05
5	55,7	3,73	-5,84	8,8	35,79	46,89
6	47,2	9,3	-11,4	14,78	36,21	45,81
7	46,9	7,62	-10,34	13,57	35,66	43,78
8	49,2	10,10	-9,43	13,44	33,78	44,56
9	55,9	3,43	-6,31	12,64	32,09	50,41
10	49,8	8,75	-9,23	7,39	32,17	51,33
MW	49,84	7,8	-8,53	12,33	35,13	45,02
STABW	3,74	2,19	2,35	3,75	2,35	3,51

Tabelle 12: Ergebnisse des Reliabilitätstests in vivo. Berechnete X, Y und Z Koordinaten der Aufpunkte und Richtungsvektoren der Rotationsachsen sowie deren Mittelwerte und Standardabweichungen.

Aufpunkt und Richtungsvektoren der bestimmten USGA in vivo						
Nr.	AP_x[mm]	AP_y[mm]	AP_z[mm]	RV_x[mm]	RV_y[mm]	RV_z[mm]
1	32,66	259,83	41,03	0,11	0,96	0,06
2	42,54	276,37	41,25	0,15	0,95	0,15
3	42,33	267,19	35,96	0,16	0,96	0,15
4	44,68	278,17	39,43	0,18	0,96	0,13
5	46,77	271,18	35,43	0,16	0,94	0,14
6	39,21	269,31	36,43	0,1	0,96	0,15
7	47,41	260,89	39,89	0,1	0,98	0,16
8	38,21	279,12	35,56	0,15	0,96	0,13
9	41,45	261,56	37,98	0,16	0,95	0,14
10	49,45	268,12	31,31	0,14	0,95	0,14
MW	42,47	269,17	35,43	0,14	0,96	0,15
STABW	4,71	6,74	2,93	0,03	0,01	0,01

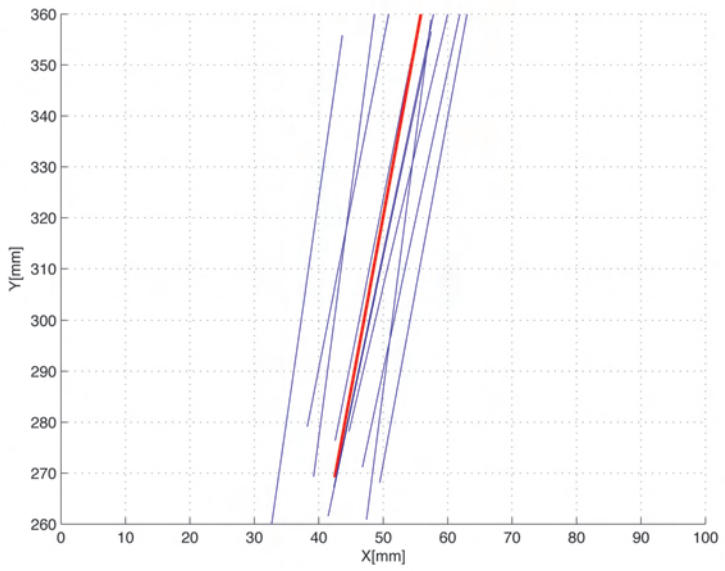

Abbildung 56: XY Ebene des Reliabilitätstest in vivo. Der zwischen den Geraden und der Y Achse eingeschlossenen Winkel entspricht dem Deviationswinkel. Die XY Ebene repräsentiert die Transversalebene. Die rote Gerade stellt den Mittelwert aus den zehn blauen Geraden der zehn Einzelversuche dar. Die Abweichungen vom Mittelwert sind wesentlich größer als beim Versuch am mechanischen Modell.

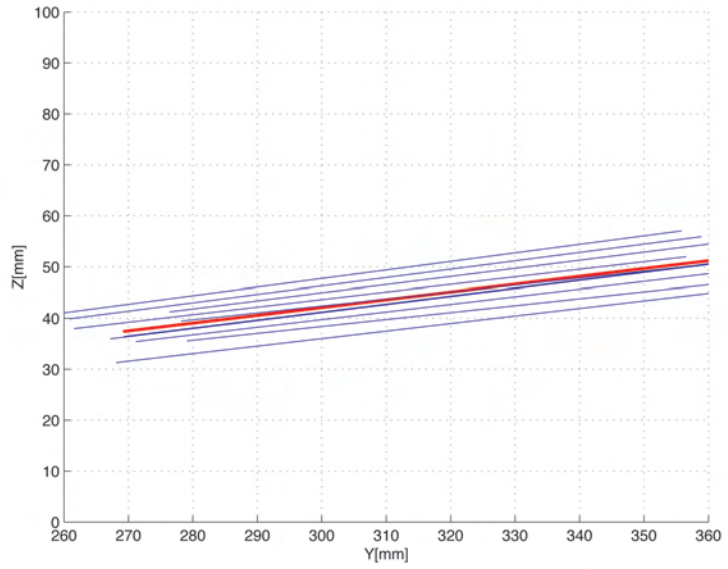

Abbildung 57: YZ Ebene des Reliabilitätstests in vivo. Der Inklinationswinkel wird von der Y Achse und den Geraden eingeschlossen. Zu beobachten ist wiederum eine größere Abweichung der blauen Geraden der zehn Einzelversuche vom Mittelwert (rote Gerade) als beim vergleichbaren Versuch am mechanischen Modell.

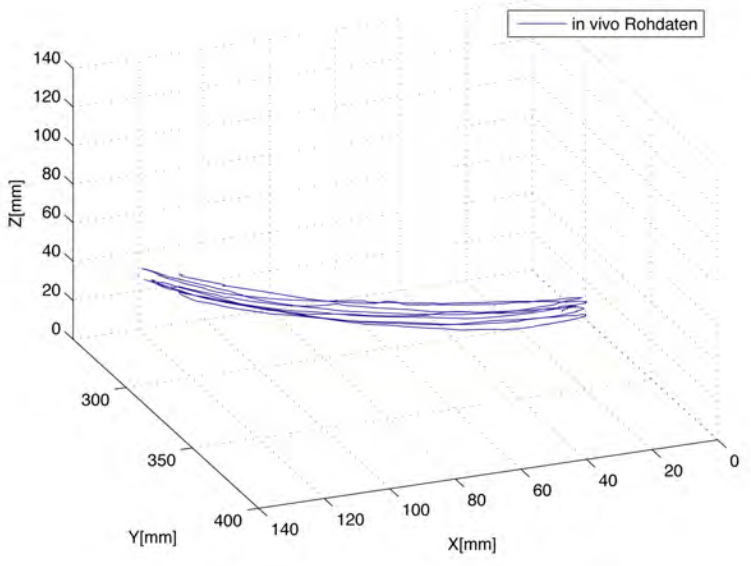

Abbildung 58: 3D-Darstellung der Rohdaten einer Messung in vivo. Die Daten wurden tiefpass gefiltert und in das *Tibiakoordinatensystem* transformiert. Die Standardabweichungen der Winkel σ_{Inc} und σ_{Dev} war bei diesem Messdurchgang $\sigma_{Inc} = \sigma_{Dev} \leq 4°$. Werte größer als die doppelte Standardabweichung werden allerdings vom Programm als Ausreißer behandelt.

3.9 Fehlerabschätzung

Um Aussagen über die Messunsicherheit des gesamten Messverfahrens machen zu können, wurden verschiedene Tests bzw. Versuche durchgeführt (Kapitel 3.8). Dieser Gesamtmessfehler, der als rein zufällig angenommen wird, setzt sich aus den der einzelnen Komponenten des Gesamtsystems zusammen. Zufällige Messfehler werden durch Streuungen der Einzelmessungen verursacht. Ursache der zufälligen Messfehler sind Fluktuationen von Ereignissen, die den Messprozess beeinflussen, jedoch nicht kontrollierbar sind. Der Gesamtmessfehler der neuen Methode setzt sich demnach wie folgt zusammen:

$$\Delta E \approx \sqrt{(\Delta E_{Zebris})^2 + (\Delta E_{math})^2 + (\Delta E_{koord})^2 + (\Delta E_{move})^2} \tag{48}$$

mit:

$\Delta E_{Zebris} \leq 0,001°$:	Entsprechend dem von der Firma Zebris® angegebene Messfehler der X,Y und Z Koordinaten des CMS 20 in Kombination mit dem JMA $11 \cdot 11$ Sensor.
$\Delta E_{math} \leq 0,001°$:	Der durch Rundungsfehler verursachte Fehler der mathematischen Methode (Tabelle 4 auf Seite 71).
$\Delta E_{koord} \leq 2°$:	Geschätzer Fehler beim Festlegen des *Tibiakoordinatensystems* in vivo (aus den Ergebnissen des Reliabilitätstests in vivo, Kapitel 3.8.2 auf Seite 91).
$\Delta E_{move} \leq 5°$:	Geschätzer Fehler verursacht durch eine ungenaue Bewegungsausführung bei der Messung in vivo (aus den Ergebnissen des Reliabilitätstests in vivo, Kapitel 3.8.2 auf Seite 91).

Man erhält eine Messungenauigkeit ΔE für das gesamte Messverfahren von $\Delta E \leq 5,0°$, d.h. der zufällige Messfehler ΔE_{Zebris} des CMS 20, verursacht durch thermisches Rauschen und der Messfehler ΔE_{math}, verursacht durch Rundungsfehler der Berechnungsroutine kann vernachlässigt werden. Die Qualität der Ergebnisse der Deviations- und Inklinationswinkel der Gelenkachsen wird vornehmlich durch die Güte der Bewegungsausführung und ebenso durch die Präzision des Festlegens des *Tibiakoordinatensystems* bestimmt. Die Ergebnisse des Reliabilitätstest am mechanischen Modell (Kapitel 3.7.2 auf Seite 72) haben gezeigt, dass bei einer möglichst exakten Bewegungsausführung selbst bei erneutem Festlegen des *Tibiakoordinatensystems*, ein Fehler der Winkel von $\Delta_{Inc} = \Delta_{Dev} \leq 1,0°$ erreicht werden konnte. Es kann angenommen werden, dass die X, Y und Z Koordinaten der Ultraschallmarker bei den Bewegungen am mechanischen Modell mit einer Messungenauigkeit von $\Delta x = \Delta y = \Delta z \leq 2mm$ angegeben werden konnten. Durch einen erneuten Versuch

an einem verbesserten mechanischen Modell, das präzisere Bewegungswiederholungen ermöglicht[34], könnte gezeigt werden, dass schon bei Messungen am Modell der größte Fehler durch eine ungenaue Bewegungsausführung zustande kommt.
Der Vergleich von Schaubild 55 auf Seite 90 mit Schaubild 58 auf Seite 94 bestätigt diese Annahme. Zu sehen sind die Rohdaten von jeweils fünf Bewegungszyklen des Versuchs am mechanischen Modell und des in vivo Tests. Beim Versuch am mechanische Modell sind kaum Abweichungen des Signalverlaufs der einzelnen Bewegungszyklen zu erkennen. Im Gegensatz dazu zeigt der Test in vivo leicht unterschiedliche Signalverläufe der einzelnen Bewegungsabfolgen.

Nach Aussage des Zentralen Grenzwertsatzes der Statistik dürfen die zufälligen Fehler zumindest approximativ als normalverteilt betrachtet werden. Aus diesem Grund werden vom Programm für jede Bewegungssequenz die Parameter der Gelenkachsen bestimmt und anschließend Mittelwert und Standardabweichung berechnet. Die so bestimmten Mittelwerte bilden die gesuchten tatsächlichen Größen wesentlich besser ab. Des Weiteren kann anhand der Standardabweichungen eine Aussage über die Übereinstimmung der sich wiederholenden Bewegungssequenzen und damit über die Qualität der ermittelten Ergebnisse gemacht werden.

[34] Vorzustellen wäre der Einsatz eines Roboters, der ebenfalls Rotationen um zwei miteinander gekoppelten Gelenkachsen, mit höherer Genauigkeit durchführen könnte.

4 Anwendung des Verfahrens in vivo

Seit der Fertigstellung eines ersten, im Feld einsetzbaren Prototypen wurde die neue Methode zur Bestimmung der Sprunggelenkachsen in einer Anzahl von Studien eingesetzt. Es handelte sich dabei um eine Reihe von Untersuchungen, die alle am Institut für Sportwissenschaft der Universität Stuttgart vom Arbeitsbereich Biomechanik durchgeführt wurden und sich mit dem Thema Prävention und Rehabilitation von Sprunggelenkverletzungen befassten. Durch den frühzeitigen Einsatz dieser Methode konnten etwaige Schwierigkeiten oder Probleme mit Hard- oder Software in der nächsten Entwicklungsphase berücksichtigt werden. Obwohl die Berechnungsroutinen ständig verändert bzw. verbessert wurden, konnten die mit einer frühen Programmversion eingezogenen Rohdaten dennoch verwendet werden. Da die vom Programm gespeicherten Rohdaten lediglich die nicht transformierten X, Y und Z Koordinaten der Ultraschallmarker enthalten, werden die Gelenkachsenparameter beim Öffnen der Rohdaten mit einer überarbeiteten Programmversion erneut berechnet.

4.1 Versuchsdurchführung in vivo

Die Methode zur Bestimmung der Sprunggelenkachsen kam wie erwähnt bei mehreren Studien zum Einsatz. Insgesamt wurden die Daten von ca. 150 Probanden erfasst. Nur Datensätze, deren Standardabweichung der Deviation oder Inklination eines Gelenks kleiner als $\sigma < 5°$ war, wurden in die Auswertung mit aufgenommen. Die $n = 97$ verbleibenden Datensätze setzten sich wie folgt zusammen:

- ◇ Prospektiver Ansatz: 41 Datensätze aus dem vom *BISP* geförderten Projekt „Fußball interdisziplinär: Zur Optimierung der Prävention, Rehabilitation und Wiederverletzungsprophylaxe von Verletzungen im Fußball“. Beteiligte Mannschaften waren die U18 vom: 1. FC Köln (6), Bayern München (7), HSV Hamburg (20), Bayer Leverkusen (8) und Mönchengladbach (7). Ziel dieses Projektes war es, eine möglichst große Datenbank mit Daten aus verschiedenen sportmotorischen Tests bzw. anthropometrischen Merkmalen von Nachwuchsathleten zu generieren. Möglicherweise lassen sich einige der erfassten Parameter als verletzungsbegünstigend identifizieren und damit Sprunggelenkverletzungen durch geeignete Maßnahmen im Voraus vermeiden.
- ◇ Retrospektiver Ansatz: 27 Datensätze aus der von der *ARAG* Sportversicherungen unterstützten Studie. Hierbei wurden die Sprunggelenkachsen von Probanden mit zurückliegenden Sprunggelenkverletzungen bestimmt. Dabei musste es sich um vollständig ausgeheilte Verletzungen handeln, da sonst eine

Inversion/Eversionsbewegung bei gleichzeitig maximaler Dorsalflexion nur bedingt möglich wäre (dieser Sachverhalt würde zu einer Verfälschung des Ergebnisses führen).

◇ 22 Probandendaten stammen aus einer Orthesenstudie die für die Firma *Otto Bock* durchgeführt wurde. Dabei wurde die Wirkung von verschiedenen Orthesen mithilfe einer Plattform zur Verletzungssimulation getestet. Auch hier gilt zu beachten, dass verschiedene Achslagen, speziell des USGs, bei gleicher Verletzungssimulation, bei den einzelnen Probanden unterschiedliche Reize auslösen.

Der kritischste Punkt bei der Untersuchung bestand in der Durchführung der für die Berechnung der Sprungelenkachsen notwendigen willkürlichen Sprungelenkbewegung. Die Beibehaltung einer maximalen Dorsalflexion bei gleichzeitiger Inversion/Eversion stellte eine hohe koordinative Anforderung dar. Aus diesem Grund wurde diese Bewegung von jedem Probanden vor der Versuchsdurchführung ausreichend geübt. Anschließend erfolgte, analog zu den Reliabilitätstest in vivo (auf Seite 91),

1. das Anbringen der Ultraschallsensoren an der Tibia,
2. das Befestigen der Ultraschallmarker am Calcaneus,
3. das Festlegen des *Tibiakoordinatensystems* in der Neutral-Null-Stellung (hüftbreiter, aufrechter Stand),
4. die Bewegung des Fusses aus der Neutral-Null-Stellung in eine maximale Dorsalflexion,
5. eine fünfmalige Wiederholung der Inversion- und Eversionsbewegung in der maximalen Dorsalflexion.

Waren die Ergebnisse unbefriedigend, die Standardabweichungen der Winkel grösser als $\sigma > 10°$, wurde die Messung wiederholt. Bei einigen Probanden war es allerdings trotz mehrfach durchgeführter Messung nicht möglich, ein ausreichend genaues Messergebnis zu erhalten. Deren Daten wurden nicht weiter berücksichtigt.

4.2 Ergebnisse des Verfahrens in vivo

Die Standardabweichung des Deviationswinkels war bei fast allen Messungen kleiner als die des Inklinationswinkels. Die Abbildungen 59 und 60 auf den Seiten 99 und 100 zeigen die Projektionen auf die Transversal- (XY-Ebene) bzw. Frontalebene (XZ-Ebene) des Ultraschallsignals einer exemplarischen Messung. Zu beobachten ist, dass der auf die XY-Ebene projizierte Signalverlauf kaum Abweichungen zwischen den einzelnen Bewegungsausführungen aufweist. Demzufolge ist auch der auf die XY-Ebene abgebildete Winkel der Deviation mit einem minimalen Fehler behaftet. Der auf die XZ-Ebene projizierte Signalverlauf lässt dagegen einige Abweichungen der einzelnen Bewegungsausführungen erkennen. Auffallend ist eine zunehmende plantare Flexion bei einer wachsenden Supination. Der Fuss scheint im Verlauf der Messung immer weiter abgesenkt zu werden, was vermutlich auf eine Ermüdung der beteiligten Muskulatur zurückzuführen ist (mehr dazu in Kapitel 5.2.2 ab Seite 106). Der auf die Sagitalebene projizierte Inklinationswinkel ist dementsprechend ebenfalls mit einem größeren Fehler behaftet. Abbildung 64 auf Seite 102 stellt die Ergebnisse der Deviationswinkel der USGA nochmals dar. Die ermittelten Deviationswinkel sind approximativ normalverteilt, wobei sowohl positive als auch negative Winkel gemessen wurden. Tabelle 13 beinhaltet die Ergebnisse von USG- und OSGA der 97 Probanden, in den Abbildung 62 und 63 sind diese grafisch dargestellt.

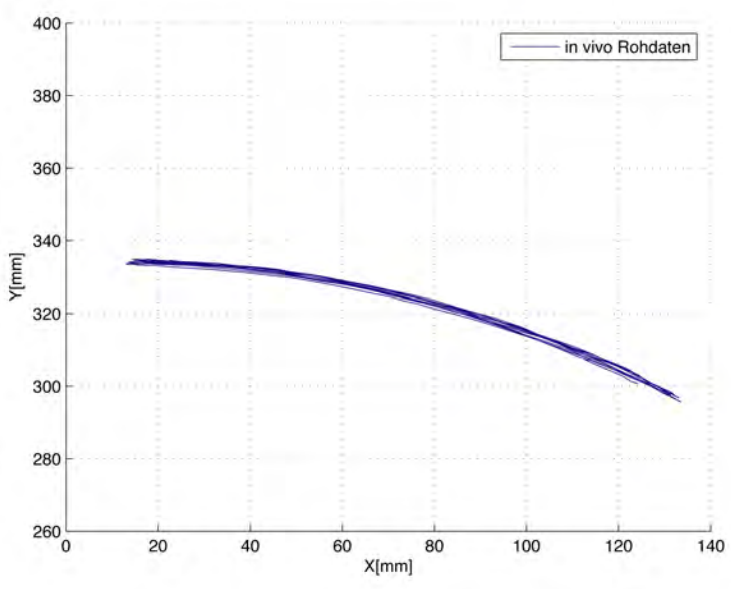

Abbildung 59: Ultraschalldaten einer exemplarischen Messung projiziert auf die Transversalebene. Da der Signalverlauf kaum Abweichungen der einzelnen Bewegungssequenzen erkennen lässt, ist auch der daraus berechnete Deviationswinkel mit einem minimalen Fehler behaftet.

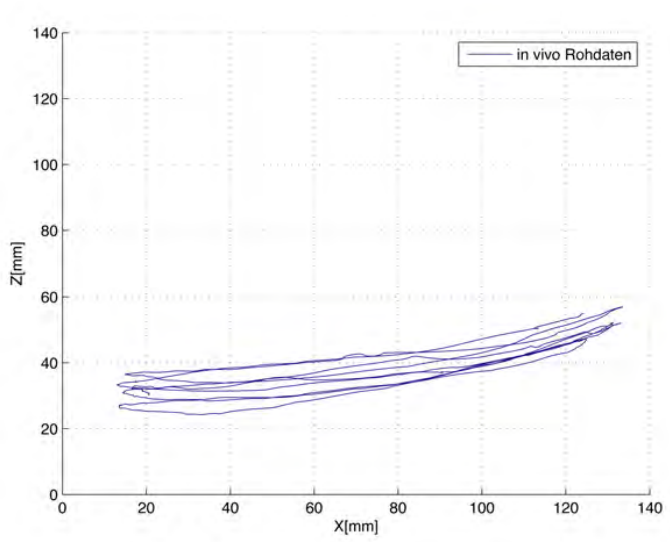

Abbildung 60: Ultraschalldaten einer exemplarischen Messung projiziert auf die Frontalebene. Zu sehen ist eine Abweichung im Signalverlauf zwischen den einzelnen Bewegungssequenzen. Dadurch kommt es auch zu einem höheren Fehler bei der Berechnung des Inklinationswinkels.

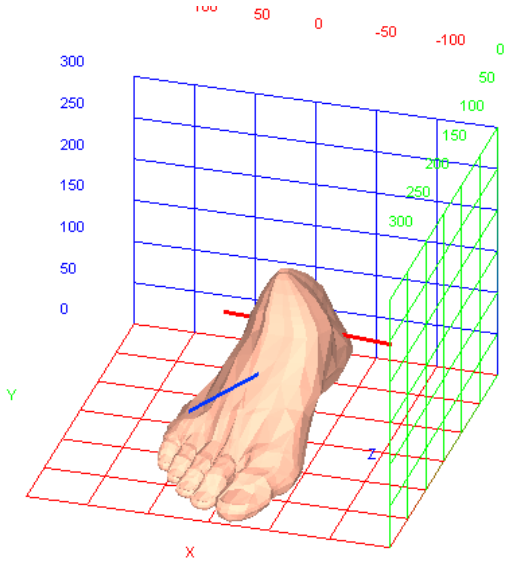

Abbildung 61: Dreidimensionale grafische Darstellung der Ergebnisse der entwickelten Software. Schaubilder lassen sich durch die Funktion *copy2clipboard* exportieren. Achse blau: USGA, Achse rot: OSGA.

Tabelle 13: Ergebnisse für das obere und untere Sprunggelenk. In vivo Messung mit $n = 97$. Die Standardabweichung aller Parameter der in die Auswertung eingegangenen Einzelmesssungen war $\sigma < 10°$.

OSGA

	Mittelwert	Standardabweichung	Minimum	Maximum
Inc[°]	81,18	7,22	55,7	100,2
Dev[°]	78,18	8,0	55,80	94,8
α[°]	37,35	13,12	9,20	68,90

USGA

	Mittelwert	Standardabweichung	Minimum	Maximum
Inc[°]	28,05	16,04	2,56	64,27
Dev[°]	3,43	10,12	-17,80	30,45
β[°]	38,38	10,18	8,94	57,70

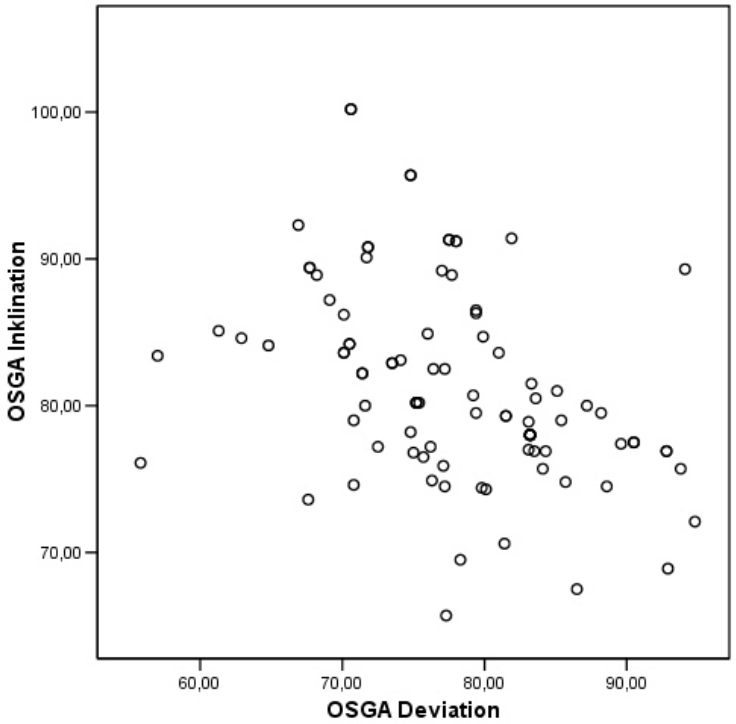

Abbildung 62: Schaubild Inklination/Deviation der OSGA. In Vivo Messung mit $n = 97$. Die Standardabweichung jedes Messwertes war $\sigma < 10°$

Das als *CSV* Datei gespeicherte Datenblatt enthält außer den Daten zu Proband und Messung noch weitere gelenkachsenspezifische Parameter, die nicht auf der

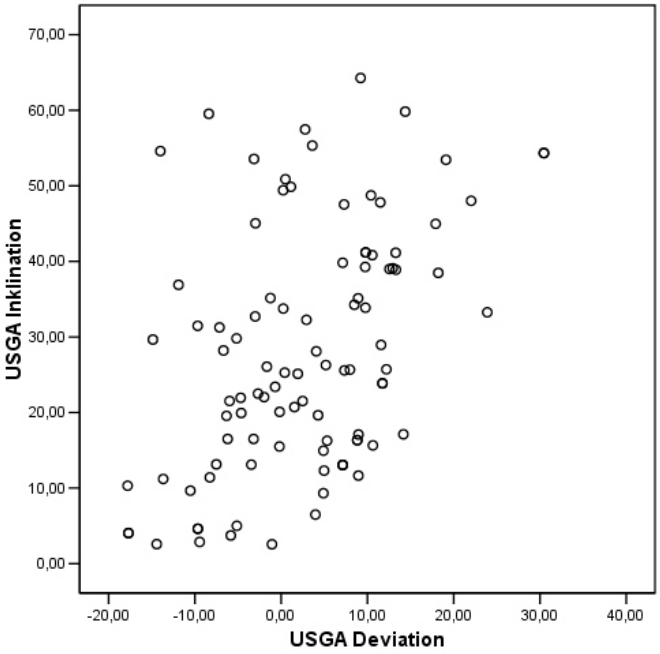

Abbildung 63: Schaubild Inklination/Deviation der USGA. In vivo Messung mit $n = 97$. Die Standardabweichung jedes Messwertes war $\sigma < 10°$.

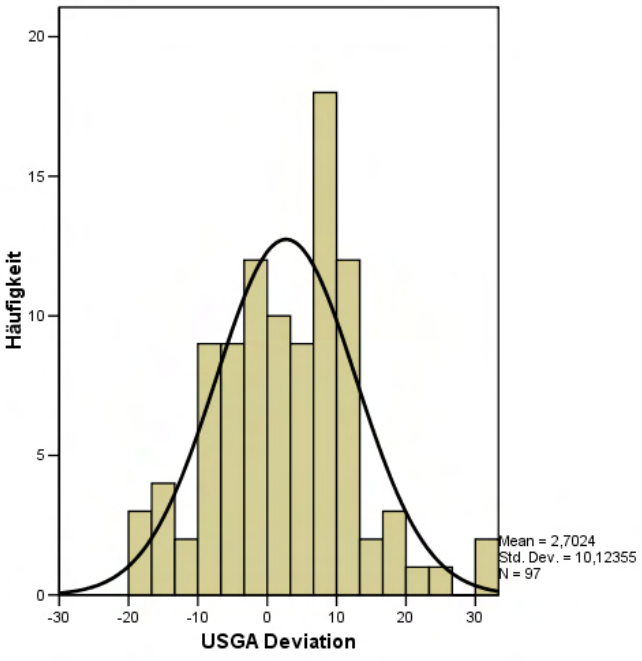

Abbildung 64: Verteilung der Deviationswinkel der USGA. Die ermittelten Werte sind approximativ Normalverteilt. Sowohl positive als auch negative Winkel wurden gemessen.

Programmoberfläche ausgegeben werden:

- ◇ Die Achsparameter der einzelnen Bewegungswiederholungen, unterteilt nach Inversions- und Eversionbewegungen.
- ◇ Ausreißer werden gekennzeichnet dargestellt.
- ◇ Minimaler senkrechter Abstand von OSG- und USG Achse zur Z Achse (Kraftangriffspunkt der Achillessehne).
- ◇ Minimaler senkrechter Abstand von OSGA zu USGA.

Möglicherweise kann im Laufe der Anwendung dieser neue Methoden einer dieser zusätzlichen Parameter als Risikofaktor identifiziert werden.

5 Diskussion

Bisher entwickelte Verfahren, die eine nicht invasive, in vivo Bestimmung der Sprunggelenkachsen ermöglichten [Alt 2001, van den Bogert et al. 1994], benötigten in der Regel Laborbedingungen, da zur Datenakquisitation hauptsächlich 3D-Vidoeanalysesysteme eingesetzt wurden; somit waren diese Verfahren nicht im Feldversuch einsetzbar. In der vorliegenden Arbeit wurde ein diagnostisches Verfahren entwickelt, das eine Bewegungsanalyse der Sprunggelenkachsen ermöglicht, welche ebenfalls in vivo und nicht invasiv ist, jedoch im Unterschied zu den bisherigen Diagnoseverfahren keine Laborbedingungen benötigt und direkt im Feld einsetzbar ist. Die neu entwickelte Methode wurde dabei verschiedenen Validierungsversuchen und Reliabilitätstests unterzogen, deren Ergebnisse ab Kapitel 3.7 auf Seite 69 dargestellt sind. Die Resultate der ersten Messungen in vivo sind in Kapitel 4 abgebildet. Nachfolgend werden diese Ergebnisse diskutiert.

5.1 Zu den Validierungsversuchen

Die Validierungsversuche aus Kapitel 3.7.1 und 3.7.2 zeigten, dass sowohl die mathematische Methode als auch das Gesamtmesssystem in der Lage ist, die Parameter der Sprunggelenkachsen zu bestimmen. Der Versuch am virtuellen Modell (Kapitel 3.7.1 auf Seite 69 diente lediglich der Überprüfung der mathematischen Funktionen und speziell dem verwendeten Verfahren zur Berechnung von Rotationsachsen. Es zeigte sich, dass die experimentelle Umsetzung weder mathematische noch logische Fehler enthält. Die beim Unterschreiten eines bestimmten Markerabstands r oder Rotationswinkel α erhaltenen Fehler $\Delta E < 0,001°$, basieren lediglich auf Rundungsfehler und sind für die Praxis irrelevant.
Das Gesamtmesssystem, die Kombination aus Software und Hardware, wurde mithilfe der Validierungsversuche am mechanischen Modell überprüft (Kapitel 3.7.2 auf Seite 72). Hier zeigte sich, dass die vom Programm ermittelten Rotationsachsenparameter sehr gut mit den am mechanischen Modell eingestellten Achsen übereinstimmte (Tabellen 5 und 6). Zwar war der Fehler der Winkel mit $\Delta E_{Inc,Dev} < \pm 1,34°$ relativ gross, es bleibt allerdings die Frage offen, welche der gemessenen Achsen, die direkt gemessenen oder die indirekt bestimmte, exakter ermittelt werden konnte. Das manuelle Fixieren der Achsmittelpunkte am mechanischen Modell, bei der direkten Messung, stellte sich als relativ problematisch heraus. Es ist deshalb anzunehmen, dass die direkt gemessene Achse mit einen wesentlich höheren Fehler behaftet ist, als die mit der neu entwickelten Methode bestimmten. Nichtsdestoweniger hat auch dieser Versuch gezeigt, dass das gesamte Messdesign zur Bestimmung der Sprunggelenkachsen geeignet ist.

5.1.1 Validierung in vivo - MRT Untersuchung

Im MRT Versuch (Kapitel 3.7.3 auf Seite 76) sollte bestätigt werden, dass der Talus in der maximalen Dorsalflexion weitgehend durch die Malleolengabel fixiert wird. Nur so ist eine Bewegung im USG, die nicht von eine Rotation der OSGA überlagert ist, möglich. Da alle nachfolgenden Berechnungen auf der Annahme einer isolierten Bewegungsanalyse des USGA basieren, wurde der von **Alt** durchgeführte in vitro Versuch [Alt 2001] erneut durch einen in vivo Versuch bestätigt. Dabei wurde der Proband angewiesen, jeweils drei Fußstellungen (die maximale Eversion, eine Mittelstellung und die maximale Inversion) willkürlich einzunehmen und für die Dauer der MRT Aufnahme zu halten. Lediglich in der maximalen Inversion zeigte sich ein minimales Absenken des Talus. Dieses Absenken bei der Inversionsbewegung zeigten sich ebenfalls im Signalverlauf bei ersten in vivo Versuchen der neuen Methode. Als Konsequenz daraus sollte bei der Bewegungsausführung der Fokus weniger auf eine größtmögliche Amplitude der Inversions/Eversionsbewegung sondern vielmehr auf das Halten der maximalen Dorsalflexion gerichtet werden. Bei einer sorgfältig ausgeführten Bewegung um die Achse des unteren Sprunggelenks kann das Absenken des Talus minimiert und damit vernachlässigt werden. Eine Inversions/Eversionsbewegung in maximalen Dorsalflexion kann demzufolge als eine annähernd isolierte Bewegung im USG betrachtet werden (Abschnitt 5.2.2 auf Seite 107).

5.2 Zu den Reliabilitästests

5.2.1 Am mechanischen Modell

Zur Überprüfung der Reliabilität wurden zwei verschiedenen Versuche am mechanischen Modell durchgeführt (Kapitel 3.8 auf Seite 83). Jeweils zehn mal wurden die Achsen des mechanischen Modells bestimmt. Im ersten Versuch wurde das *Tibiakoordinatensystem* nur ein einziges Mal festgelegt, während beim anschließenden Versuch das *Tibiakoordinatensystem* vor jeder Messung erneut festgelegt wurde. Die Fehler der Winkel oder Vektoren beider Versuche ließen keine signifikanten Unterschiede erkennen und waren mit $\Delta E_{Inc,Dev} < 0,9°$ für die Winkel und $\Delta E_{\vec{AP},\vec{RV}} < 2,5mm$ zufriedenstellend gering (Tabellen 7, 8, 9 und 10). Der Hauptanteil des Fehlers resultiert mit Sicherheit aus einer ungenauen Bewegungsreproduktion. Durch eine leichte Variation der Neutralstellung, der Bewegungsfreiheit der mechanischen Gelenke sowie einer möglichen minimalen Veränderung der Achsstellungen, bedingt durch die manuelle Bewegung des mechanischen Modells, könnte dieser Fehler verursacht worden sein.
Der Vergleich der Fehler für Winkel und Vektoren des OSGA und USGA zeigen, dass am mechanischen Modell das Festlegen des *Tibiakoordinatensystems* extrem reproduzierbar ist (siehe Tabelle 9 auf Seite 88).

5.2.2 In vivo

Abschließend wurde ein weiterer Reliabilitätstest in vivo durchgeführt. Dabei wurde bei einem Probanden jeweils zehnmal die Achsen des Sprunggelenks bestimmt. Vor jedem Versuch wurde das *Tibiakoordinatensystem* erneut festgelegt. Im Vergleich zu den Versuchen am mechanischen Modell waren die Fehler mit $\Delta E < 5°$ wesentlich größer. Das erneute Festlegen des *Tibiakoordinatensystems* wird mit einem Fehler von $\Delta E_{tibia} < 2°$ angenommen. Fehlerquellen, die am mechanischen Modell nur eine untergeordnete Rolle spielten, waren die Marker- und Sensorbefestigungen. Bei nicht invasiven in vivo Versuchen kann eine Marker- und Sensorverschiebung nicht ausgeschlossen werden. Durch eine Verbesserung der Markerbefestigung und der Wahl eines kleineren und damit leichteren Sensors (z.b. dem *WinJaw* $8 \cdot 8$) könnte dieser Fehler weiter veringert werden. Der auf Seite 66 abgebildete Schuh zur Ultraschall-Marker-Befestigung könnte z. B. durch eine Fersenkappe oder Fersenklammer ersetzt werden. So könnten die Ultraschall-Marker über eine einfache Konstruktion an der Probanden-Ferse befestigt werden. Mögliche Bewegungsübertragungen vom Vorfuß könnten damit ausgeschlossen werden. Die Konstruktion sollte möglichst leicht sein, trotzdem stabil und den Stand in der Neutral-Null-Stellung zulassen.
Die Hauptfehlerquelle beim in vivo Versuch liegt in der ungenauen Bewegungsausführung der Inversion und Eversion in der maximalen Dorsalflexion. Abbildung 58 auf Seite 94 zeigt den Signalverlauf eines geübten Probanden. Es lassen sich im Vergleich zum Versuch am mechanischen Modell (Abbildung 55 auf Seite 90) größere Abweichungen der einzelnen Bewegungssequenzen erkennen. In einer frühen Prototypenphase wurden die Sprunggelenksachsparameter aus einer einzigen Inversionsbewegung berechnet. Wurde diese einzige Bewegung unsauber ausgeführt, waren die daraus berechneten Rotationsachsenparameter mit einem dementsprechend großen Fehler behaftet. Durch eine Wiederholung der Inversions/ Eversionsbewegung, einem Berechnen der Achsparameter für jede dieser Einzelbewegungen und anschließendem Ermitteln der Mittelwerte und Standardabweichungen dieser Bewegungsphasen konnte der durch eine unsauber ausgeführte Bewegung verursachte Fehler minimiert werden. Da angenommen werden kann, dass der zufällige Fehler der Einzelbewegungen annähernd normalverteilt ist, sollte eine Messung mindesten fünf Inversion- und Eversionbewegungen enthalten. Allgemein gilt, je häufiger die Inversion- und Eversion durchgeführt werden können, desto steiler wird die Kurve der Normalverteilung, was wiederum bedeutet, desto exakter entspricht der errechnete Mittelwert dem tatsächlichen Wert. In in vivo Versuchen zeigten die meisten Probanden allerdings nach mehrmaliger Wiederholung der Inversion/ Eversionbewegung in der maximalen Dorsalflexion Ermüdungserscheinungen. Ab einer bestimmten Anzahl an Wiederholungen kann der Fuss nicht durchgehend in der maximalen Dorsalflexion gehalten werden. Die Bewegungsausführung wird dadurch ungenauer und somit nimmt der Fehler der Messgrößen wieder zu.

In verschiedenen Versuchen haben sich zehn Wiederholungen als optimal heraus gestellt.
Allgemein gibt es bei in vivo Messungen folgendes zu beachten:

- ◇ Das *Tibiakoordinatensystem* muss sorgfältig festgelegt werden. Da die erfassten Parameter relativ zum *Tibiakoordinatensystem* angegeben werden, setzt sich der gesamte Messfehler aus dem Fehler des *Tibiakoordinatensystems* und dem Fehler aus der Messung der Sprunggelenkachsen zusammen:
  $\Delta E_g = \Delta E_{Tibia} + \Delta E_{Messung}$. Die anatomischen Punkte müssen möglichst exakt in der Neutralstellung des Probanden (hüftbreiter, aufrechter Stand) markiert werden.

- ◇ Bei der Bestimmung der OSGA Parameter, speziell dem Drehwinkel, sollte darauf geachtet werden, dass der Start der Messung aus der bei der Bestimmung des *Tibiakoordinatensystems* eingenommen Neutral-Null-Position des Probanden erfolgt.

- ◇ Mindestens fünf, höchstens zehn Wiederholungen der Inversions/Eversionsbewegung sollten zur Messung der USGA durchgeführt werden. Besondere Aufmerksamkeit sollte dabei auf die Einhaltung der maximalen Dorsalflexion während der Durchführung der Inversionbewegung gerichtet werden.

Des Weiteren hat sich bewährt, die Probanden vor einer Messung die Inversions/ Eversionsbewegung in der maximalen Dorsalflexion üben zu lassen. Während der Messung half eine leichte Bewegungsführung durch einen schwachen Druck gegen die Fußsohle des Probanden, die Pantarflexion zu minimieren. Werden diese Punkte beachtet und die Messung sorgfältig durchgeführt, sind ohne weiteres reproduzierbare Ergebnisse mit Standardabweichungen von $\sigma < 5°$ möglich. Kleine Standardabweichungen bedeuten nicht nur eine hohe Reliabilität, sondern ebenfalls eine hohe Validität der Messergebnisse der USGA. Bei einer kombinierten Bewegung aus einer Rotation um die OSGA und USGA erhält man eine wesentlich höhere Abweichung des Signalverlaufs der einzelnen Bewegungswiederholungen der Inversion- bzw. Eversionsbewegung. Ist die Standardabweichung der ermittelten Parameter klein, bedeutet dies, dass bei der durchgeführten Bewegung der Talus in der Tibia - Fibula Knochengabel festgesetzt war, es sich also um eine reine Rotationsbewegung um die USGA handelte. Es gilt für die Genauigkeit bzw. den relativen Fehler der ermittelten Parameter

$$\Delta F_{Dev} < \Delta F_{\vec{RV}} < \Delta F_{Inc} < \Delta F_{\vec{AP}} < \Delta F_{Hebel}$$

mit,

ΔF_{Dev}: relativer Fehler des Deviationswinkel,
ΔF_{Inc}: relativer Fehler des Inklinationswinkel,
$\Delta F_{\vec{RV}}$: relativer Fehler des Richtungsvektor der Rotationsachse,
$\Delta F_{\vec{AP}}$: relativer Fehler des Vektors zum Aufpunkt der Rotationsachse,
ΔF_{Hebel}: relativer Fehler des minimalen senkrechten Abstand der Rotationsachse zur Z Achse (Kraftangriffspunkt der Achillessehne).

Das bedeutet, dass z. B. der relative Fehler der Deviation der USGA am geringsten ist, also die Messgenauigkeit am höchsten, der Hebel (senkrechter minimaler Abstand der USGA zum Kraftangriffspunkt der Achillessehne) dagegen am wenigsten genau bestimmt werden kann.

5.3 Zur Anwendung in vivo

Seit der Fertigstellung eines ersten im Feld einsetzbaren Prototypen wurden die Methode zur Bestimmung der Sprunggelenkachsen in verschiedenen Studien eingesetzt. Insgesamt wurden dabei wie erwähnt $n = 150$ Sprunggelenkachsen ermittelt. Da es in den Anfangsphasen noch zu Umstrukturierungen des Messablaufs kam, konnten nicht alle dieser erhobenen Datensätze für die Auswertung verwendet werden. Datensätze, deren Standardabweichung größer als $\sigma > 5°$ war, wurden ebenfalls nicht berücksichtigt. In die Auswertung wurden $n = 97$ Achsdaten aufgenommen. Damit ist n bereits wesentlich größer als bei den in vivo Studien von **Alt** [Alt 2001] und **van den Bogert** [van den Bogert et al. 1994]. In Tabelle 14 sind die Mittelwerte und Standardabweichungen für die Inklination und Deviation dieser in vivo Studien sowie die Ergebnisse der hier beschriebenen ersten Anwendung in vivo abgebildet.

Tabelle 14: Ergebnisse andere Autoren im Vergleich zu den in dieser Arbeit erhaltenen Ergebnisse für die Inklination und Deviation der USGA.

Autor, Jahr	n	Inc. [°]	STAB Inc. [°]	Dev. [°]	STAB Dev. [°]
van den Bogert et al.	14	37,4	2,7	18,0	16,2
Alt	22	36,0	7,3	7,8	9,8
Hochwald	97	28,1	15,9	3,4	11,4

Im Vergleich zu den in vitro durchgeführten Studien [Manter 1941, Hicks 1953, van Langelaan 1983], deren Ergebnisse für die Inklination bzw. Deviation bei $Inc \approx 43°$ und $Dev \approx 20°$ liegen, scheinen die erhobenen Werte der Deviation bei

den in vivo Untersuchungen geringer zu sein. Die in vitro durchgeführten Untersuchungen wurden an isolierten Knochen durchgeführt. Lage und räumliche Orientierung der USGA basieren ausschließlich auf der Geometrie der subtalaren Gelenkflächen. Für die Relativbewegungen zwischen Talus und Calcaneus sind besonders die Strukturen im Sinus und Canalis tarsi von Bedeutung [Alt 2001]. Dies könnte die mögliche Ursache für die Differenz der in vivo erhobenen Befunde sein.
Abweichungen den Deviationswinkel betreffend lassen sich auch zu den Studien von **van den Bogert** und **Alt** beobachten. Der von **van den Boger** erhaltene Wert für die Deviation ist vermutlich auf die geringe Probandenzahl zurückzuführen. Möglicherweise rühren geringe Unterschiede in den Winkeln auch von unterschiedlich definierten Relativkoordinatensysteme her. Während **Alt** für die Festlegung des *Tibiakoordinatensystems* zwei Punkte auf der Tibiakante und einen weiteren Punkt parallel zum zweiten Strahl markiert, wird in dieser Arbeit der gesamte Vorfuß ausgeschlossen. Als weiteren Punkt zur Berechnung des *Tibiakoordinatensystems* wird ein Punkt auf der Achillessehnenmitte gewählt. Dadurch wird eine höhere Reproduzierbarkeit und Objektivität beim Festlegen des Koordinatensystems erreicht. Abbildung 65 zeigt die auf die Transversalebene projizierten Y Achsen zweier Relativkoordinatensysteme. Da der Deviationswinkel relativ zu diesen Koordinatenachsen angegeben wird, gibt es durchaus minimale Differenzen zwischen den beiden unterschiedlich festgelegten Koordinatensystemen.

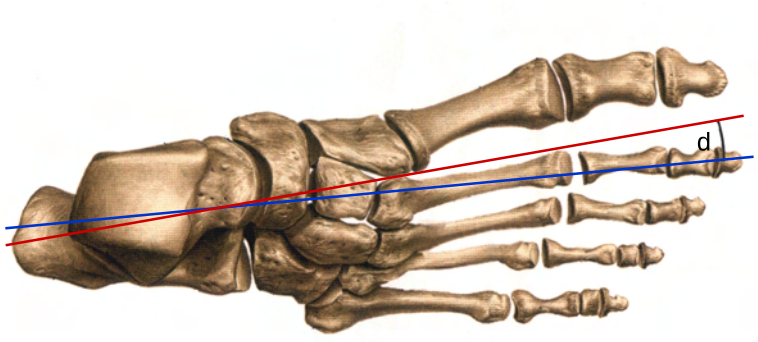

Abbildung 65: Differenz beim Festlegen des *Tibiakoordinatensystems*. Die blaue und rote gerade entprechen der Projektion der Y Achse der zwei Koordinatensysteme. Bei beiden Geraden wurden dieselben zwei Punkte auf der tibiakante markiert. Die blaue Gerade benutzt als dritten anatomischen Punkt einen Punkt auf dem zweiten Strahl. Die rote Gerade verwendet einen Punkt auf der Achillessehnenmitte. Demzufolge werden im *roten Tibiakoordinatensystem* kleinere Deviationswinkel ausgegeben. Abbildung modifiziert aus Sobota „Atlas der Anatomie des Menschen“ [Sobota 2000].

Mit Sicherheit sind die Deviationswinkel in vivo kleiner als die aus den in vitro Versuchen bestimmten Winkel. Abbildung 64 auf Seite 102 zeigt die Verteilung der Deviation bei $n = 97$ Probanden. Diese Verteilung lässt durchaus Spekulationen zu, der Mittelwert der Deviation könnte bei Stichproben ($n > 100$) weiter minimiert

werden.
Die Standardabweichung der Deviation nimmt auch bei großen Stichproben nicht ab. Die sowohl in in vitro als auch in vivo Studien gefundenen interindividuelle Varianz der Deviation wurde erneut bestätigt.

Diese Tatsache lässt zwei Interpretationen zu:

1. Es gibt eine Beziehung zwischen den interindividuellen Achsstellung des USG und Parameter der Beingeometrie, z.B. mit der Trittspur. Möglicherweise muss die Summe aus Deviationswinkel und Trittspur betrachtet werden.

   $$\alpha_{DevG} = \alpha_{Dev} + \alpha_{Tritt}$$

   Unter bestimmten Umständen würde sich die Standardabweichung bzw. die interindividuelle Varianz der Deviation verringern lassen, nämlich dann, wenn eine nach lateral rotierte Fuß/Beinstellung einen größeren Deviationswinkel der USGA ausprägen würde. Abbildung 66 soll diesen Sachverhalt verdeutlichen. Obwohl im ersten Fall ein negativer im zweiten Fall ein positiver Deviationswinkel in Bezug zum *Tibiakoordinatensystem* gemessen wurde, sind beide Winkel relativ zur Körpersagittalebene angegeben vergleichbar. Die interindividuelle Varianz der Deviation ist durch diese Betrachtungsweise minimiert worden. Als Konsequenz daraus müssten zur Identifizierung von verletzungsbegünstigenden Faktoren nicht nur die Parameter der USGA sondern auch die Aspekte der Beingeometrie erfasst werden.

2. Die interindividuelle Varianz die Deviation betreffend lässt sich durch eine Kombination der Achslage des USG und der Beingeometrie nicht weiter minimieren. Möglicherweise lässt sich erst bei großen Stichproben ein statistischer Zusammenhang zwischen bestimmten Achsparametern und Anfälligkeit für akute Sprunggelenkverletzungen oder chronische Überlastungsschäden finden. Eventuell müssen dennoch Faktoren der Beingeometrie betrachtet werden. Es ist durchaus vorzustellen, dass diskrete Kombinationen aus USGA Stellungen und Beingeometrie als verletzungsbegünstigend identifiziert werden können.

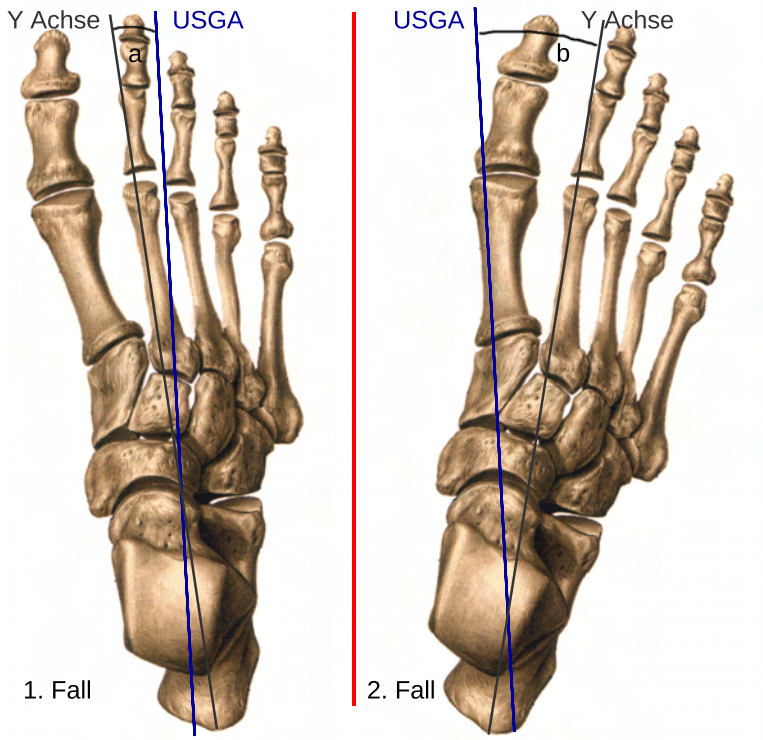

Abbildung 66: Deviation relativ zur Trittspur. Die schwarze Gerade entspricht der Projektion der Y Achse, die blaue Gerade der pojizierten USGA auf die Transversalebene. Im ersten Fall wurde ein negativer Deviationswinkel bestimmt, im zweiten Fall ein positiver. Trotzdem ist die Deviation beider Achsen relativ zur Körpersagitalebene, dargestellt durch die rote Linie, deckungsgleich.

6 Zusammenfassung

Zielstellung der vorliegenden Arbeit war, ein Instrumentarium zu entwickeln, mit dem es möglich ist, die Achsen des Sprunggelenks an größeren Fallzahlen zu ermitteln. Insbesondere mussten die in Kapitel 3.1 formulierten Anforderungen an das Verfahren erfüllt werden:

- ◇ Mobil und damit im Feld einsetzbar,
- ◇ automatischer Dateneinzug und Datenverarbeitung,
- ◇ Echtzeit-Berechnung und grafische Aufbereitung der Ergebnisse,
- ◇ geringer Zeitaufwand pro Messung um große Fallzahlen zu ermöglichen,
- ◇ größtmögliche Objektivität durch benutzerfreundliche GUI und geführten Messablauf.

Mit dem ersten im Feld einsetzbaren Prototypen des Messverfahrens wurden bereits all diese geforderten Eigenschaften erfolgreich umgesetzt. Die Ergebnisse aus den Validierungen und Reliabilitätstest am mechanischen Modell lieferten mit Fehler für die Winkel $\Delta E < 1°$ hervorragende Ergebnisse. Nach ersten Versuchen in vivo wurden einige Berechnungsroutinen aber auch der Messablauf erneut modifiziert:

- ◇ Die Ultraschalldaten werden durch einen Lowpassfilter geglättet,
- ◇ die Berechnungsroutinen wurde verändert, statt instantaner werden die Parameter finiter Rotationsachsen berechnet,
- ◇ der Messablauf wurde ebenfalls modifiziert, die Inversions- und Eversionsbewegung wird nun mehrmals durchgeführt,
- ◇ durch die Veränderung des Messablaufs musste auch die Routine zum Auffinden geeigneter Markerpositionen zur Berechnung der Rotationsachsen modifiziert werden.

Mit diesen Verbesserungen ist es möglich, Sprungelenkachsen in vivo mit einem Fehler für die Winkel kleiner $\Delta E_{Winkel} < 2°$ bei einem geübten Probanden zu bestimmen. Die Hauptfehlerquelle stellt nach wie vor die Bewegungsausführung um die USGA dar. Um diesen Einfluss zu minimieren, werden Mittelwert und Standardabweichung der Parameter der bestimmten Sprunggelenkachsen aus den Einzelbewegungen berechnet und ausgegeben. So kann nicht nur der Einfluss einer unsauber

ausgeführten Einzelbewegung reduziert werden, gleichzeitig können auch anhand des Betrags der Standardabweichung Aussagen über Güte bzw. Genauigkeit der Messung gemacht werden. Da davon ausgegangen werden kann, dass der Messfehler normalverteilt ist, wird die reelle gesuchte Größe mit Zunahme der Messwiederholungen durch den berechneten Mittelwert schärfer abgebildet. Der entscheidende Vorteil gegenüber anderen bereits eingesetzten nicht invasiven in vivo Messmethoden (vergleiche [Alt 2001, van den Bogert et al. 1994]) besteht in der automatischen Datenerfassung und Datenverarbeitung. Dadurch ist es möglich, die Lage der USGA auch mehrmals zu bestimmen und Aussagen über die Messgenauigkeit bzw. die Fehler der erfassten Parameter zu machen.
In ersten Feldeinsätzen wurden die Sprunggelenkachsen von insgesamt $n = 97$ Probanden bestimmt. Die Deviationswinkel der USGA waren mit durchschnittlich $\alpha_{Dev} = 3,4°$ kleiner als bei in vitro Untersuchungen, aber nur minimal kleiner als der von **Alt** gemessene mittlere Deviationswinkel von $\alpha_{Dev} = 7,8°$ [Alt 2001]. Die Standardabweichung der Deviation bei $n = 97$ Probanden betrug $\sigma_{Dev} = 11,4°$. Damit scheint eine starke interindividuelle Varianz der Lage der USGA betreffend bestätigt zu sein, wenngleich die biologische Zweckmäßigkeit dieser Unterschiede noch geklärt werden muss.
Erstmals ist es möglich hinreichend genau in vivo und nicht invasiv die Achsen des Sprunggelenks zu bestimmen. Mit einer durchschnittlichen Messdauer von weniger als 10 Minuten bietet das System die Möglichkeit, auch eine große Anzahl an Probanden zu vermessen. Die Frage, ob die Lage der Achsen des Sprunggelenkes einen Zusammenhang zu akuten Sprunggelenkverletzungen oder chronischen Überlastungsschäden erkennen lassen [Jones 1945, McClay und Bray 1996, Hintermann 1998], sollte sich mit dieser Methode beantworten lassen.

6.1 Ausblick

Die Anzahl der Probanden war mit $n = 97$ zu gering, um statistische Aussagen über den Zusammenhang von Achslagen des unteren oder oberen Sprunggelenks und chronische Überlastungsschäden bzw. akute Verletzungen des Sprunggelenks machen zu können. Es sollten Studien anschließen, die unter Verwendung dieser Methode eine größere Stichprobe untersuchen. Dabei sollten diese nachfolgenden Untersuchungen unbedingt Parameter der Beingeometrie wie z. B. die Trittspur zusätzlich zu den Parametern der Sprunggelenkachsen erfassen. Möglicherweise lässt sich eine Kombination aus diesen erfassten Größen als verletzungsbegünstigend identifizieren. Desweiteren ist es unbedingt notwendig zwischen lateralen und medialen Verletzungen des Sprunggelenkkomplexes zu unterscheiden, da anzunehmen ist, dass den verschiedenen Verletzungsmechanismen auch unterschiedliche mechanische Gegebenheiten zugrunde liegen.

Eine Modellierung des Sprunggelenkkomplexes könnte zum besseren Verständnis der Vorgänge im Sprunggelenk und damit zur Theoriebildung beitragen und erscheint deswegen notwendig. Mit einem solchen Modell könnten für unterschiedliche Sprunggelenkachslagen die Verletzungsmechanismen bzw. die Mechanismen, die zu chronischen Überlastungsschäden führen, beleuchtet werden. Die mithilfe dieses Verfahrens erhaltenen in vivo Daten des Sprunggelenks, könnten bei der Entwicklung eines solchen Modells ebenfalls hilfreich sein.
Sollten sich bestimmte Sprunggelenk-Achslagen als verletzungsbegünstigend identifieren lassen, könnten Risikogruppen frühzeitig erkannt und damit prophylaktische Maßnahmen zur Vermeidung von Sprunggelenkverletzungen ergriffen werden. Die Unterschiede zwischen in vitro und in vivo Untersuchungen, die Deviation des USG betreffend lassen vermuten, dass die Lage der USGA nicht allein durch die artikulierenden Gelenkflächen von Talus und Calcaneus bestimmt werden, sondern auch durch den anliegenden Kapselbandapparat. Das wirft die Frage auf, ob die Lage der USGA durch Training und damit verbundene Haltungsänderungen verändert werden kann.
Zur Beantwortung dieser Fragen ist eine weitere Forschung auf diesem Gebiet unbedingt notwendig.

Literatur

[Alt et al. 1998] Alt, W., A. Gollhofer, H. A. C. Jacob, H. Lohrer und B. Rappe (1998). *In vitro and in vivo determination of ankle joint and subtalar joint axes using the helical axis method.*. ISBS Proceedings, Konstanz, S. 288–291.

[Alt 2001] Alt, W. W. (2001). *Biomechanische Aspekte der Gelenkstabilisierung*. Maurer.

[Arndt et al. 2004] Arndt, Anton, P. Westblad, I. Winson, T. Hashimoto und A. Lundberg (2004). *Ankle and subtalar kinematics measured with intracortical pins during the stance phase of walking.*. Foot Ankle Int, 25(5):357–364.

[Barnett und Napier 1952] Barnett, C. H. und J. R. Napier (1952). *The axis of rotation at the ankle joint in man; its influence upon the form of the talus and the mobility of the fibula.*. J Anat, 86(1):1–9.

[Bartsch 1988] Bartsch, H. J. (1988). *Taschenbuch Mathematischer Fromeln*. Verlag Harri Deutsch Thun.

[Baumhauer et al. 1995] Baumhauer, J. F., D. M. Alosa, A. F. Renström, S. Trevino und B. Beynnon (1995). *A prospective study of ankle injury risk factors.*. Am J Sports Med, 23(5):564–570.

[Beggs 1966] Beggs, J. S. (1966). *Advanced Mechanism*. The Macmillan Company.

[Beynnon et al. 2001] Beynnon, B. D., P. A. Renström, D. M. Alosa, J. F. Baumhauer und P. M. Vacek (2001). *Ankle ligament injury risk factors: a prospective study of college athletes.*. J Orthop Res, 19(2):213–220.

[Boehm 1988] Boehm, B. W. (1988). *A Spiral Model of Software Development and Enhancement*. Computer No. 5, 21:61–72.

[van den Bogert et al. 1994] Bogert, A. J. van den, G. D. Smith und B. M. Nigg (1994). *In vivo determination of the anatomical axes of the ankle joint complex: an optimization approach.*. J Biomech, 27(12):1477–1488.

[Bronstein und Semendjajew 1997] Bronstein, I. N. und K. A. Semendjajew (1997). *Taschenbuch der Mathemaik*. Verlag Harri Deustch.

[Dalheimer 1998] Dalheimer, M. K. (1998). *LATEX kurz & gut*. O'Reilly.

[Debrunner und Jacob 1998] Debrunner, H. U. und H. A. C. Jacob (1998). *Biomechanik des Fußes*. F. Enke Stuttgart.

[Demarais et al. 2002] Demarais, Denise M, R. A. Bachschmidt und G. F. Harris (2002). *The instantaneous axis of rotation (IAOR) of the foot and ankle: a self-determining system with implications for rehabilitation medicine application.*. IEEE Trans Neural Syst Rehabil Eng, 10(4):232–238.

[Denegar et al. 2002] Denegar, Craig R, J. Hertel und J. Fonseca (2002). *The effect of lateral ankle sprain on dorsiflexion range of motion, posterior talar glide, and joint laxity.*. J Orthop Sports Phys Ther, 32(4):166–173.

[Dougherty 2002] Dougherty, J. (2002). *XMath*. Internet, http://www.codeproject.com.

[Downing et al. 1978] Downing, J. W., S. J. Klein und J. C. D'Amico (1978). *The axis of motion of the rearfoot complex.*. J Am Podiatry Assoc, 68(7):484–499.

[Elftman und Manter 1935] Elftman, H. und J. Manter (1935). *The evolution of the human foot with especial reference to the joints*. J Anat Lond, 70:56–57.

[Engsberg 1992] Engsberg, J. R. (1992). *A Method to Determine the Range of Motion of the Ankle Joint Complex, In Vivo*. J. Biomechanics, 26:69–76.

[ESAOTE 2004] ESAOTE, Biomedica Deutschland GmbH (2004). *G-Scan Dedicated MRI*. Anleitung, MRI Sportklinik Canstatt.

[Fick 1911] Fick, R. (1911). *Handbuch der Anatomie und Mechanik der Gelenke*. Verlag Gustav Fischer.

[Gawehn 1993] Gawehn, W. (1993). *Vektor- und Matrizenalgebra für Maschinenbauer*. Mannheim, Leibzig, Wien, Zürich, BI-Wiss.-Verl.

[Gross et al. 1994] Gross, M. T., A. M. Batten, A. L. Lamm, J. L. Lorren, J. J. Stevens, J. M. Davis und G. B. Wilkerson (1994). *Comparison of DonJoy ankle ligament protector and subtalar sling ankle taping in restricting foot and ankle motion before and after exercise.*. J Orthop Sports Phys Ther, 19(1):33–41.

[Grossmann 1991] Grossmann, S. (1991). *Mathematischer Einführungskurs für die Physik*. Teubner Stuttgart.

[Günther 2005] Günther, K. (2005). *LATEX ge-packt*. mitp-Verlag, Bonn.

[Hainzl 1974] Hainzl, J. (1974). *Mathematik für Naturwissenschaftler*. Teubner Stuttgart.

[Hartrumpf 1991] Hartrumpf, M. (1991). *Verfahren zur Lagebestimmung eines bewegten Objektes nach dem Triangulationsverfahren*. Technischer Bericht Fraunhofer Gesellschaft.

[Henke und Gläser 2001] Henke, T. und H. Gläser (2001). *Sportunfälle - Häufigkeit, Kosten, Präfention*. ARAG Allgemeine Sportversicherungen.

[Hertel 2000] Hertel, J. (2000). *Functional instability following lateral ankle sprain.*. Sports Med, 29(5):361–371.

[Hertel 2002] Hertel, Jay (2002). *Functional Anatomy, Pathomechanics, and Pathophysiology of Lateral Ankle Instability.*. J Athl Train, 37(4):364–375.

[Hicks 1953] Hicks, J. H. (1953). *The mechanics of the foot. I. The joints.*. J Anat, 87(4):345–357.

[Hintermann 1998] Hintermann, B. (1998). *Biomechanik der Sprunggelenke–Unfalllmechanismen.*. Swiss Surg., 4:63–69.

[Hintermann und Holzach 1992] Hintermann, B. und P. Holzach (1992). *[Sub-Achilles bursitis–a biomechanical analysis and clinical study]*. Z Orthop Ihre Grenzgeb, 130(2):114–119.

[Hintermann et al. 1994] Hintermann, B., B. M. Nigg und C. Sommer (1994). *Foot movement and tendon excursion: an in vitro study.*. Foot Ankle Int, 15(7):386–395.

[Hochwald et al. 2004] Hochwald, H., W. Alt und G. Busch (2004). *Ultrasonic Pulse-Echo based in Vivo Real-time Determination of the Ankle and Subtalar Joint Axis*. In: *9th Annual Congress, European College of Sport Science*.

[Inman 1976] Inman, T. V. (1976). *The Joints of the Ankle*.

[Isman et al. 1969] Isman, R. E., V. T. V. T. Inman und P. M. Poor (1969). *Anthropometric Studies of the Human Foot and Ankle*. Bulletin of Prosthetics Research, 11:97–108.

[Jacob 1989] Jacob, H. A. C. (1989). *Biomechanics of the forefoot*. Doktorarbeit, Bioengineering Unit University of Strathclyde.

[Jones 1945] Jones, R. L. (1945). *The Functional Significance of the Declination of the Axis of the Subtalar Joint.*. Anat Rec, 93:151–159.

[Kannus et al. 2002] Kannus, P., M. Paavola, T. Paakkala, J. Parkkari, T. Järvinen und M. Järvinen (2002). *[Pathophysiology of overuse tendon injury]*. Radiologe, 42(10):766–770.

[Kuttruff 1988] Kuttruff, H. (1988). *Physik und Techik des Ultraschalls*. Hirzel Verlag.

[Kutzner 1983] Kutzner, J. (1983). *Grundlagen der Ultraschallphysik*. Teubner Stuttgart.

[Kuypers 1989] Kuypers, F. (1989). *Klassische Mechanik*. VCH.

[van Langelaan 1983] Langelaan, E. J. van (1983). *A kinematical analysis of the tarsal joints. An X-ray photogrammetric study.*. Acta Orthop Scand Suppl, 204:1–269.

[Lassiter et al. 1989] Lassiter, T. E., T. R. Malone und W. E. Garrett (1989). *Injury to the lateral ligaments of the ankle.*. Orthop Clin North Am, 20(4):629–640.

[Louwerens et al. 1995] Louwerens, J. W., A. Z. Ginai, B. van Linge und C. J. Snijders (1995). *Stress radiography of the talocrural and subtalar joints.*. Foot Ankle Int, 16(3):148–155.

[Lundberg 1989] Lundberg, A. (1989). *Kinematics of the ankle and foot. In vivo roentgen stereophotogrammetry.*. Acta Orthop Scand Suppl, 233:1–24.

[Läuger 1992] Läuger, K. (1992). *Mathematik kompakt*. Oldenburg Verlag GmbH.

[Manter 1941] Manter, J. T. (1941). *Movements of the subtalar and transversal talar joints*. Anat Rec., 80:397–409.

[Mayer und Dickhut 2002] Mayer, F. und H. H. Dickhut (2002). *Chronische Achillessehnenbeschwerden im Sport*. Deutsche Zeitschrift für Sportmedizin.

[McClay und Bray 1996] McClay, I. und J. Bray (1996). *The subtalar angle: a proposed measure of rearfoot structure.*. Foot Ankle Int, 17(8):499–502.

[Merziger und Wirth 1993] Merziger, G. und T. Wirth (1993). *Repetitorium der höheren Mathematik*. Binomiverlag, Springe Verlag.

[Microsoft 2005] Microsoft (2005). *MSDN Home Page*. http://msdn.microsoft.com.

[Millner 1987] Millner, R. (1987). *Ultraschalltechnik: Grundlagen u. Anwendung*. VEB Fachbuchverlag Leibzig.

[Mönicke 1994] Mönicke, H. J. (1994). *Zur Messung und Analyse kinematischer Prozesse mittels terrestrischer Meßverfahren*. In: *XX International Congress, Melbourne*.

[Olmsted et al. 2004] Olmsted, Lauren C., L. I. Vela, C. R. Denegar und J. Hertel (2004). *Prophylactic Ankle Taping and Bracing: A Numbers-Needed-to-Treat and Cost-Benefit Analysis.*. J Athl Train, 39(1):95–100.

[Paus 1995] Paus, H. J. (1995). *Physik in Experimenten und Beispielen*. Carl Hanser Verlag.

[Peterson und Renström 2002] Peterson, L. und P. Renström (2002). *Verletzungen im Sport*. Deutscher Ärtze-Verlag.

[Phillips und Lidtke 1992] Phillips, R. D. und R. H. Lidtke (1992). *Clinical determination of the linear equation for the subtalar joint axis.*. J Am Podiatr Med Assoc, 82(1):1–20.

[Root et al. 1966] Root, M. L., J. H. Weed und T. E. Sgoriat (1966). *Axis of Motion of the Subtalar Joint, an Anatomical Study*. J Vestibular Res, 56:149–155.

[Royce 1970] Royce, W. W. (1970). *Managing the Development of Large Software Systems: Concepts and Techniques*. WESCON.

[Sarrafian 1993] Sarrafian, S. K. (1993). *Biomechanics of the subtalar joint complex.*. Clin Orthop Relat Res, (290):17–26.

[Segesser und Nigg 1993] Segesser, B. und B. M. Nigg (1993). *Orthopädische und biomechanische Konzepte Sportschuhbau*. Sportverletzung und Sportschaden, 7:150–162.

[Siegler et al. 1988] Siegler, S., J. Chen und C. D. Schneck (1988). *The three-dimensional kinematics and flexibility characteristics of the human ankle and subtalar joints–Part I: Kinematics.*. J Biomech Eng, 110(4):364–373.

[Siegler et al. 2005] Siegler, S., J. K. Udupa, S. I. Ringleb, C. W. Imhauser, B. E. Hirsch, D. Odhner, P. K. Saha, E. Okereke und N. Roach (2005). *Mechanics of the ankle and subtalar joints revealed through a 3D quasi-static stress MRI technique.*. J Biomech, 38(3):567–578.

[Sobota 2000] Sobota (2000). *Atlas der Anatomie des Menschen*. Putz R., Pabst R.

[Sphar 1999] Sphar, C. (1999). *Visual C++ 6 Schritt für Schritt*. Microsoft Press.

[Stacoff et al. 2000] Stacoff, A., B. M. Nigg, C. Reinschmidt, A. J. van den Bogert, A. Lundberg, E. Stüssi und J. Denoth (2000). *Movement coupling at the ankle during the stance phase of running.*. Foot Ankle Int, 21(3):232–239.

[Sutilov 1984] Sutilov, V. A. (1984). *Physik des Ultraschalls*. Springer Verlag.

[Teofilov 2003] Teofilov, N. (2003). *NTGraph3D ActiveX Controll v2.1*. http://www.codeproject.com. An ATL/STL ActiveX control based on OpenGL library for 3D data visualization.

[Thalip 2000] Thalip, F. (2000). *Ultraschall-Füllstandsmessung nach dem Impulslaufzeit-Meßprinzip mit verbesserter Echoimpulserkennung*. Shaker Verlag.

[Tibesku und Pässler 2005] Tibesku, C. O. und H. H. Pässler (2005). *[Jumper's knee–a review]*. Sportverletz Sportschaden, 19(2):63–71.

[Wagner 2006] Wagner, C. (2006). *Zur Analyse der Sprunggelenkachsen in vivo*. Diplomarbeit, Institut für Sportwissenschaft der Universität Stuttgart.

[Westblad et al. 2002] Westblad, Pär, T. Hashimoto, I. Winson, A. Lundberg und A. Arndt (2002). *Differences in ankle-joint complex motion during the stance phase of walking as measured by superficial and bone-anchored markers.*. Foot Ankle Int, 23(9):856–863.

[Wilder und Sethi 2004] Wilder, Robert P und S. Sethi (2004). *Overuse injuries: tendinopathies, stress fractures, compartment syndrome, and shin splints.*. Clin Sports Med, 23(1):55–81, vi.

[Wright et al. 2000] Wright, I. C., R. R. Neptune, A. J. van den Bogert und B. M. Nigg (2000). *The influence of foot positioning on ankle sprains.*. J Biomech, 33(5):513–519.

[Xorx 2005] Xorx, User (2005). *Triangulation (Messtechnik)*. http://de.wikipedia.org. http://www.wikipedia.de.

[Ying et al. 2004] Ying, Ning, W. Kim, Y. Wong und B. H. Kam (2004). *Analysis of passive motion characteristics of the ankle joint complex using dual Euler angle parameters.*. Clin Biomech (Bristol, Avon), 19(2):153–160.

[Zebris Medizintechnik 1999] Zebris Medizintechnik, GmbH (1999). *WinData 2.19 für Windows, Programm zur Meßdatenerfassung*.

[Zebris Medizintechnik 2000] Zebris Medizintechnik, GmbH (2000). *WinJaw 10.x für Windows*.

[Zebris Medizintechnik 2004] Zebris Medizintechnik, GmbH (2004). *ZEBSDK*.

[Ziegler 1996] Ziegler, C. (1996). *Entwicklung und Erprobung eines Positionierungssystems für den lokalen Anwendungsbereich*. Verlag der Bayerischen Akademie der Wissenschaft.

A Prozessablauf des Messdurchgangs

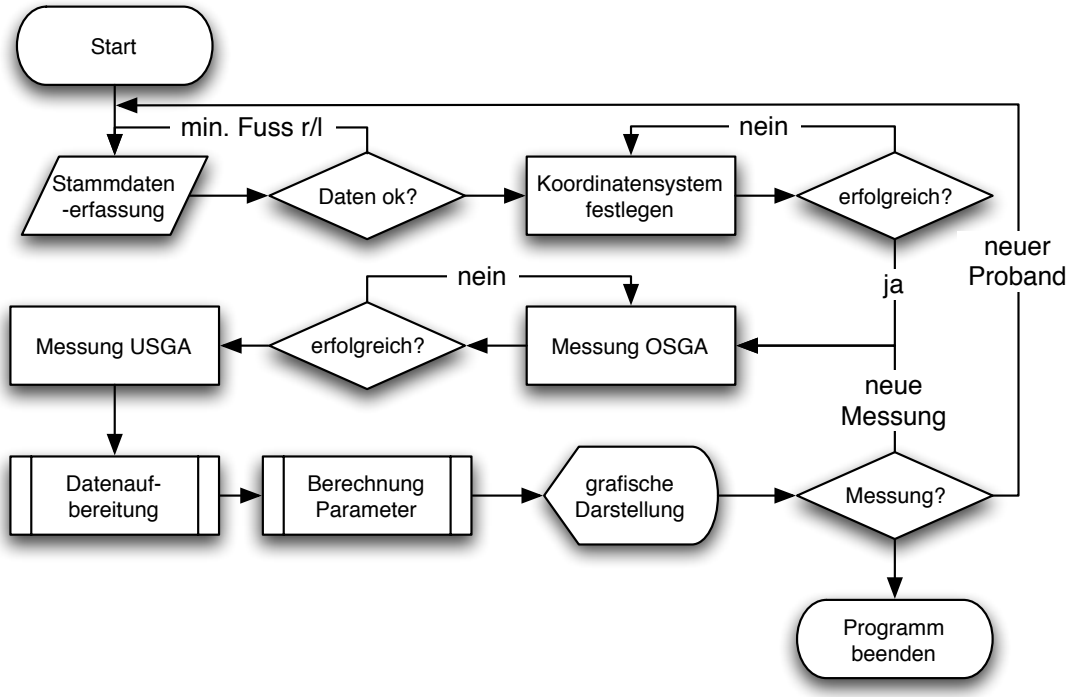

Abbildung 67: Prozessablauf der Messmethode.

B Satz vom Umfangswinkel

Ist k ein Kreis mit dem Mittelpunkt M, und sind P_1 und P_2 zwei Punkte auf dem Kreis k, dann heißt der Winkel P_1MP_2 der *Mittelpunktswinkel* der *Sehne* P_1P_2 im Kreis k (siehe Abbildung 68 auf Seite 124). Die folgende Verallgemeinerung gilt für beliebige Dreiecke und heißt Umfangswinkelsatz oder Peripheriewinkelsatz:

Sind P_1, P_2 und P_3 Punkte auf einem Kreis k mit dem Mittelpunkt M, und ist der Winkel $P_1P_2P_3$ spitz, so ist der Winkel P_1MP_2 doppelt so groß wie der Winkel $P_1P_3P_2$.

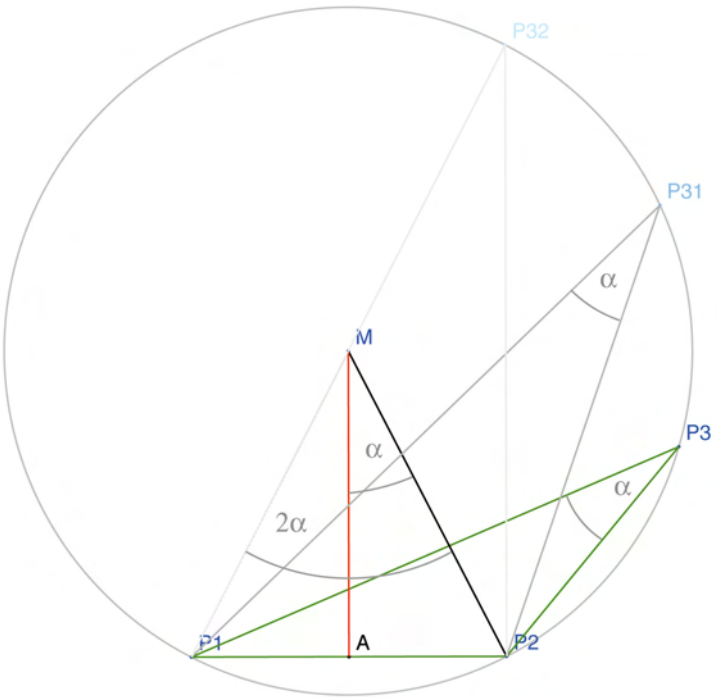

Abbildung 68: Der Winkel P_1MP_2 ist immer doppelt so groß wie der Winkel $P_1P_3P_2$. Werden die Punkte P_1 und P_2 nicht bewegt, verändert sich der Winkel α ($P_1P_3P_2$) nicht. In der Abbildung wurde der Punkt P_3 in den Punkt P_{31} und P_{32} verschoben, der Winkel α blieb dabei erhalten.

C C++ Klasse CalcAxis

```
1  // calcAxis.cpp: Implementierung der Klasse calcAxis.
2  //
3  //////////////////////////////////////////////////////////////////////
4
5  #include "stdafx.h"
6  #include "calcAxis.h"
7  #include <iostream.h>
8  #include <math.h>
9  #include "XMath.h"
10
11 #define PI            3.14159265358979323846264338327 95f
12
13 //////////////////////////////////////////////////////////////////////
14 // Konstruktion/Destruktion
15 //////////////////////////////////////////////////////////////////////
16
17 calcAxis::calcAxis() //Klasse inizieren
18 {
19   //Setz die Bool Variable auf false, wird bei Angabe von Tibia Koordinatensys. auf true gesetzt
20   XYZ = false;
21
22 }
23
24 //Berechnung des Temperatur Faktors
25 calcAxis::calcTemp(double m1x1, double m1y1, double m1z1, double m2x1, double m2y1, double m2z1)
26 {
27   UX_VECTOR3 MM(m1x1-m2x1, m1y1-m2y1, m1z1-m2z1); //Vektor zw. M1 u. M2 des Taststiftes
28   double MML = MM.Magnitude();        //gemessene Laenge des Vektors
29   tempFact = 50/ MML;          //TemperaturFaktor
30 }
31
32 // Berechnung von Punkten aus 2 Zebris Markern des Taststiftes (Dorn vorn --PF--)
33 calcAxis::calcPeakFront(double x1, double y1, double z1,
34         double x2, double y2, double z2, UX_VECTOR3 *PF)
35 {
36   UX_VECTOR3 M(x1-x2,y1-y2,z1-z2); //Richtungsvektor M2M1
37   M.Normalize();  //Normiert den richtungsvektor
38   UX_VECTOR3 AP(x2,y2,z2);
39   *PF = V3.Add(AP,V31.Multiply(M,85));
40
41 }
42
43 //Berechnung Taststift (Dorn hinten --PB--)
44 calcAxis::calcPeakBack(double x1, double y1, double z1,
45            double x2, double y2, double z2, UX_VECTOR3 *PB)
46 {
47   UX_VECTOR3 M(x2-x1,y2-y1,z2-z1); //Richtungsvektor M1M2
48   M.Normalize();  //Normiert den richtungsvektor
49   UX_VECTOR3 AP(x1,y1,z1);
50   *PB = V3.Add(AP,V31.Multiply(M,205));
51 }
52
53
54 //Berechnung der OSGA mit Malleolen Punkte
55 calcAxis::meshOSG(double m1p1x, double m1p1y, double m1p1z,
56       double m2p1x, double m2p1y, double m2p1z,
57       double m1p2x, double m1p2y, double m1p2z,
58       double m2p2x, double m2p2y, double m2p2z)
59 {
60   m1p1x = m1p1x * tempFact;
61   m1p1y = m1p1y * tempFact;
62   m1p1z = m1p1z * tempFact;
63   m2p1x = m2p1x * tempFact;
64   m2p1y = m2p1y * tempFact;
65   m2p1z = m2p1z * tempFact;
66
67   m1p2x = m1p2x * tempFact;
68   m1p2y = m1p2y * tempFact;
69   m1p2z = m1p2z * tempFact;
70   m2p2x = m2p2x * tempFact;
71   m2p2y = m2p2y * tempFact;
72   m2p2z = m2p2z * tempFact;
73
74
75   calcPeakFront(m1p1x, m1p1y, m1p1z, m2p1x, m2p1y, m2p1z, &ML);//Erstellt den Vektor auf ML
76
77   calcPeakFront(m1p2x, m1p2y, m1p2z, m2p2x, m2p2y, m2p2z, &MM);//Erstellt Vektor auf MM
78
79   if (XYZ) {
80     //Vektoren werden transformiert
81     transform(&ML);
```

```
82      transform(&MM);
83    }
84    mOSG_RV = V3.Subtract(ML,MM);    //Berechnen des Richtungsvektors
85    dMM_Distance = mOSG_RV.Magnitude(); //Berechne Abstand der Malleolenpunkte
86    mOSG_RV.Normalize();        //Normieren des Richtungsvektors
87    mOSG_AP = ML;          //Zuweisen Aufpunkt der OSGA, Malleolus lateralis
88
89    //Berechnung des Deviationswinkels
90    calcDevOsg(&mOSG_RV,&mOSG_Dev);
91
92    //Berechnung des Inclinationswinkels
93    calcIncOsg(&mOSG_RV, &mOSG_Inc);
94
95
96    //Berechnung Abstand OSG zu Kraftvektor Achilles
97    calcDistance(&Otrans, &WWN, &mOSG_AP, &mOSG_RV, &lever_OSG_MM);
98
99  }
100
101 ////////////////////////////////////////////////////////////////////////////////////////////
102 //Methode zur Übergabe der durch Zebris eingelesenen Markerpunkte des
103 //Tibia-Koordinatensystems und dessen Bestimmung
104 ////////////////////////////////////////////////////////////////////////////////////////////
105 calcAxis::coordTibia(double m1p1x, double m1p1y, double m1p1z,
106       double m2p1x, double m2p1y, double m2p1z,
107       double m1p2x, double m1p2y, double m1p2z,
108       double m2p2x, double m2p2y, double m2p2z,
109       double m1p3x, double m1p3y, double m1p3z,
110       double m2p3x, double m2p3y, double m2p3z,
111       double m1p4x, double m1p4y, double m1p4z,
112       double m2p4x, double m2p4y, double m2p4z)
113 {
114   //Variablen definieren
115   float HU,ab12,ab23,ab13;
116
117   //Berechnen des Faktors zur Schallausbreitungsgeschwindigkeit (TemperaturabhLngig)
118   calcTemp(m1p1x, m1p1y, m1p1z, m2p1x, m2p1y, m2p1z);
119
120
121   //Berechne aus 2 Zeigerpunkte -> Punkte des Tibiakoordinatensystems
122   //Punkt 1 auf der Tibia
123   UX_VECTOR3 MK1;
124   calcPeakFront(m1p1x, m1p1y, m1p1z, m2p1x, m2p1y, m2p1z, &MK1);
125   //Punkt 2 auf der Tibia
126   UX_VECTOR3 MK2;
127   calcPeakFront(m1p2x, m1p2y, m1p2z, m2p2x, m2p2y, m2p2z, &MK2);
128   //Punkt auf der Achillessehne
129   UX_VECTOR3 MK3;
130   calcPeakBack(m1p3x, m1p3y, m1p3z, m2p3x, m2p3y, m2p3z, &MK3);
131   //Punkt auf der Standfläche
132   UX_VECTOR3 MK4;
133   calcPeakFront(m1p4x, m1p4y, m1p4z, m2p4x, m2p4y, m2p4z, &MK4);
134
135   ////////////////////////////////////////////////////////////////////////
136   //Erzeugen eines Kartesischen Koordinatensystems mit normierten Achsen U-V-W
137   ////////////////////////////////////////////////////////////////////////
138
139   ab12=V3.Distance(MK1,MK2);
140   ab23=V3.Distance(MK2,MK3);
141   ab13=V3.Distance(MK1,MK3);
142   //MK1 wird dem Aufpunkt zugewiesen (f§r Berechnung Abstand USG zur TibialLngsachse)
143   Tibia_AP = MK1;
144
145   //Berechnung des Höhenunterschiedes zwischen MK2 und MK3 bezüblich U-V-W Koord.
146   HU=(ab13*ab13-ab12*ab12-ab23*ab23)/(2*ab12);//entspricht "e" in meiner Zeichnung
147
148   //Bestimmung der W-Achse (Normiert) (Tibialängsachse)
149   WWN=V3.Subtract(MK1,MK2);
150   WWN.Normalize();
151
152   //Bestimmung des Schnittpunktes zwischen W-Achse und der
153   //Orthogonalen zur W-Achse durch den Punkt MK3
154   UX_VECTOR3 HUW;
155   HUW=V3.Subtract(MK2,V31.Multiply(HU,WWN));
156
157   //Bestimmung der V-Achse (Normiert)
158   //(Achse Schnitt Sagital mit Transversal, Achse nach vorne)
159   VVN=V3.Subtract(HUW,MK3);
160   VVN.Normalize();
161
162   //Bestimmung der U-Achse (Normiert) (Achse durch Maleolen)
163   UUN=V3.CrossProduct(VVN,WWN);
164   UUN.Normalize();
165
```

```
166    //Berechnung Vektor Bodenpunkt - Achillesmarker
167    UX_VECTOR3 BA;
168    BA=V3.Subtract(MK3,MK4);
169
170    //Winkel BA UUN(Tibia-Längsachsenrichtung)
171    double Winkel_BA;
172    Winkel_BA=V3.AngleBetweenVectors(BA,WWN);
173    //Länge des Vektors BA
174    double l_BA;
175    l_BA = BA.Magnitude();
176    //Abstand Boden Achillesmarker
177    double l_AO;
178    //Achtung!!! trigonometrische Funktionen brauchen Winkel im BOGENMASS!!!
179    l_AO=l_BA*cos(Winkel_BA*PI/180);
180    //Berechnung des Koordinatenursprungs O
181    O=V3.Add(V31.Multiply(l_AO,-WWN),MK3);
182    //wurde ein Tibia Koordinatensystem berechnet XYZ = 1
183    XYZ = true;
184    //Transformierter Koordinatenursprung
185    Otrans.X = 0;
186    Otrans.Y = 0;
187    Otrans.Z = 0;
188  }
189
190
191  ////////////////////////////////////////////////////////////////////////////
192  //Methode für Koordinatentransformation. Transformiert wird immer genau ein Punkt
193  ////////////////////////////////////////////////////////////////////////////
194  calcAxis::transform(UX_VECTOR3 *vect) {
195
196
197
198    ////////////////////////////////////////////////////////////////////
199    //Durchführung der Koordinatentransformation.
200    //Ursprung des Tibiasystems liegt in ref2 bzw. MK2
201    ////////////////////////////////////////////////////////////////////
202    //Definition des ersten Messpunktes, welcher von Zebris ermittelt wurde
203    UX_VECTOR3 point1_alt;
204    point1_alt = *vect;
205    //Initialisierung des um den Vektor MK2 zurückverschobenen Messpunktes
206    UX_VECTOR3 point1_alt1;
207    //Verschiebung des Markerpunktes um O(Ursprung des Tibia-Koordinatensystems)
208    point1_alt1 = V3.Subtract(point1_alt, O);
209    //Definition der Abbildungsmatrix ABB zur Koordinatentransformation
210    UX_MATRIX3 ABB(UUN.X,VVN.X,WWN.X,UUN.Y,VVN.Y,WWN.Y,UUN.Z,VVN.Z,WWN.Z);
211    UX_MATRIX3 ABBI;
212    //Die Inverse der Abbildungsmatrix
213    //(Der Markerpunkt wird nur mit dieser Inversen multipliziert!)
214    ABBI = ABB.Inverse();
215    //Bestimmung des tranformierten Punktes point1_neu
216    *vect = M3.Multiply(ABBI,point1_alt1);
217  }
218
219  ////////////////////////////////////////////////////////////////////////////
220  //Methode für Koordinatentransformation Rechts in Links System
221  ////////////////////////////////////////////////////////////////////////////
222  calcAxis::transform_rl(UX_VECTOR3 *vect) {
223
224    //Definition der Abbidlungsmatrix zur Koordinatentransformation
225    //Da WWN ungefähr -1, aber nur die X(U) Komponente transformiert
226    //werden soll (gespielelt) WWN-> -WWN
227    UX_MATRIX3 ABM(-1,0,0,0,1,0,0,0,1);
228    *vect = M3.Multiply(ABM,*vect);
229  }
230  ////////////////////////////////////////////////////////////////////////
231  //Berechnung von Rotationsachen mit dem Einpunktverfahren
232  ////////////////////////////////////////////////////////////////////////
233  calcAxis::calcEV(UX_VECTOR3 vect_p1, UX_VECTOR3 vect_p2, UX_VECTOR3 vect_p3,
234      UX_VECTOR3 *vect_AP,UX_VECTOR3 *vect_rv,double *angle, int from) {
235
236    //from = 1 -> von calcUSG; from = 0 -> von calcOSG
237    double drehwinkel;
238    //Vektor zum Punkt1 initialisieren
239    UX_VECTOR3 vec_p1;
240    //Vektor zum Punkt2 initialisieren
241    UX_VECTOR3 vec_p2;
242    //Vektor zum Punkt3 initialisieren
243    UX_VECTOR3 vec_p3;
244
245    //Zuweisen der Vektoren
246    vec_p1 = vect_p1;
247    vec_p2 = vect_p2;
248    vec_p3 = vect_p3;
249
```

```
250   //Wurde ein alternativ Koordinatensystem angegeben -> Trafo
251   if (XYZ) {
252   //Vektoren werden transformiert
253   transform(&vec_p1);
254   transform(&vec_p2);
255   transform(&vec_p3);
256   }
257
258   //Für linkes Bein trafo.
259   if (from == 1 && Leg == 'L') {
260     transform_rl(&vec_p1);
261     transform_rl(&vec_p2);
262     transform_rl(&vec_p3);
263   }
264
265
266   //Verbindungsvektor zwischen Punkt1 und Punkt2 -> vec_a
267   UX_VECTOR3 vec_a;    //Vektor inizziieren
268   vec_a = V3.Subtract(vec_p2, vec_p1);
269
270
271   //Verbindungsvektor zwischen Punkt2 und Punkt3 -> vec_b
272   UX_VECTOR3 vec_b;    //Vektor inizziieren
273   vec_b = V3.Subtract(vec_p3, vec_p2);
274
275   //Verbindungsvektor zwischen Punkt1 und Punkt3 -> vec_c
276   UX_VECTOR3 vec_c;    //Vektor inizziieren
277   vec_c = V3.Subtract(vec_p3, vec_p1);
278
279
280   //Berechnung des Richtungsvektors der Achse (Kreuzprodukt von vec_p1p3 x vec_p1p2)
281   UX_VECTOR3 vec_rv;//Vector ini,
282   vec_rv = V3.CrossProduct(vec_a, vec_c);
283
284   //Linkes Bein lateral -> Medial; Rechtssystem -x -> +x
285   //if (Leg == 1 && from == 1)
286   //{
287   //  vec_rv = -vec_rv;
288   //}
289   //Richtungsvektor normieren
290   UX_VECTOR3 vec_rvn; //Vec. ini
291   vec_rv.Normalize();      //Normieren
292   vec_rvn = vec_rv;      //zuweisen
293
294
295
296   //Berechnung des Winkel zwischen Vec. c und b (Satz des Umfangswinkels)
297   double winkel_a;
298   winkel_a = V3.AngleBetweenVectors(vec_c, vec_b);
299
300   //negative Winkel ausschließen
301   if (winkel_a < 0) {
302     winkel_a = -winkel_a;
303   }
304   //für die Dreiecksberechnung müssen die BetrLge der Vektoren ermittelt werden
305   // l_vec_a -> länge vektor a;
306   //l_vec_f -> lLnge Vektor f (Vektor von a/2 nach Kreismittelpunkt)
307   float l_vec_a, l_vec_f; //Variable def.
308   l_vec_a = vec_a.Magnitude(); // Betrag von Vec. a
309   l_vec_a = l_vec_a / 2; //Betrag /2 für Dreiecksberechnung
310
311   //Berechnung des Streckungsfaktor des Vektors vec_f (Mittelsenkrechte auf Vec a)
312   l_vec_f = l_vec_a / tan((winkel_a * PI) / 180); //tan Winkel im Bogenmaß
313
314   //Berechnung der Richtung des Vec. vec_f (Kreuzprodukt von vec_a und Achsrichtung)
315   UX_VECTOR3 vec_f; //Vec. ini.
316   vec_f = V3.CrossProduct(vec_rvn, vec_a);
317   //vec_rvn=-vec_rvn;//Richtungskorrektur für rechtes Bein bzgl Richtungsvektor der Achse
318
319   //Normieren des Vektors vec_f
320   UX_VECTOR3 vec_fn; //Vec. ini.
321   vec_f.Normalize();
322   vec_fn = vec_f; //zuweisen
323
324   //Berechnung des Vectors zum Aufpunkt der Achse (vec_AP)
325   UX_VECTOR3 vec_AP; //Vec. ini.
326
327     //Berechnung des Vektors a/2
328     UX_VECTOR3 vec_a2; //Vec. ini.
329     vec_a2 = V3.Divide(vec_a,2); //Vec. a/2
330     //Berechnung des Vektors von a/2 zum AP
331     UX_VECTOR3 vec_f2; //Vec. ini.
332     //Richtung des normierten Vec. f mal Streckungsfaktor aus der Dreiecksberechnung
333     vec_f2 = V3.Multiply(vec_fn, l_vec_f);
```

```
334     //Hier die Berechnung zum AP (vec. zum Punkt 1 + vec a/2 + vec f2)
335     vec_AP = V3.Add(V31.Add(vec_p1, vec_a2), vec_f2);
336
337   //Berechnung des Drehwinkels
338   //Iniziieren der Vectoren von AP zu P1 und P2
339   UX_VECTOR3 vec_dw1;
340   UX_VECTOR3 vec_dw2;
341
342   vec_dw1 = V3.Subtract(vec_p1, vec_AP); //Punkt1 und AP
343   vec_dw2 = V3.Subtract(vec_p3, vec_AP); //Punk3 und AP
344
345   //Winkel berechnen
346
347   drehwinkel = V3.AngleBetweenVectors(vec_dw1, vec_dw2);
348   //negative Winkel ausschließen
349   if (drehwinkel < 0 ) {
350     drehwinkel = -drehwinkel;
351   }
352
353   //Ausgabe des Drehwinkel (Überschreiben des Speichers)
354   *angle = drehwinkel;
355
356
357   //Für linkes Bein Trafo Aufpunkt
358   if (from == 1 && Leg == 'L') {
359     transform_rl(&vec_AP);
360     transform_rl(&vec_rvn);
361   }
362
363
364   //Ausgabe Ergbniss Aufpunkt (AP) und Richtungsvektor der Achse (RV)
365   *vect_AP = vec_AP;
366   *vect_rv = vec_rvn;
367
368
369
370 }
371
372
373 ////////////////////////////////////////////////////////////////////////////////////////
374 //Berechnung der OSGA
375 ////////////////////////////////////////////////////////////////////////////////////////
376 calcAxis::calcOSG(double p1x, double p1y, double p1z,
377     double p2x, double p2y, double p2z,
378     double p3x, double p3y, double p3z)
379 {
380   double angle, point1x, point1y, point1z,
381     point2x, point2y, point2z,
382     point3x, point3y, point3z;
383
384   //Temperaturanpassung
385   p1x = p1x * tempFact;
386   p1y = p1y * tempFact;
387   p1z = p1z * tempFact;
388
389   p2x = p2x * tempFact;
390   p2y = p2y * tempFact;
391   p2z = p2z * tempFact;
392
393   p3x = p3x * tempFact;
394   p3y = p3y * tempFact;
395   p3z = p3z * tempFact;
396
397   point1x = p1x;//Marker Messung 1
398   point1y = p1y;
399   point1z = p1z;
400
401   point2x = p2x;//Marker Messung 2
402   point2y = p2y;
403   point2z = p2z;
404
405   point3x = p3x;//Marker Messung 3
406   point3y = p3y;
407   point3z = p3z;
408
409   //Initialisieren der Vektoren
410   UX_VECTOR3 vect_p1(p1x,p1y,p1z);
411   UX_VECTOR3 vect_p2(p2x,p2y,p2z);
412   UX_VECTOR3 vect_p3(p3x,p3y,p3z);
413   //Initialisieren der Vektoren für Aufpunkt und Richtungvektor
414   UX_VECTOR3 vect_AP;
415   UX_VECTOR3 vect_rv;
416
417   //Bei OSGA wird "0" übergeben
```

```
418   calcEV(vect_p1, vect_p2, vect_p3, &vect_AP, &vect_rv, &angle, 0);
419
420   //Zuweisen des Ergebnisses an Public variablen
421   //Aufpunkt OSG
422   OSG_APX = vect_AP.X;
423   OSG_APY = vect_AP.Y;
424   OSG_APZ = vect_AP.Z;
425
426   //Definieren der public Variable OSG_AP
427   OSG_AP = vect_AP;
428
429   //Richtungsvektor
430   OSG_RVX = vect_rv.X;
431   OSG_RVY = vect_rv.Y;
432   OSG_RVZ = vect_rv.Z;
433
434   //Definieren der public Variable OSG_RV
435   OSG_RV = vect_rv;
436
437   //Drehwinkel
438   OSG_winkel = angle;
439
440   //Berechnung des Deviationswinkels
441   calcDevOsg(&OSG_RV,&OSG_Dev);
442
443   //Berechnung des Inclinationswinkels
444   calcIncOsg(&OSG_RV, &OSG_Inc);
445
446   //Berechnung Abstand OSGA zu Kraftvektor Achilles
447   calcDistance(&Otrans, &WWN, &OSG_AP, &OSG_RV, &Lever_OSGA);
448
449   //Berechne Punkte für Ausgabe
450   calcPoints(&vect_AP, &vect_rv, &OSGA_P1, &OSGA_P2, 80);
451
452
453 }
454
455
456 ////////////////////////////////////////////////////////////////////////////////////
457 //Berechnung der USGA
458 ////////////////////////////////////////////////////////////////////////////////////
459 calcAxis::calcUSG(double p1x, double p1y, double p1z,
460     double p2x, double p2y, double p2z,
461     double p3x, double p3y, double p3z)
462 {
463   double angle,point1x, point1y, point1z,
464       point2x, point2y, point2z,
465       point3x, point3y, point3z;
466
467   //Temperatur
468   p1x = p1x * tempFact;
469   p1y = p1y * tempFact;
470   p1z = p1z * tempFact;
471
472   p2x = p2x * tempFact;
473   p2y = p2y * tempFact;
474   p2z = p2z * tempFact;
475
476   p3x = p3x * tempFact;
477   p3y = p3y * tempFact;
478   p3z = p3z * tempFact;
479
480   point1x = p1x;//Marker Messung 1
481   point1y = p1y;
482   point1z = p1z;
483
484   point2x = p2x;//Marker Messung 2
485   point2y = p2y;
486   point2z = p2z;
487
488   point3x = p3x;//Marker Messung 3
489   point3y = p3y;
490   point3z = p3z;
491
492   //Initialisieren der Vektoren
493   UX_VECTOR3 vect_p1(p1x,p1y,p1z);
494   UX_VECTOR3 vect_p2(p2x,p2y,p2z);
495   UX_VECTOR3 vect_p3(p3x,p3y,p3z);
496   //Initialisieren der Vektoren für Aufpunkt und Richtungvektor
497   UX_VECTOR3 vect_AP;
498   UX_VECTOR3 vect_rv;
499
500   //Bei USGA wird "1" übergeben
501   calcEV(vect_p1, vect_p2, vect_p3, &vect_AP, &vect_rv, &angle, 1);
```

```
502
503     ////////////////////////////////////////////////////////////
504     //Nicht rotierte USGA
505     //Kopieren der Vektoren
506     UX_VECTOR3 vect_AP2 = vect_AP;
507     UX_VECTOR3 vect_RV2 = vect_rv;
508     //Zuweisen der nicht zurückrotierten Vektoren
509     USG_APX2 = vect_AP.X;
510     USG_APY2 = vect_AP.Y;
511     USG_APZ2 = vect_AP.Z;
512     USG_RVX2 = vect_rv.X;
513     USG_RVY2 = vect_rv.Y;
514     USG_RVZ2 = vect_rv.Z;
515
516     //Berechnung des Deviationswinkels
517     calcDevUsg(&vect_RV2,&USG_dev2);
518
519     //Berechnung des Inclinationswinkels
520     calcIncUsg(&vect_RV2, &USG_inc2);
521
522     //Berechne Punkte für Ausgabe
523     calcPoints(&vect_AP2, &vect_RV2, &USGA_P1_2, &USGA_P2_2, 100);
524     //
525     ////////////////////////////////////////////////////////////
526
527     ////////////////////////////////////////////////////////////
528     // um die MMAchse rotierte USGA
529     //kopieren der Vektoren (AP und RV)
530     UX_VECTOR3 vect_AP1 = vect_AP;
531     UX_VECTOR3 vect_RV1 = vect_rv;
532
533     //rotieren um die OSGA der Malleolen
534     rotBack(&mOSG_AP, &mOSG_RV, &vect_AP1, &vect_RV1);
535     //zuweisen der Vektroeinträge (Malleolen OSGA)
536     USG_APX1 = vect_AP1.X;
537     USG_APY1 = vect_AP1.Y;
538     USG_APZ1 = vect_AP1.Z;
539     USG_RVX1 = vect_RV1.X;
540     USG_RVY1 = vect_RV1.Y;
541     USG_RVZ1 = vect_RV1.Z;
542
543     //Berechnung des Deviationswinkels
544     calcDevUsg(&vect_RV1,&USG_dev1);
545
546     //Berechnung des Inclinationswinkels
547     calcIncUsg(&vect_RV1, &USG_inc1);
548
549     //Berechne Punkte für Ausgabe
550     calcPoints(&vect_AP1, &vect_RV1, &USGA_P1_1, &USGA_P2_1, 100);
551     //
552     ////////////////////////////////////////////////////////////
553
554
555     ////////////////////////////////////////////////////////////
556     //Um die gemessene OSGA rotierte USGA
557     //Zurückdrehen der USGA um die berechnete OSGA
558     rotBack(&OSG_AP, &OSG_RV, &vect_AP, &vect_rv);
559
560     //Zuweisen des Ergebnisses an Public variablen
561     //Aufpunkt OSG
562     USG_APX = vect_AP.X;
563     USG_APY = vect_AP.Y;
564     USG_APZ = vect_AP.Z;
565     //Richtungsvektor
566     USG_RVX = vect_rv.X;
567     USG_RVY = vect_rv.Y;
568     USG_RVZ = vect_rv.Z;
569
570     USG_RV = vect_rv;
571     USG_AP = vect_AP;
572
573     //Drehwinkel
574     USG_winkel = angle;
575
576
577     //Berechnung des Deviationswinkels
578     calcDevUsg(&USG_RV,&USG_Dev);
579
580     //Berechnung des Inclinationswinkels
581     calcIncUsg(&USG_RV, &USG_Inc);
582
583     //Berechnung Winkel zwischen USGA und OSGA
584     calcAngle();
585
```

```
586    //Berechnung des Abstands zwischen OSG und USG
587    calcDistance(&OSG_AP,&OSG_RV,&vect_AP1,&vect_RV1,&Dist);
588
589
590    //Berechnung Abstand USGA zu Kraftvektor Achilles
591    calcDistance(&Otrans, &WWN, &USG_AP, &USG_RV, &Lever_USGA);
592    //Berechnung absatnd USGA1 zu Kraftvektor Achilles (USGA um MM Achse zurückrotiert)
593    calcDistance(&Otrans, &WWN, &vect_AP1, &vect_RV1, &lever_USG_1);
594    //Berechnung absatnd USGA2 zu Kraftvektor Achilles (USGA nicht zurückrotiert)
595    calcDistance(&Otrans, &WWN, &vect_AP2, &vect_RV2, &lever_USG_2);
596    //Berechne Punkte für Ausgabe
597    calcPoints(&vect_AP, &vect_rv, &USGA_P1, &USGA_P2, 100);
598
599    //Konvertieren in String
600    //stringOutput();
601
602
603 }
604
605
606 ////////////////////////////////////////////////////////////////////////////////////////
607 // Methode, bei der Aufpunkt und Richtungsvektor der USGA
608 //um einen definierten Winkel um die OSGA gedreht wird
609 ////////////////////////////////////////////////////////////////////////////////////////
610 calcAxis::rotBack(UX_VECTOR3 *vect_AP_osg, UX_VECTOR3 *vect_rv_osg,
611        UX_VECTOR3 *vect_AP_usg, UX_VECTOR3 *vect_rv_usg) {
612    double rdw;
613    //Initialisieren der Vektoren zum AP und des Richtungsvektors der OSGA
614    UX_VECTOR3 result_AP;
615    UX_VECTOR3 result_RV;
616    //Zuweisen der Vectoren
617    result_AP = *vect_AP_osg;
618
619    //RV bei L Bein negativ (mll-mlm) daher wird rotationmatrix falsch berechnet
620    if (Leg == 'R') //rechtes Bein
621    {
622    result_RV = -(*vect_rv_osg);  //Rechtes bein
623    }
624    if (Leg == 'L') //rechtes Bein
625    {
626    result_RV = (*vect_rv_osg);   //Linkes Bein
627    }
628
629    //Zweisen des Drehwinkels
630    rdw = OSG_winkel*PI/180;
631    //Definition der Rotationsmatrix Rot
632    UX_MATRIX3 Rot;
633    UX_MATRIX3 Rot1(result_RV.X*result_RV.X,result_RV.X*result_RV.Y,result_RV.X*result_RV.Z,
634         result_RV.X*result_RV.Y,result_RV.Y*result_RV.Y,result_RV.Y*result_RV.Z,
635         result_RV.X*result_RV.Z,result_RV.Y*result_RV.Z,result_RV.Z*result_RV.Z);
636    UX_MATRIX3 Rot2(1,0,0,0,1,0,0,0,1);
637    UX_MATRIX3 Rot3(0,-(result_RV.Z),result_RV.Y,
638         result_RV.Z,0,-(result_RV.X),
639         -(result_RV.Y),result_RV.X,0);
640    UX_MATRIX3 Rot11;
641    Rot11 = M3.Multiply((1-cos(rdw)),Rot1);
642    UX_MATRIX3 Rot22;
643    Rot22 = M3.Multiply(cos(rdw),Rot2);
644    UX_MATRIX3 Rot33;
645    Rot33 = M3.Multiply(sin(rdw),Rot3);
646
647    Rot = M3.Add(M31.Add(Rot11,Rot22),Rot33);
648
649    //Beispiel für Rotation von P_rot1 und P_rot2 um OSG Achse um Winkel rdw
650    //Initialisieren des Vektors Aufpunkt USGA
651    UX_VECTOR3 P_rot1;
652    //zuweisen des Vektors
653    P_rot1 = *vect_AP_usg;
654
655    UX_VECTOR3 Pver11;
656    UX_VECTOR3 Pver21;
657    UX_VECTOR3 P_rotiert1;
658    Pver11=V3.Subtract(P_rot1,result_AP);
659    Pver21=M3.Multiply(Rot,Pver11);
660    //P_rotiert1 ist dann der um den Rückdrehwinkel rdw rotierte Punkt P_rot1
661    P_rotiert1=V3.Add(Pver21,result_AP);
662    //Initialisieren des Richtungsvektors USGA
663    UX_VECTOR3 P_rot2;
664    //zuweisen des Richtungsvektors USGA
665    P_rot2 = *vect_rv_usg;
666
667    UX_VECTOR3 Pver12;
668    UX_VECTOR3 Pver22;
669    UX_VECTOR3 P_rotiert2;
```

```
670    Pver12=V3.Subtract(P_rot2,result_RV);
671    Pver22=M3.Multiply(Rot,Pver12);
672    //P_rotiert2 ist dann der um den Rückdrehwinkel rdw rotierte Punkt P_rot2
673    P_rotiert2=V3.Add(Pver22,result_RV);
674
675    //Ausgabe der zurückgedrehten USG Achse
676    //Zuweisen der rotierten USGA
677    *vect_AP_usg = P_rotiert1;
678    *vect_rv_usg = P_rotiert2;
679  }
680
681
682  ////////////////////////////////////////////
683  //Methode dur Berechnung des Deviationswinkels OSGA
684  ////////////////////////////////////////////
685  calcAxis::calcDevOsg(UX_VECTOR3 *vect, double *angle)
686  {
687    UX_VECTOR3 vec;
688    vec = *vect;
689
690    if (Leg == 'L') //linkes Bein
691    {
692      *angle = 90-((atan(vec.Y/vec.X))*180/PI);
693    }
694    if (Leg == 'R') //rechtes Bein
695    {
696      vec.Y = - vec.Y;//Rechtes Bein -> Linkssystem
697      *angle = 90-((atan(vec.Y/vec.X))*180/PI);
698    }
699  }
700
701
702  ////////////////////////////////////////////
703  //Methode dur Berechnung des Inclinationswinkels OSGA
704  ////////////////////////////////////////////
705  calcAxis::calcIncOsg(UX_VECTOR3 *vect, double *angle)
706  {
707    UX_VECTOR3 vec;
708    vec = *vect;
709
710    if (Leg == 'L') //linkes Bein
711    {
712      *angle = 90-((atan(vec.Z/vec.X))*180/PI);
713    }
714    if (Leg == 'R') //rechtes Bein
715    {
716      vec.Z = - vec.Z;//Rechtes Bein -> Linkssystem
717      *angle = 90-((atan(vec.Z/vec.X))*180/PI);
718    }
719
720    //Winkel wird zuerst zur x Achse gemessen (zwischen 0-180) danach 90° abgezogen*/
721  }
722
723  ////////////////////////////////////////////
724  //Methode dur Berechnung des Deviationswinkels USGA
725  ////////////////////////////////////////////
726  calcAxis::calcDevUsg(UX_VECTOR3 *vect, double *angle)
727  {
728
729    UX_VECTOR3 vec;
730    vec = *vect;
731
732    if (Leg == 'R') //rechtes Bein
733    {
734
735      *angle = -(atan (vec.X/vec.Y))*180/PI;
736    }
737
738    if (Leg == 'L') //linkes Bein
739    {
740      *angle = (atan (vec.X/vec.Y))*180/PI;
741    }
742
743  }
744
745  ////////////////////////////////////////////
746  //Methode dur Berechnung des Inclinationswinkels USGA
747  ////////////////////////////////////////////
748  calcAxis::calcIncUsg(UX_VECTOR3 *vect, double *angle)
749  {
750    UX_VECTOR3 vec;
751    vec = *vect;
752    //Stimm f§r beide Beine da Rotationmatrix re/li unterscheidet
753    *angle = (atan (vec.Z/vec.Y))*180/PI;
```

```
754 }
755
756 ////////////////////////////////////////////////////////
757 //Methode zur Berechnung des Winkels zwischen OSG und USG
758 ////////////////////////////////////////////////////////
759 calcAxis::calcAngle()
760 {
761   Angle_OSG_USG = V3.AngleBetweenVectors(OSG_RV,USG_RV);
762   if (Angle_OSG_USG >90)
763   {
764     Angle_OSG_USG = 180-Angle_OSG_USG;
765   }
766 }
767
768
769 ////////////////////////////////////////////////////////
770 //Methode zur Berechnung des Abstands zwischen OSG und USG
771 ////////////////////////////////////////////////////////
772 calcAxis::calcDistance(UX_VECTOR3 *vect_AP_osg,UX_VECTOR3 *vect_RV_osg,
773       UX_VECTOR3 *vect_AP_usg,UX_VECTOR3 *vect_RV_usg, double *Abstand)
774 {
775 UX_VECTOR3 vec_AP_osg;
776 UX_VECTOR3 vec_RV_osg;
777 UX_VECTOR3 vec_AP_usg;
778 UX_VECTOR3 vec_RV_usg;
779
780 vec_AP_osg = *vect_AP_osg;
781 vec_RV_osg = *vect_RV_osg;
782 vec_AP_usg = *vect_AP_usg;
783 vec_RV_usg = *vect_RV_usg;
784
785 UX_VECTOR3 a = V3.Subtract(vec_AP_usg, vec_AP_osg);    //erste Spalte Spatprodukt
786 UX_MATRIX3 Matrix3(a.X, vec_RV_osg.X, vec_RV_usg.X,
787     a.Y, vec_RV_osg.Y, vec_RV_usg.Y,
788     a.Z, vec_RV_osg.Z, vec_RV_usg.Z);
789 double det = Matrix3.Determinate();
790 det = abs(det);
791 UX_VECTOR3 nenner = V3.CrossProduct(vec_RV_osg, vec_RV_usg);
792 double nenner2 = nenner.Magnitude();
793 *Abstand = det/nenner2;
794 }
795
796
797 ////////////////////////////////////////////////////////
798 //Methode zur Berechnung zweier Punkte für die Ausgabe
799 ////////////////////////////////////////////////////////
800 calcAxis::calcPoints(UX_VECTOR3 *vec_AP, UX_VECTOR3 *vec_RV,
801       UX_VECTOR3 *POINT1, UX_VECTOR3 *POINT2, int lenght)
802 {
803   *POINT1 = V3.Add(*vec_AP, V31.Multiply(*vec_RV,lenght));  //Punkt 1
804   *POINT2 = V3.Add(*vec_AP, V31.Multiply(*vec_RV,-lenght)); //Punkt 2
805
806 }
807
808
809
810 //generiere String für Ausgabe
811 calcAxis::stringOutput()
812 {
813   //Typen casten (konvertieren)
814
815   //Konvertieren der double in CString mit vier Nachkommastellen
816   sOSG_APX.Format("%.3f",OSG_APX); //konvertieren in String mit 4 Nachkommastellen
817   sOSG_APY.Format("%.3f",OSG_APY);
818   sOSG_APZ.Format("%.3f",OSG_APZ);
819   sOSG_RVX.Format("%.3f",OSG_RVX);
820   sOSG_RVY.Format("%.3f",OSG_RVY);
821   sOSG_RVZ.Format("%.3f",OSG_RVZ);
822
823   sUSG_APX.Format("%.3f",USG_APX); //konvertieren in String mit 4 Nachkommastellen
824   sUSG_APY.Format("%.3f",USG_APY);
825   sUSG_APZ.Format("%.3f",USG_APZ);
826   sUSG_RVX.Format("%.3f",USG_RVX);
827   sUSG_RVY.Format("%.3f",USG_RVY);
828   sUSG_RVZ.Format("%.3f",USG_RVZ);
829
830
831   osg_P1X.Format("%.3f",OSGA_P1.X); //konvertiert Vektoreinträge in String
832   osg_P1Y.Format("%.3f",OSGA_P1.Y);
833   osg_P1Z.Format("%.3f",OSGA_P1.Z);
834   osg_P2X.Format("%.3f",OSGA_P2.X);
835   osg_P2Y.Format("%.3f",OSGA_P2.Y);
836   osg_P2Z.Format("%.3f",OSGA_P2.Z);
837
```

```
838     usg_P1X.Format("%.3f",USGA_P1.X); //konvertiert Vektoreinträge in String
839     usg_P1Y.Format("%.3f",USGA_P1.Y);
840     usg_P1Z.Format("%.3f",USGA_P1.Z);
841     usg_P2X.Format("%.3f",USGA_P2.X);
842     usg_P2Y.Format("%.3f",USGA_P2.Y);
843     usg_P2Z.Format("%.3f",USGA_P2.Z);
844
845     meshOSG_APX.Format("%.3f", mOSG_AP.X);  //Konvertiere VektoreintrLge in String, gemessenen OSGA
846     meshOSG_APY.Format("%.3f", mOSG_AP.Y);
847     meshOSG_APZ.Format("%.3f", mOSG_AP.Z);
848     meshOSG_RVX.Format("%.3f", mOSG_RV.X);  //Konvertiere VektoreintrLge in String, gemessenen OSGA
849     meshOSG_RVY.Format("%.3f", mOSG_RV.Y);
850     meshOSG_RVZ.Format("%.3f", mOSG_RV.Z);
851     sOSG_inc_mesh.Format("%.1f", mOSG_Inc);
852     sOSG_dev_mesh.Format("%.1f", mOSG_Dev);
853
854     //USGA1 (mit gemessener OSGA MM)
855     sUSG_APX_1.Format("%.3f", USG_APX1);
856     sUSG_APY_1.Format("%.3f", USG_APY1);
857     sUSG_APZ_1.Format("%.3f", USG_APZ1);
858     sUSG_RVX_1.Format("%.3f", USG_RVX1);
859     sUSG_RVY_1.Format("%.3f", USG_RVY1);
860     sUSG_RVZ_1.Format("%.3f", USG_RVZ1);
861
862     sUSG_dev_1.Format("%.1f", USG_dev1);
863     sUSG_inc_1.Format("%.1f", USG_inc1);
864
865     sUSG_P1X_1.Format("%.3f", USGA_P1_1.X);
866     sUSG_P1Y_1.Format("%.3f", USGA_P1_1.Y);
867     sUSG_P1Z_1.Format("%.3f", USGA_P1_1.Z);
868     sUSG_P2X_1.Format("%.3f", USGA_P2_1.X);
869     sUSG_P2Y_1.Format("%.3f", USGA_P2_1.Y);
870     sUSG_P2Z_1.Format("%.3f", USGA_P2_1.Z);
871     sLever_USG_1.Format("%.3f", lever_USG_1);
872
873     //USGA2 (nicht rotiert)
874     sUSG_APX_2.Format("%.3f", USG_APX2);
875     sUSG_APY_2.Format("%.3f", USG_APY2);
876     sUSG_APZ_2.Format("%.3f", USG_APZ2);
877     sUSG_RVX_2.Format("%.3f", USG_RVX2);
878     sUSG_RVY_2.Format("%.3f", USG_RVY2);
879     sUSG_RVZ_2.Format("%.3f", USG_RVZ2);
880
881     sUSG_dev_2.Format("%.1f", USG_dev2);
882     sUSG_inc_2.Format("%.1f", USG_inc2);
883
884     sUSG_P1X_2.Format("%.3f", USGA_P1_2.X);
885     sUSG_P1Y_2.Format("%.3f", USGA_P1_2.Y);
886     sUSG_P1Z_2.Format("%.3f", USGA_P1_2.Z);
887     sUSG_P2X_2.Format("%.3f", USGA_P2_2.X);
888     sUSG_P2Y_2.Format("%.3f", USGA_P2_2.Y);
889     sUSG_P2Z_2.Format("%.3f", USGA_P2_2.Z);
890     sLever_USG_2.Format("%.3f", lever_USG_2);
891
892     sUO_winkel.Format("%.1f",Angle_OSG_USG);
893     sUO_dist.Format("%.1f",Dist);
894
895     //Messung MM ML
896     sMM_Distance.Format("%.2f",dMM_Distance);
897     sMMX.Format("%.3f",MM.X);
898     sMMY.Format("%.3f",MM.Y);
899     sMMZ.Format("%.3f",MM.Z);
900     sMLX.Format("%.3f",ML.X);
901     sMLY.Format("%.3f",ML.Y);
902     sMLZ.Format("%.3f",ML.Z);
903     sLever_OSG_MM.Format("%.1f", lever_OSG_MM);
904
905     sOSG_winkel.Format("%.1f",OSG_winkel);
906     sOSG_dev.Format("%.1f",OSG_Dev);
907     sOSG_inc.Format("%.1f",OSG_Inc);
908     sOSG_lever.Format("%.1f",Lever_OSGA);
909
910     sUSG_winkel.Format("%.1f",USG_winkel);
911     sUSG_dev.Format("%.1f",USG_Dev);
912     sUSG_inc.Format("%.1f",USG_Inc);
913     sUSG_lever.Format("%.1f",Lever_USGA);
914
915     //Ersetzen des "." durch "," da excel das sonst nicht kapiert
916     sOSG_inc_mesh.Replace(".",",");
917     sOSG_dev_mesh.Replace(".",",");
918     meshOSG_APX.Replace(".",",");
919     meshOSG_APY.Replace(".",",");
920     meshOSG_APZ.Replace(".",",");
921     meshOSG_RVX.Replace(".",",");
```

```
922     meshOSG_RVY.Replace(".",",");
923     meshOSG_RVZ.Replace(".",",");
924
925     sOSG_APX.Replace(".",",");
926     sOSG_APY.Replace(".",",");
927     sOSG_APZ.Replace(".",",");
928     sOSG_RVX.Replace(".",",");
929     sOSG_RVY.Replace(".",",");
930     sOSG_RVZ.Replace(".",",");
931     sUSG_APX.Replace(".",",");
932     sUSG_APY.Replace(".",",");
933     sUSG_APZ.Replace(".",",");
934     sUSG_RVX.Replace(".",",");
935     sUSG_RVY.Replace(".",",");
936     sUSG_RVZ.Replace(".",",");
937
938     osg_P1X.Replace(".",",");
939     osg_P1Y.Replace(".",",");
940     osg_P1Z.Replace(".",",");
941     osg_P2X.Replace(".",",");
942     osg_P2Y.Replace(".",",");
943     osg_P2Z.Replace(".",",");
944     usg_P1X.Replace(".",",");
945     usg_P1Y.Replace(".",",");
946     usg_P1Z.Replace(".",",");
947     usg_P2X.Replace(".",",");
948     usg_P2Y.Replace(".",",");
949     usg_P2Z.Replace(".",",");
950
951     //USGA1 (rotiert um MM OSGA)
952     sUSG_APX_1.Replace(".",",");
953     sUSG_APY_1.Replace(".",",");
954     sUSG_APZ_1.Replace(".",",");
955     sUSG_RVX_1.Replace(".",",");
956     sUSG_RVY_1.Replace(".",",");
957     sUSG_RVZ_1.Replace(".",",");
958
959     sUSG_dev_1.Replace(".",",");
960     sUSG_inc_1.Replace(".",",");
961
962     sUSG_P1X_1.Replace(".",",");
963     sUSG_P1Y_1.Replace(".",",");
964     sUSG_P1Z_1.Replace(".",",");
965     sUSG_P2X_1.Replace(".",",");
966     sUSG_P2Y_1.Replace(".",",");
967     sUSG_P2Z_1.Replace(".",",");
968     sLever_USG_1.Replace(".",",");
969
970     //USGA2 (nicht rotiert)
971     sUSG_APX_2.Replace(".",",");
972     sUSG_APY_2.Replace(".",",");
973     sUSG_APZ_2.Replace(".",",");
974     sUSG_RVX_2.Replace(".",",");
975     sUSG_RVY_2.Replace(".",",");
976     sUSG_RVZ_2.Replace(".",",");
977
978     sUSG_dev_2.Replace(".",",");
979     sUSG_inc_2.Replace(".",",");
980
981     sUSG_P1X_2.Replace(".",",");
982     sUSG_P1Y_2.Replace(".",",");
983     sUSG_P1Z_2.Replace(".",",");
984     sUSG_P2X_2.Replace(".",",");
985     sUSG_P2Y_2.Replace(".",",");
986     sUSG_P2Z_2.Replace(".",",");
987     sLever_USG_2.Replace(".",",");
988
989     sOSG_winkel.Replace(".",",");
990     sOSG_dev.Replace(".",",");
991     sOSG_inc.Replace(".",",");
992     sOSG_lever.Replace(".",",");
993     sUSG_winkel.Replace(".",",");
994     sUSG_dev.Replace(".",",");
995     sUSG_inc.Replace(".",",");
996     sUSG_lever.Replace(".",",");
997
998     sLever_OSG_MM.Replace(".",",");
999
1000    sUO_winkel.Replace(".",",");
1001    sUO_dist.Replace(".",",");
1002
1003    sMM_Distance.Replace(".",",");
1004    sMMX.Replace(".",",");
1005    sMMY.Replace(".",",");
```

```
1006    sMMZ.Replace(".",",");
1007    sMLX.Replace(".",",");
1008    sMLY.Replace(".",",");
1009    sMLZ.Replace(".",",");
1010    //generierend des Output Strings
1011
1012    //Für Ausgabe aller USGA
1013    //Ausgabe = "OSGA;          ;       ;   ;           ;           ;       ;       ;       ;   ;
            ;                   ;   ;   USGA        ;       ;   ;           ;           ;
            ;       ;   ;   ;                   ;   ;   ;USGA u.OSGA;;OSGA gemessen
        ;;;;;;;;;;;;;;USGA1(rotiert um Malleolenachse);;;;;;;;;;;;;USGA2 (nicht rotiert)\r\n";
1014    //Ausgabe += "Punkt 1/2 (X);(Y);(Z);Drehwinkel;Inklination;Deviation;Hebel;Aufpunkt (X);(Y);(Z)
        ;Richtungsvektor (X);(Y);(Z);Punkt 1/2 (X);(Y) ;(Z);Drehwinkel;Inklination;Deviation;Hebel
        ;Aufpunkt (X);(Y);(Z);Richtungsvektor (X);(Y);(Z);Abstand;Winkel; Aufpunkt (X); (Y);(Z);
        Richtungsvektor (X);(Y);(Z);Deviation;Inklination;Malleolenpunkte(lateralis/medialis);;;
        Hebel;Abstand Malleolen;Punkt 1/2 (X);(Y);(Z);Aufpunkt(X);(Y);(Z);Richtungsvektor(X);(Y);(
        Z);Inklination;Deviation;Hebel;Punkt 1/2 (X);(Y);(Z);Aufpunkt(X);(Y);(Z);Richtungsvektor(X
        );(Y);(Z);Inklination;Deviation;Hebel\r\n";
1015    //Ausgabe += osg_P1X + ";" +osg_P1Y + ";" +osg_P1Z + ";" + sOSG_winkel + ";" + sOSG_inc + ";" +
         sOSG_dev + ";" +sOSG_lever + ";" + sOSG_APX + ";" +  sOSG_APY + ";" + sOSG_APZ + ";" +
        sOSG_RVX + ";" + sOSG_RVY + ";" + sOSG_RVZ + ";" + usg_P1X + ";" + usg_P1Y + ";" + usg_P1Z
         + ";" + sUSG_winkel + ";" + sUSG_inc + ";" + sUSG_dev + ";" + sUSG_lever + ";"+ sUSG_APX
        + ";" + sUSG_APY + ";" + sUSG_APZ + ";" + sUSG_RVX + ";" + sUSG_RVY + ";" + sUSG_RVZ + ";"
         + sUO_dist + ";" + sUO_winkel + ";" + meshOSG_APX + ";" + meshOSG_APY + ";" + meshOSG_APZ
         + ";" + meshOSG_RVX + ";" + meshOSG_RVY + ";" + meshOSG_RVZ +";"+ sOSG_dev_mesh +";"+
        sOSG_inc_mesh +";" + sMLX + ";" + sMLY + ";" + sMLZ + ";"+ sLever_OSG_MM + ";" +
        sMM_Distance + ";" + sUSG_P1X_1 + ";" + sUSG_P1Y_1 + ";" + sUSG_P1Z_1 + ";" + sUSG_APX_1 +
         ";" + sUSG_APY_1 + ";" + sUSG_APZ_1 + ";" + sUSG_RVX_1 + ";" + sUSG_RVY_1 + ";" +
        sUSG_RVZ_1 + ";" + sUSG_inc_1 + ";" + sUSG_dev_1 + ";" + sLever_USG_1 +";" + sUSG_P1X_2 +
        ";" + sUSG_P1Y_2 +";"+ sUSG_P1Z_2 +";"+ sUSG_APX_2 +";"+ sUSG_APY_2 +";"+ sUSG_APZ_2 +";"+
         sUSG_RVX_2 +";"+ sUSG_RVY_2 +";"+ sUSG_RVZ_2 +";"+ sUSG_inc_2 +";"+ sUSG_dev_2 +";"+
        sLever_USG_2 +"\r\n";
1016    //Ausgabe += osg_P2X +";"+osg_P2Y +";"+osg_P2Z +";;;;;;;;;;;" + usg_P2X + ";" + usg_P2Y + ";" +
         usg_P2Z + ";;;;;;;;;;;;;;;;;;;;;;;" + sMMX + ";" + sMMY + ";" + sMMZ + ";;;" + sUSG_P2X_1 +
        ";" + sUSG_P2Y_1 + ";" + sUSG_P2Z_1 + ";;;;;;;;;;"+ sUSG_P2X_2 +";"+ sUSG_P2Y_2 +";"+
        sUSG_P2Z_2 +"\r\n";
1017
1018    Ausgabe = "OSGA;\r\n";//        ;       ;   ;           ;           ;       ;       ;
                ;   ;   ;                   ;   ;   USGA        ;       ;   ;           ;
            ;           ;       ;           ;   ;   ;                   ;   ;   ;USGA u.OSGA;;
        OSGA gemessen;;;;;;;;;;;;;USGA1(rotiert um Malleolenachse);;;;;;;;;;;;;USGA2 (nicht rotiert
        )\r\n";
1019    Ausgabe += "Inklination;Deviation;Hebel;Drehwinkel;Aufpunkt␣(X);(Y);(Z);Richtungsvektor␣(X);(Y)
        ;(Z);\r\n";  //Punkt 1/2 (X);(Y) ;(Z);Drehwinkel;Inklination;Deviation;Hebel;Aufpunkt (X)
        ;(Y);(Z);Richtungsvektor (X);(Y);(Z);Abstand;Winkel; Aufpunkt (X); (Y);(Z);Richtungsvektor
         (X);(Y);(Z);Deviation;Inklination;Malleolenpunkte(lateralis/medialis);;;Hebel;Abstand
        Malleolen;Punkt 1/2 (X);(Y);(Z);Aufpunkt(X);(Y);(Z);Richtungsvektor(X);(Y);(Z);Inklination
        ;Deviation;Hebel;Punkt 1/2 (X);(Y);(Z);Aufpunkt(X);(Y);(Z);Richtungsvektor(X);(Y);(Z);
        Inklination;Deviation;Hebel\r\n";
1020    Ausgabe += sOSG_inc_mesh + ";" + sOSG_dev_mesh + ";" + sLever_OSG_MM + ";" + sOSG_winkel + ";"
        + meshOSG_APX + ";" +  meshOSG_APY + ";" + meshOSG_APZ + ";" + meshOSG_RVX + ";" +
        meshOSG_RVY + ";" + meshOSG_RVZ + ";\r\n";    //  + usg_P1X + ";" + usg_P1Y + ";" +
        usg_P1Z + ";" + sUSG_winkel + ";" + sUSG_inc + ";" + sUSG_dev + ";" + sUSG_lever + ";"+
        sUSG_APX + ";" + sUSG_APY + ";" + sUSG_APZ + ";" + sUSG_RVX + ";" + sUSG_RVY + ";" +
        sUSG_RVZ + ";" + sUO_dist + ";" + sUO_winkel + ";" + meshOSG_APX + ";" + meshOSG_APY + ";"
         + meshOSG_APZ + ";" + meshOSG_RVX + ";" + meshOSG_RVY + ";" + meshOSG_RVZ +";"+
        sOSG_dev_mesh +";"+ sOSG_inc_mesh +";" + sMLX + ";" + sMLY + ";" + sMLZ + ";"+
        sLever_OSG_MM + ";" + sMM_Distance + ";" + sUSG_P1X_1 + ";" + sUSG_P1Y_1 + ";" +
        sUSG_P1Z_1 + ";" + sUSG_APX_1 + ";" + sUSG_APY_1 + ";" + sUSG_APZ_1 + ";" + sUSG_RVX_1 +
        ";" + sUSG_RVY_1 + ";" + sUSG_RVZ_1 + ";" + sUSG_inc_1 + ";" + sUSG_dev_1 + ";" +
        sLever_USG_1 +";" + sUSG_P1X_2 + ";" + sUSG_P1Y_2 +";"+ sUSG_P1Z_2 +";"+ sUSG_APX_2 +";"+
        sUSG_APY_2 +";"+ sUSG_APZ_2 +";"+ sUSG_RVX_2 +";"+ sUSG_RVY_2 +";"+ sUSG_RVZ_2 +";"+
        sUSG_inc_2 +";"+ sUSG_dev_2 +";"+ sLever_USG_2 +"\r\n";
1021
1022    //meshOSG_APX + ";" + meshOSG_APY + ";" + meshOSG_APZ + ";" + meshOSG_RVX + ";" + meshOSG_RVY +
         ";" + meshOSG_RVZ +";"+ sOSG_dev_mesh +";"+ sOSG_inc_mesh +";" + sMLX + ";" + sMLY + ";"
        + sMLZ + ";"+ sLever_OSG_MM + ";" + sMM_Distance
1023 }
1024
1025 //Destruktor (Variablen leeren, Speicherfrei machen)
1026 calcAxis::~calcAxis()
1027 {
1028
1029
1030 }
```

D Beispiel einer Anwendung der Klasse calcAxis

Zur Berechnung von Rotationsachsen mit Hilfe der Klasse *calcAxis* müssen nur wenige Methoden der Klasse aufgerufen werden. Der nachfolgende Programmauszug soll beispielhaft zeigen wie die C++ Klasse angewendet werden kann.

```
1   calcAxis achse;
2   achse.Leg = R;
3   achse.coordTibia(p1x1,p1y1,p1z1,p2x1,p2y1,p2z1,
4                    p1x2,p1y2,p1z2,p2x2,p2y2,p2z2,
5                    p1x3,p1y3,p1z3,p2x3,p2y3,p2z3,
6                    p1x4,p1y4,p1z4,p2x4,p2y4,p2z4);
7   achse.calcOSG(x1,y1,z1,x2,y2,z2,x3,y3,z3);
8   achse.calcUSG(x1,y1,z1,x2,y2,z2,x3,y3,z3);
```

Zeile eins des Beispiels initiiert die Klasse *calcAxis*. Damit stehen der Klasse *achse* alle Eigenschaften und Methoden von *calcAxis* zur Verfügung. In Zeile zwei wird Variable *Leg* gesetzt. In diesem Beispiel werden alle nachfolgenden Berechnungen in einem Rechtssystem durchgeführt. Die Methode *coordTibia* in Zeile drei berechnet aus den Koordinatendaten der Markerpunkte P_1 und P_2 des Taststiftes, die anatomischen Punkten des *Tibiakoordinatensystems*. Alle Punkte die der Klasse *achse* übergeben werden, werden in dieses Koordinatensystem transformiert. Ebenfalls wird der Eichfaktor für die Temperaturabhängigkeit des Ultraschall über den bekannten Abstand der Markerpunkte P_1 und P_2 berechnet. In Zeile 7 und Zeile 8, werden jeweils Rotationsachsen berechnet, mit dem Unterschied, dass die Achse aus Zeile 8 um die Rotationsachse aus Zeile 7 rotiert wird. Mit Hilfe der auf Seite 49 beschriebenen Eigenschaften, können alle Ergebnisse ausgelesen werden.

E C++ Klasse zebClass

```
1  // zebClass.cpp: Implementierung der Klasse zebClass.
2  //
3  //////////////////////////////////////////////////////////////////////
4
5  #include "stdafx.h"
6  #include "zebClass.h"
7
8  //////////////////////////////////////////////////////////////////////
9  // Konstruktion/Destruktion
10 //////////////////////////////////////////////////////////////////////
11
12 zebClass::zebClass()
13 {
14    err            = true;
15    device         = 0;
16    totalAtomCount = 0;
17
18    motionInput = 0;
19    motionFrame = 0;
20
21    //Digital Input
22    digitalInput = 0;
23
24    //Variable zum Fehler abfangen
25    zebError = false;
26
27 }
28
29 ////////////////////////////////////
30 //Fehler abfangen
31 zebClass::zebCheck(BOOL result)
32 {
33    if(!result) {
34       CString LastError = Zeb_GetLastError();
35       MessageBox(0,LastError,"Error",MB_ICONERROR+MB_SETFOREGROUND+MB_OK);
36       zebError = true;
37       closeDialog = 1;
38    }
39
40 }//Fehler abfangen
41
42 ////////////////////////////////////
43 //Digital INIT (Fußschalter)
44 BOOL zebClass::digitalInit() {
45
46    ZebComponent* component;
47
48     if(!ZebDevice_FindComponent(device, "digital_input", &component))
49        return TRUE;
50
51     if(!ZebDigitalInput_Get(component, &digitalInput))
52        return FALSE;
53
54     if (!ZebDigitalInput_GetChannelCount(digitalInput, &digitalChannelCount))
55        return FALSE;
56
57     digitalBytesPerQuant = (digitalChannelCount+7)/8;
58
59     if (!ZebDigitalInput_SetQuantDuration(digitalInput, 1.0/15)) // 15 Hz
60        return FALSE;
61
62     //cout << "digital input" << endl;
63     //cout << "channels: " << digitalChannelCount << endl;
64     //cout << endl;
65
66     return TRUE;
67 }
68
69 //Digital Update Conf
70 BOOL zebClass::digitalUpdateCfg() {
71
72     if (!digitalInput)
73        return TRUE;
74
75     if (!ZebDigitalInput_GetQuantsPerAtom(digitalInput, &digitalQuantsPerAtom))
76        return FALSE;
77
78     //cout << "digital input" << endl;
79     //cout << "frequency: " << digitalQuantsPerAtom/atomDuration << endl;
80     //cout << "quants per atom: " << digitalQuantsPerAtom << endl;
81     //cout << endl;
```

```
82     return TRUE;
83 }
84
85 //Digital Feed
86 BOOL zebClass::digitalFeed(int atomCount) {
87      if (!digitalInput)
88        return TRUE;
89
90      //cout << "digital input HHHHHHHHHHHHH" << endl;
91
92     const byte* pos;
93
94     if (!ZebDigitalInput_GetPos(digitalInput, &pos))
95        return FALSE;
96
97
98        const byte* ptr  = pos;
99        byte        mask = 1; //Schalter 1
100     byte        mask2 = 0x2;  //Schalter 2
101
102     if (*ptr & mask) ON1 = 1; //Binärvergleich
103     else ON1 = 0;
104
105     if (*ptr & mask2) ON2 = 1;
106     else ON2 = 0;
107
108       //cout << endl;
109
110
111    return TRUE;
112 }
113 //Digital
114 ////////////////////////////////////////////////////////////////////////////
115
116
117 //ZebMotionInit
118 BOOL zebClass::zebMotionInit() {
119   CString sMotionChannelCount;
120   ZebComponent* component;
121
122    zebCheck(ZebDevice_FindComponent(device, "motion_input", &component));
123
124
125    zebCheck(ZebMotionInput_Get(component, &motionInput));
126
127 // request frequency
128    zebCheck(ZebMotionInput_SetQuantDuration(motionInput, 1.0/100));
129
130 // retrive frame
131    zebCheck(ZebMotionInput_GetFrame(motionInput, 0, &motionFrame));
132
133 // request channels
134    zebCheck(ZebMotionFrame_GetChannelCount(motionFrame, &motionChannelCount));
135
136    motionChannelCount = min(motionChannelCount, MAX_MOTION_CHANNELS);
137
138    for(int ch = 0; ch < motionChannelCount; ch++)
139    {
140       zebCheck(ZebMotionFrame_GetChannel(motionFrame, ch, &motionChannels[ch]));
141    }
142
143    zebStatus += "motion input\r\n";
144    zebStatus += "channels: ";
145   //sMotionChannelCount = (CString)motionChannelCount;
146    //sMotionChannelCount.Format("%",motionChannelCount);
147    sMotionChannelCount.Format("%d",motionChannelCount);
148
149    zebStatus += sMotionChannelCount;
150    //zebStatus += motionChannelCount;
151    zebStatus += "\r\n";
152
153
154    return TRUE;
155 }
156
157
158 //zebMotionUpdateCFG
159 BOOL zebClass::zebMotionUpdateCfg() {
160   //Variablen Configurieren
161   CString sMotionQ, sMotionQpA;
162   if (!motionInput)
163       return TRUE;
164
165    if (!ZebMotionInput_GetQuantsPerAtom(motionInput, &motionQuantsPerAtom))
```

```
166         return FALSE;
167
168     zebStatus += "motion input\r\n";
169     zebStatus += "frequency: ";
170     sMotionQ.Format("%.4f",motionQuantsPerAtom/atomDuration);
171     zebStatus += sMotionQ;
172     //zebStatus += motionQuantsPerAtom/atomDuration;
173     zebStatus += "\r\n";
174     zebStatus += "quants per atom: ";
175     sMotionQpA.Format("%d",motionQuantsPerAtom);
176     zebStatus += sMotionQpA;
177     //zebStatus += motionQuantsPerAtom;
178     zebStatus += "\r\n";
179
180
181     return TRUE;
182 }
183
184
185 //zebMotionFeed
186 BOOL zebClass::zebMotionFeed(int atomCount) {
187
188    if (!motionInput)
189        return TRUE;
190
191     //zebStatus += "Auslesen der Koordinaten\r\n";
192
193     const ZebVector3* pos[MAX_MOTION_CHANNELS];
194
195     for(int ch = 0; ch < motionChannelCount; ch++)
196     {
197         if (!ZebMotionChannel_GetPos(motionChannels[ch], &pos[ch]))
198            return FALSE;
199     }
200
201     for(int ctr = atomCount*motionQuantsPerAtom; ctr--; )
202     {
203         for(int ch = 0; ch < motionChannelCount; ch++)
204         {
205             const ZebVector3* v = pos[ch]++;
206             //augabe in formular
207       if(ch==0) //Ausgabe 1. Marker
208       {
209        zebX1 = v->x;
210        zebY1 = v->y;
211        zebZ1 = v->z;
212       }
213       if(ch==1) //Ausgabe 2. Marker
214       {
215         zebX2 = v->x;
216         zebY2 = v->y;
217         zebZ2 = v->z;
218       }
219       if(ch==2) //Ausgabe 3. Marker
220       {
221         zebX3 = v->x;
222         zebY3 = v->y;
223         zebZ3 = v->z;
224       }
225       if(ch==3) //Ausgabe 4. Marker
226       {
227         zebX4 = v->x;
228         zebY4 = v->y;
229         zebZ4 = v->z;
230       }
231
232        }
233
234
235     }
236
237     return TRUE;
238
239 }
240
241 //ZebFeedData Single
242 BOOL zebClass::zebFeedDataSingle() {
243
244    //Variablen definieren
245    ZebReadInfo info;
246
247    zebCheck(ZebDevice_Read(device, &info));
248
249         if (info.atomCount)
```

```
250         {
251             if (info.syncStart)
252             totalAtomCount += info.atomCount;
253             zebCheck(zebMotionFeed(info.atomCount));
254         zebCheck(digitalFeed(info.atomCount));
255         }
256   return TRUE;
257 }
258
259
260 ////////////////////////////////////////////////
261 //Zebris Configuration
262 zebClass::zebConf() {
263
264   if (!ZebDeviceManager_Init("data"))
265       error = true;
266   // execute the hardware setup
267   if (!ZebDeviceManager_Setup(0))
268       error = true;
269 }
270
271
272
273 ////////////////////////////////////////////////
274 //ZebFeedData Schleife
275 BOOL zebClass::zebFeedData(DWORD time_ms, bool exitOnStop) {
276
277   DWORD       startTime = GetTickCount();
278    bool        stopped = true;
279    ZebReadInfo info;
280
281    do
282    {
283        if (!ZebDevice_Read(device, &info))
284            return FALSE;
285
286        if (info.atomCount)
287        {
288            stopped = false;
289
290            if (info.syncStart)
291                //cout << "Started" << endl;
292
293            //cout << "Atoms " << totalAtomCount;
294
295            totalAtomCount += info.atomCount;
296
297            //cout << "-" << totalAtomCount << endl;
298
299            if (!zebMotionFeed(info.atomCount))
300                return FALSE;
301        }
302
303        if (info.dataOver)
304        {
305            if (!stopped)
306            {
307              // cout << "Stopped" << endl;
308                stopped = true;
309            }
310
311            if (exitOnStop)
312                return TRUE;
313        }
314        //if (kbhit())    return TRUE;
315    }
316    while(GetTickCount() - startTime < time_ms);
317
318    return TRUE;
319
320 }
321
322
323 //initialisiert Zebris, nur beim Start des Programms
324 zebClass::zebHdwInit()
325 {
326
327     //Variablen definieren
328   CString sAtomDuration;
329   // initialize the device manager
330   if(!zebError) {
331     zebCheck(ZebDeviceManager_Init("data"));
332         zebStatus = "Hardware wird initialisiert\r\n";
333   }
```

```
334   // get the device
335    if(!zebError) zebCheck(ZebDeviceManager_GetDevice(&device));
336
337    // configure the device
338    if (!zebError) zebCheck(zebMotionInit());
339    if (!zebError) zebCheck(digitalInit());
340    if (!zebError) zebCheck(ZebDevice_Activate(device));
341    // read-out the granted configuration after activation
342    if (!zebError) zebCheck(ZebDevice_GetAtomDuration(device, &atomDuration));
343    if (!zebError) zebCheck(zebMotionUpdateCfg());
344    if (!zebError) zebCheck(digitalUpdateCfg());
345 }
346
347
348 //startet Zebris Hardware
349 zebClass::zebStart() {
350
351   //Variablen definieren
352   CString sAtomDuration;
353   zebStatus = "";
354   zebError=false;
355   // get the device
356   /* if(!zebError) zebCheck(ZebDeviceManager_GetDevice(&device));
357
358    // configure the device
359    if (!zebError) zebCheck(zebMotionInit());
360    if (!zebError) zebCheck(ZebDevice_Activate(device));
361    // read-out the granted configuration after activation
362    if (!zebError) zebCheck(ZebDevice_GetAtomDuration(device, &atomDuration));
363    if (!zebError) zebCheck(zebMotionUpdateCfg() );*/
364    // start the device
365    if (!zebError) zebCheck(ZebDevice_Start(device));
366   // pull the data within 10 sec
367    zebStatus += "Start the device\r\n";
368    zebStatus += "Auslesen der Koordinaten\r\n";
369 }
370
371
372 //liest Wert aus Zebris Hardware (Schleife)
373 zebClass::zebGet() {
374
375   if (!zebFeedData(10000, false))
376       error = true;
377 }
378
379 //liest einzelnen Wert aus Zebris aus
380 zebClass::zebGetSingle() {
381
382   zebCheck(zebFeedDataSingle());
383
384 }
385
386
387 //stoppt Zebris Hardware
388 zebClass::zebStop() {
389
390   //Variable definieren
391   CString sTotalAtomCount;
392   // stop the device
393    zebCheck(ZebDevice_Stop(device));
394   // pull the remaining data
395    zebCheck(zebFeedData(100000, true));
396    //verbindung zum zebris schliessen
397   // ZebDevice_Release(device);
398
399 }
400
401 zebClass::~zebClass()
402 {
403
404 }
```

F C++ Klasse USG, das Hauptprogramm

```
1  // usgDlg.cpp : Implementierungsdatei
2  //
3
4  #include "stdafx.h"
5  #include "usg.h"
6  #include "usgDlg.h"
7  #include "common.h"
8  #include "afxtempl.h"  //CArray Klasse
9  #include "XMath.h"
10 #include <math.h>
11 #include "mmsystem.h"
12 #ifdef _DEBUG
13 #define new DEBUG_NEW
14 #undef THIS_FILE
15 static char THIS_FILE[] = __FILE__;
16 #endif
17
18 /////////////////////////////////////////////////////////////////////////////
19 // CAboutDlg-Dialogfeld für Anwendungsbefehl "Info"
20
21 class CAboutDlg : public CDialog
22 {
23 public:
24   CAboutDlg();
25
26 // Dialogfelddaten
27   //{{AFX_DATA(CAboutDlg)
28   enum { IDD = IDD_ABOUTBOX };
29   //}}AFX_DATA
30
31   // Vom Klassenassistenten generierte Überladungen virtueller Funktionen
32   //{{AFX_VIRTUAL(CAboutDlg)
33   protected:
34   virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV-Unterstützung
35   //}}AFX_VIRTUAL
36
37 // Implementierung
38 protected:
39   //{{AFX_MSG(CAboutDlg)
40   //}}AFX_MSG
41   DECLARE_MESSAGE_MAP()
42 };
43
44 CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
45 {
46   //{{AFX_DATA_INIT(CAboutDlg)
47   //}}AFX_DATA_INIT
48 }
49
50 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
51 {
52   CDialog::DoDataExchange(pDX);
53   //{{AFX_DATA_MAP(CAboutDlg)
54   //}}AFX_DATA_MAP
55 }
56
57 BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
58   //{{AFX_MSG_MAP(CAboutDlg)
59     // Keine Nachrichten-Handler
60   //}}AFX_MSG_MAP
61 END_MESSAGE_MAP()
62
63 /////////////////////////////////////////////////////////////////////////////
64 // CUsgDlg Dialogfeld
65
66 CUsgDlg::CUsgDlg(CWnd* pParent /*=NULL*/)
67   : CDialog(CUsgDlg::IDD, pParent)
68 {
69   //{{AFX_DATA_INIT(CUsgDlg)
70   //}}AFX_DATA_INIT
71   // Beachten Sie, dass LoadIcon unter Win32 keinen nachfolgenden DestroyIcon-Aufruf benötigt
72   m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
73 }
74
75 void CUsgDlg::DoDataExchange(CDataExchange* pDX)
76 {
77   CDialog::DoDataExchange(pDX);
78   //{{AFX_DATA_MAP(CUsgDlg)
79   DDX_Control(pDX, IDC_BUTTON2, m_fooButtonCopy2CB);
80   DDX_Control(pDX, IDC_FOOTTYPE, m_fooButtonPaint);
```

```
81    DDX_Control(pDX, IDC_VIEW2, m_fooButtonView2);
82    DDX_Control(pDX, IDC_YZ, m_fooButtonYZ);
83    DDX_Control(pDX, IDC_XY, m_fooButtonXY);
84    DDX_Control(pDX, IDC_XZ, m_fooButtonXZ);
85    DDX_Control(pDX, IDC_VIEW, m_fooButtonView);
86    DDX_Control(pDX, IDC_PROPERTY, m_fooButtonProperty);
87    DDX_Control(pDX, IDC_RESET, m_fooButtonReset);
88    DDX_Control(pDX, IDC_MESSUNG, m_fooButtonMessung);
89    DDX_Control(pDX, IDC_KOOR, m_fooButtonKoor);
90    DDX_Control(pDX, IDC_NEU, m_fooButtonProband);
91    DDX_Control(pDX, IDOK, m_fooButtonOK);
92    DDX_Control(pDX, IDC_SAVE, m_fooButtonSave);
93    DDX_Control(pDX, IDC_NTGRAPH3D1, m_3D);
94    //}}AFX_DATA_MAP
95  }
96
97  BEGIN_MESSAGE_MAP(CUsgDlg, CDialog)
98    //{{AFX_MSG_MAP(CUsgDlg)
99    ON_WM_SYSCOMMAND()
100   ON_WM_PAINT()
101   ON_WM_QUERYDRAGICON()
102   ON_BN_CLICKED(IDC_NEU, OnNeu)
103   ON_COMMAND(PROB_NEW, OnProbNew)
104   ON_COMMAND(MESS_SETTINGS, OnSettings)
105   ON_COMMAND(HELP_CONTENT, OnContent)
106   ON_COMMAND(HELP_INFO, OnInfo)
107   ON_COMMAND(MESS_XYZ, OnXyz)
108   ON_BN_CLICKED(IDC_KOOR, OnKoor)
109   ON_BN_CLICKED(IDC_RESET, OnReset)
110   ON_COMMAND(FILE_SAVE, OnSave)
111   ON_COMMAND(FILE_OPEN, OnOpen)
112   ON_BN_CLICKED(IDC_MESSUNG, OnMessung)
113   ON_COMMAND(SAVEDATA, OnSAVEDATA)
114   ON_BN_CLICKED(IDC_CALCNEW, OnCalcnew)
115   ON_COMMAND(MENUCALCNEW, OnMENUCALCNEW)
116   ON_COMMAND(MESS_RESET, OnResetDialog)
117   ON_BN_CLICKED(IDC_MITTEL, OnMittel)
118   ON_BN_CLICKED(IDC_PUNKT3, OnPunkt3)
119   ON_BN_CLICKED(IDC_PROPERTY, OnProperty)
120   ON_BN_CLICKED(IDC_VIEW, OnView)
121   ON_BN_CLICKED(IDC_YZ, OnYz)
122   ON_BN_CLICKED(IDC_XY, OnXy)
123   ON_BN_CLICKED(IDC_XZ, OnXz)
124   ON_COMMAND(ID_ANSICHT_ZOOM, OnAnsichtZoom)
125   ON_COMMAND(ID_ANSICHT_ROT, OnAnsichtRot)
126   ON_COMMAND(ID_ANSICHT_PAN, OnAnsichtPan)
127   ON_COMMAND(ID_ANSICHT_SAGITALEBENEYZ, OnAnsichtSagitalebeneyz)
128   ON_COMMAND(ID_ANSICHT_TRANSVERSALEBENEXY, OnAnsichtTransversalebenexy)
129   ON_COMMAND(ID_ANSICHT_FRONTALEBENEXZ, OnAnsichtFrontalebenexz)
130   ON_COMMAND(ID_ANSICHT_EIGENSCHAFTEN, OnAnsichtEigenschaften)
131   ON_COMMAND(ID_ANSICHT_GRAPHKOPIEREN, OnAnsichtGraphkopieren)
132   ON_BN_CLICKED(IDC_VIEW2, OnView2)
133   ON_BN_CLICKED(IDC_FOOTTYPE, OnFootType)
134   ON_BN_CLICKED(IDC_SAVE, OnSave)
135   ON_COMMAND(MESS_PUNKT3, OnPunkt3)
136   ON_COMMAND(MESS_MITTEL, OnMittel)
137   ON_BN_CLICKED(IDC_BUTTON2, OnCopy2CB)
138   //}}AFX_MSG_MAP
139 END_MESSAGE_MAP()
140
141 /////////////////////////////////////////////////////////////////////////////
142 // CUsgDlg Nachrichten-Handler
143
144 BOOL CUsgDlg::OnInitDialog()
145 {
146   CDialog::OnInitDialog();
147
148   //Probandendaten
149   m_Proband.m_mFuss = 1;
150   m_Proband.m_pGeschlecht = 1;
151   m_Proband.m_pName = "Nachname, Vorname";
152     m_Proband.m_pAnamnese = "---";
153   m_Proband.m_pBemerkung = "---";
154   m_Proband.m_pGewicht = "---";
155   m_Proband.m_pBeins = "---";
156   m_Proband.m_mDatum = "---";
157   m_Proband.m_mBemerkung = "---";
158   m_Proband.m_pCode = "---";
159   m_Proband.m_pDatum = "dd.mm.yyyy";
160   m_Proband.m_pFuss = "300";
161   m_Proband.m_pTritt = "---";
162
163   //Variablen initiieren für Ausgabe Stab (Fehler) an GUI
164   m_iStabDev = 1;
```

```
165     m_iStabInc = 1;
166     m_iStabAng = 1;
167     m_iStabLever = 1;
168     // Hinzufügen des Menübefehls "Info..." zum Systemmenü.
169     output2 = 'm'; //Anzeigemodus Instantan/Finite
170     //Die Anzeige der Winkel formatiern
171     m_stcUsgInc.SubclassDlgItem(IDC_USGINC, this); //Klasse erstellen USGA Inc
172     m_stcUsgInc.SetTextColor(RGB(255,255,50));
173     m_stcUsgInc.SetBkColor(RGB(51,102,102));
174
175     m_stcUsgDev.SubclassDlgItem(IDC_USGDEV, this); //Klasse für DEV
176     m_stcUsgDev.SetTextColor(RGB(255,255,50));
177     m_stcUsgDev.SetBkColor(RGB(51,102,102));
178
179     m_stcUsgAngle.SubclassDlgItem(IDC_USGDREH, this); //Klasse erstellen Drehwinkel
180     m_stcUsgAngle.SetTextColor(RGB(255,255,50));
181     m_stcUsgAngle.SetBkColor(RGB(51,102,102));
182
183     m_stcUsgLever.SubclassDlgItem(IDC_USGHEBEL, this); //Klasse erstellen Hebel
184     m_stcUsgLever.SetTextColor(RGB(255,255,50));
185     m_stcUsgLever.SetBkColor(RGB(51,102,102));
186     ////////////////////////////////////////////////////////////////////////////7
187     //Die Anzeige der Winkel formatiern
188     m_stcOsgInc.SubclassDlgItem(IDC_OSGINC, this); //Klasse erstellen OSGA Inc
189     m_stcOsgInc.SetTextColor(RGB(255,255,50));
190     m_stcOsgInc.SetBkColor(RGB(51,102,102));
191
192     m_stcOsgDev.SubclassDlgItem(IDC_OSGDEV, this); //Klasse für DEV erstellen
193     m_stcOsgDev.SetTextColor(RGB(255,255,50));
194     m_stcOsgDev.SetBkColor(RGB(51,102,102));
195
196     m_stcOsgAngle.SubclassDlgItem(IDC_OSGDREH, this); //Klasse erstellen Drehwinkel
197     m_stcOsgAngle.SetTextColor(RGB(255,255,50));
198     m_stcOsgAngle.SetBkColor(RGB(51,102,102));
199
200     m_stcOsgLever.SubclassDlgItem(IDC_OSGHEBEL, this); //Klasse erstellen Hebel
201     m_stcOsgLever.SetTextColor(RGB(255,255,50));
202     m_stcOsgLever.SetBkColor(RGB(51,102,102));
203
204     //Klasse Font erstellen
205     Font.CreateFont(48, 0, 0, 0, FW_BOLD,0, 0, 0, DEFAULT_CHARSET,OUT_CHARACTER_PRECIS,
            CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,DEFAULT_PITCH | FF_DONTCARE, "Arial");
206     //Winkel1 Text font ändern
207     GetDlgItem(IDC_USGINC)->SetFont(&Font);
208     //Winkel2 Text Font ändern
209     GetDlgItem(IDC_USGDEV)->SetFont(&Font);
210     //Winkel3 Text Font ändern
211     GetDlgItem(IDC_USGDREH)->SetFont(&Font);
212     //Winkel4 Text Font ändern
213     GetDlgItem(IDC_USGHEBEL)->SetFont(&Font);
214
215     GetDlgItem(IDC_USGINC)->SetWindowText("--");
216     GetDlgItem(IDC_USGDEV)->SetWindowText("--");
217     GetDlgItem(IDC_USGDREH)->SetWindowText("--");
218     GetDlgItem(IDC_USGHEBEL)->SetWindowText("--");
219
220
221
222     ////////////////////////////////////////////////////////////////
223     //Winkel1 Text font ändern
224     GetDlgItem(IDC_OSGINC)->SetFont(&Font);
225     //Winkel2 Text Font ändern
226     GetDlgItem(IDC_OSGDEV)->SetFont(&Font);
227     //Winkel3 Text Font ändern
228     GetDlgItem(IDC_OSGDREH)->SetFont(&Font);
229     //Winkel4 Text Font ändern
230     GetDlgItem(IDC_OSGHEBEL)->SetFont(&Font);
231
232     GetDlgItem(IDC_OSGINC)->SetWindowText("--");
233     GetDlgItem(IDC_OSGDEV)->SetWindowText("--");
234     GetDlgItem(IDC_OSGDREH)->SetWindowText("--");
235     GetDlgItem(IDC_OSGHEBEL)->SetWindowText("--");
236
237     //Setzt die Variable zebFirstRun
238     zebFirstRun = true;
239
240     //Buttons
241     m_fooButtonSave.setBitmapId(IDB_SAVE, RGB(255,0,255));  //Save Button
242     m_fooButtonOK.setBitmapId(IDB_CLOSE, RGB(255,0,255)); //Close Button
243     m_fooButtonProband.setBitmapId(IDB_PROB, RGB(255,0,255)); //Probandendaten Button
244     m_fooButtonKoor.setBitmapId(IDB_XYZ1, RGB(255,0,255));     //Koordinatenfestlegen Button
245     m_fooButtonMessung.setBitmapId(IDB_MESS, RGB(255,0,255)); //Messung starten Button
246     m_fooButtonReset.setBitmapId(IDB_DEL, RGB(255,0,255));     //Zurücksetzen Button
247     m_fooButtonProperty.setBitmapId(IDB_XYZ, RGB(255,0,255)); //Property Button
```

```
248     m_fooButtonView.setBitmapId(IDB_VIEW, RGB(255,0,255));      //View Button (Rot,Pan,Zoom)
249     m_fooButtonXZ.setBitmapId(IDB_YZ, RGB(255,0,255));
250     m_fooButtonXY.setBitmapId(IDB_XY, RGB(255,0,255));
251     m_fooButtonYZ.setBitmapId(IDB_XZ, RGB(255,0,255));
252     m_fooButtonCopy2CB.setBitmapId(IDB_COPY2CB, RGB(255,0,255));
253     m_fooButtonView2.setBitmapId(IDB_VIEW2, RGB(255,0,255));  //view Button orthographic/
            perspektive
254     m_fooButtonPaint.setBitmapId(IDB_PAINT, RGB(255,0,255));  //Paint button
255
256
257     //group buttons
258     m_fooButtonXY.addToGroup(_T("view"));
259     m_fooButtonXZ.addToGroup(_T("view"));
260     m_fooButtonYZ.addToGroup(_T("view"));
261
262     //NTGraph3D Einstellungen
263     m_iTrackMode = 2;    //TrackMode = Rot
264     m_iView2 = 1;       //ansicht rechtwinklig/orthographic
265     m_3D.SetCaption("keine Daten");
266     m_3D.SetTrackMode(m_iTrackMode);     //Beim Dialogstart wird Rotation angeschalten
267     m_3D.SetProjection(m_iView2);          //Ansicht Orthographic
268     m_3D.SetRange(-150,150,0,300,0,300);  //Range
269     m_3D.SetCaptionColor(RGB(66,66,66));
270     m_3D.SetXGridNumber(6);
271     m_3D.SetYGridNumber(6);
272     m_3D.SetZGridNumber(6);
273
274     IFontDisp* TestFont;
275     //TestFont->Invoke();
276
277     //m_3D.SetFont(TestFont);
278     //fläche zeichnen
279     m_iFootType = 1;
280
281     // IDM_ABOUTBOX muss sich im Bereich der Systembefehle befinden.
282     ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
283     ASSERT(IDM_ABOUTBOX < 0xF000);
284
285     CMenu* pSysMenu = GetSystemMenu(FALSE);
286     if (pSysMenu != NULL)
287     {
288       CString strAboutMenu;
289       strAboutMenu.LoadString(IDS_ABOUTBOX);
290       if (!strAboutMenu.IsEmpty())
291       {
292         pSysMenu->AppendMenu(MF_SEPARATOR);
293         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
294       }
295     }
296
297     // Symbol für dieses Dialogfeld festlegen. Wird automatisch erledigt
298     //  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
299     SetIcon(m_hIcon, TRUE);      // Großes Symbol verwenden
300     SetIcon(m_hIcon, FALSE);     // Kleines Symbol verwenden
301
302     // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
303
304
305     return TRUE;  // Geben Sie TRUE zurück, außer ein Steuerelement soll den Fokus erhalten
306 }
307
308 void CUsgDlg::OnSysCommand(UINT nID, LPARAM lParam)
309 {
310     if ((nID & 0xFFF0) == IDM_ABOUTBOX)
311     {
312       CAboutDlg dlgAbout;
313       dlgAbout.DoModal();
314     }
315     else
316     {
317       CDialog::OnSysCommand(nID, lParam);
318     }
319 }
320
321 // Wollen Sie Ihrem Dialogfeld eine Schaltfläche "Minimieren" hinzufügen, benötigen Sie
322 //  den nachstehenden Code, um das Symbol zu zeichnen. Für MFC-Anwendungen, die das
323 //  Dokument/Ansicht-Modell verwenden, wird dies automatisch für Sie erledigt.
324
325 void CUsgDlg::OnPaint()
326 {
327     if (IsIconic())
328     {
329       CPaintDC dc(this); // Gerätekontext für Zeichnen
330
```

```
331       SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
332
333       // Symbol in Client-Rechteck zentrieren
334       int cxIcon = GetSystemMetrics(SM_CXICON);
335       int cyIcon = GetSystemMetrics(SM_CYICON);
336       CRect rect;
337       GetClientRect(&rect);
338       int x = (rect.Width() - cxIcon + 1) / 2;
339       int y = (rect.Height() - cyIcon + 1) / 2;
340
341       // Symbol zeichnen
342       dc.DrawIcon(x, y, m_hIcon);
343     }
344     else
345     {
346       CDialog::OnPaint();
347     }
348 }
349
350 // Die Systemaufrufe fragen den Cursorform ab, die angezeigt werden soll, während der Benutzer
351 //  das zum Symbol verkleinerte Fenster mit der Maus zieht.
352 HCURSOR CUsgDlg::OnQueryDragIcon()
353 {
354   return (HCURSOR) m_hIcon;
355 }
356
357 void CUsgDlg::OnNeu()
358 {
359   // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
360   m_Proband.DoModal();
361
362   CString m_cGeschlecht, m_cFuss;
363
364   if (m_Proband.m_pGeschlecht == 0) m_cGeschlecht = "männlich";
365   else m_cGeschlecht = "weiblich";
366
367   if (m_Proband.m_mFuss == 0) m_cFuss = "links";
368   else m_cFuss = "rechts";
369
370
371   //Zeile 1
372   m_ProbTemp = "";
373   m_ProbTemp += "\rDatenblatt␣OSGA/␣USGA␣Bestimmung\n"; //war mit \r
374   //Zeile 2
375   m_ProbTemp += "Probandendaten\n";
376   //Zeile 3
377   m_ProbTemp += "Name;␣Code;␣Geburtsdatum;␣Gewicht;␣Geschlecht;␣Fusslänge;␣Anamnese;␣Trittspur;␣
          Beinstellung;␣Bemerkung\n";
378   //Zeile 4
379   m_ProbTemp += m_Proband.m_pName +";"  +m_Proband.m_pCode +";" +m_Proband.m_pDatum +";" +
          m_Proband.m_pGewicht +";" +m_cGeschlecht +";" +m_Proband.m_pFuss +";" +m_Proband.
          m_pAnamnese +";" +m_Proband.m_pTritt +";" +m_Proband.m_pBeins +";" +m_Proband.m_pBemerkung
           +"\n";
380   //Zeile 5
381   m_ProbTemp += "Messdaten\n";
382   //Zeile 6
383   m_ProbTemp += "Messdatum;␣Bemerkung;␣Fuss\n";
384   m_ProbTemp += m_Proband.m_mDatum +";" +m_Proband.m_mBemerkung +";" +m_cFuss +"\n\n";
385
386
387 }
388
389
390
391 void CUsgDlg::OnProbNew()
392 {
393   // TODO: Code für Befehlsbehandlungsroutine hier einfügen
394   //Hier Funktion einfügen
395   OnNeu();
396
397 }
398
399 void CUsgDlg::OnSettings()
400 {
401   // TODO: Code für Befehlsbehandlungsroutine hier einfügen
402   //Configurations Fenster aufrunfen, in der zebris dll
403   cMessung.zebConf();
404
405 }
406
407 void CUsgDlg::OnContent()
408 {
409   // TODO: Code für Befehlsbehandlungsroutine hier einfügen
410   m_Inhalt.DoModal();
```

```
411
412 }
413
414 void CUsgDlg::OnInfo()
415 {
416   // TODO: Code für Befehlsbehandlungsroutine hier einfügen
417   m_Info.DoModal();
418
419 }
420
421
422
423
424
425
426
427
428 ////////////////////////////////////////////////////////////////////////////
429 //Funktion um die Punkte für die Berechnung des Koordinatensystems einzuziehen
430 void CUsgDlg::OnXyz()
431 {
432   // TODO: Code für Befehlsbehandlungsroutine hier einfügen
433
434   ZebSignal1 = 0; //kein Signal von Zebris Marker 1
435   ZebSignal2 = 0; //kein Signal von Zebris Marker 2
436   ZebRec = 0;
437   ZebSignalOK = 0;//Aufgenommenes Signal OK?!?
438   pNextStep = 0;  //Zeiger um dialoge weiter zuschalten
439
440   //Aufrufen des Dialogs für festlegen Koordinatensystem, FENSTER 1
441   m_Koordinaten1.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal an/aus)
442   m_Koordinaten1.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal an/aus)
443   m_Koordinaten1.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
444   m_Koordinaten1.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
445   m_Koordinaten1.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
446   m_Koordinaten1.pNextStep = &pNextStep;    //Zeiger auf var um Dialoge weiter zu schalten
447
448
449   //STEP 1
450   xyz_step = 1;  //Schritt ein, erster Punkt wird aufgezeichnet
451   m_Koordinaten1.DoModal();
452   ZebSignalOK = 0; //SignalOK zurücksetzen
453   closeDialog = 0;
454   pNextStep = 0;  //Zeiger um dialoge weiter zuschalten
455
456   //STEP 2
457   //Abfrage ob weiter gedrückt, ansonsten abruch
458   if (m_Koordinaten1.m_Cali == 1) {
459     xyz_step = 2; //war einzug step 1 OK ?!? dann step 2
460     //Aufruf des Dialogs, Übergabe der wichtigen Variablen
461     m_Koordinaten2.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal an/aus)
462     m_Koordinaten2.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal an/aus)
463     m_Koordinaten2.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
464     m_Koordinaten2.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
465     m_Koordinaten2.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
466     m_Koordinaten2.pNextStep = &pNextStep;    //Zeiger auf var um Dialoge weiter zu schalten
467     //Aufruf Dialog2
468     m_Koordinaten2.DoModal();
469     ZebSignalOK = 0; //SignalOK zurücksetzen
470     closeDialog = 0; //zurücksetzen
471     pNextStep = 0;  //Zeiger um dialoge weiter zuschalten zurücksetzen
472
473     //STEP 3
474     //Wenn in Dialog2 weiter gedrückt ...
475     if (m_Koordinaten2.m_Cali == 1) {
476       xyz_step = 3; //war einzug step 1 OK ?!? dann step 2
477       //Aufruf des Dialogs, Übergabe der wichtigen Variablen
478       m_Koordinaten3.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal an/aus
              )
479       m_Koordinaten3.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal an/aus
              )
480       m_Koordinaten3.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
481       m_Koordinaten3.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
482       m_Koordinaten3.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
483       m_Koordinaten3.pNextStep = &pNextStep;    //Zeiger auf var um Dialoge weiter zu schalten
484
485       //Aufruf Dialog3
486       m_Koordinaten3.DoModal();
487       ZebSignalOK = 0; //SignalOK zurücksetzen
488       closeDialog = 0; //zurücksetzen
489       pNextStep = 0;  //Zeiger um dialoge weiter zuschalten zurücksetzen
490       //Wenn in Dialog3 weiter gedrückt
491       if (m_Koordinaten3.m_Cali == 1) {
492         xyz_step = 4; //war einzug step 1 OK ?!? dann step 2
```

```
493         //Aufruf des Dialogs, Übergabe der wichtigen Variablen
494         m_Koordinaten4.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal an/
                aus)
495         m_Koordinaten4.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal an/
                aus)
496         m_Koordinaten4.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
497         m_Koordinaten4.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
498         m_Koordinaten4.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
499         m_Koordinaten4.pNextStep = &pNextStep;     //Zeiger auf var um Dialoge weiter zu schalten
500
501         //Dialog4 aufrufen
502         m_Koordinaten4.DoModal();
503         ZebSignalOK = 0; //SignalOK zurücksetzen
504         closeDialog = 0; //zurücksetzen
505         pNextStep = 0;  //Zeiger um dialoge weiter zuschalten zurücksetzen
506
507
508         //Wenn in Dialod4 weiter gedrückt
509         if (m_Koordinaten4.m_Cali == 1) {
510           xyz_step = 5; //war einzug step 4 OK ?!? dann step 5
511           //Aufruf des Dialogs, Übergabe der wichtigen Variablen
512           m_Koordinaten5.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal an
                /aus)
513           m_Koordinaten5.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal an
                /aus)
514           m_Koordinaten5.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
515           m_Koordinaten5.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
516           m_Koordinaten5.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
517           m_Koordinaten5.pNextStep = &pNextStep;     //Zeiger auf var um Dialoge weiter zu
                schalten
518
519           //Dialog5 aufrufen
520           m_Koordinaten5.DoModal();
521           ZebSignalOK = 0; //SignalOK zurücksetzen
522           closeDialog = 0; //zurücksetzen
523           pNextStep = 0;  //Zeiger um dialoge weiter zuschalten zurücksetzen
524
525           //Wenn in Dialog5 weiter gedrückt
526           if (m_Koordinaten5.m_Cali == 1) {
527             xyz_step = 6; //war einzug step 5 OK ?!? dann step 6
528             //Aufruf des Dialogs, Übergabe der wichtigen Variablen
529             m_Koordinaten6.pSignal1 = &ZebSignal1;  //übergibt den zeiger auf ZebSignal1 (Signal
                    an/aus)
530             m_Koordinaten6.pSignal2 = &ZebSignal2;  //übergibt den zeiger auf ZebSignal2 (Signal
                    an/aus)
531             m_Koordinaten6.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
532             m_Koordinaten6.ZebSignalOK = &ZebSignalOK; //Zeiger auf Signal OK
533             m_Koordinaten6.closeDialog = &closeDialog; //Zeiger auf Var closeDialog
534             m_Koordinaten6.pNextStep = &pNextStep;     //Zeiger auf var um Dialoge weiter zu
                    schalten
535
536             //Dialog6 aufrufen
537             m_Koordinaten6.DoModal();
538             ZebSignalOK = 0; //SignalOK zurücksetzen
539             closeDialog = 0; //zurücksetzen
540             pNextStep = 0;  //Zeiger um dialoge weiter zuschalten zurücksetzen
541
542             //Wenn in Dialog6 fertigstellen gedrückt
543             if (m_Koordinaten6.m_Cali == 1) {
544               xyz_step = 7;
545               //Button Messung aktivieren
546               GetDlgItem(IDC_MESSUNG)->EnableWindow(TRUE);
547             }
548           }
549         }
550
551       }
552
553     }
554   }
555   m_zebControl = 0; //Variable für zebris an/aus (Thread wird gestoppt)
556
557   //Rechtes oder Linkes Bein?!?
558   if (m_Proband.m_mFuss == 0)
559   {
560     Achse.Leg = 'L';  //Linkes Bein
561   }
562   else
563   {
564     Achse.Leg = 'R'; //Rechtes Bein
565   }
566   ////////////////////////////////////////////////////////////////////////////////////
567   //Berechne Tibiakoordinatensystem
568   Achse.coordTibia(pmarker1X1, pmarker1Y1, pmarker1Z1, pmarker2X1, pmarker2Y1, pmarker2Z1,
```

```
569             pmarker1X2, pmarker1Y2, pmarker1Z2, pmarker2X2, pmarker2Y2, pmarker2Z2,
570             pmarker1X3, pmarker1Y3, pmarker1Z3, pmarker2X3, pmarker2Y3, pmarker2Z3,
571             pmarker1X6, pmarker1Y6, pmarker1Z6, pmarker2X6, pmarker2Y6, pmarker2Z6);
572   //Berechnen der OSGA aus den Maleollen punkten
573   Achse.meshOSG(pmarker1X4, pmarker1Y4, pmarker1Z4,pmarker2X4, pmarker2Y4, pmarker2Z4,
574         pmarker1X5, pmarker1Y5, pmarker1Z5, pmarker2X5, pmarker2Y5, pmarker2Z5);
575 }
576
577 //
578 ////////////////////////////////////////////////////////////////////////////////////////////////
579
580
581
582 void CUsgDlg::OnKoor()
583 {
584   m_zebControl = 1; //Variable für zebris an/aus
585
586   //Dialog zurücksetzen
587   output2 = 'r';
588   UpdateDialog(output2);
589   output2 = 'i';
590   //Thread starten
591   CWinThread* pThread = AfxBeginThread (thrFunction, this); //Thread
592
593   //Aufruf der Funktion koordinaten festlegen
594   OnXyz();
595
596 }
597
598
599 //Threadfunktion für die Messung
600 UINT CUsgDlg::thrFunction(LPVOID pParam)
601 {
602   CUsgDlg* pDlg = (CUsgDlg*) pParam;
603   pDlg->OnMess();
604
605   return 0;
606 }
607
608 ////////////////////////////////////////
609 // Messung vom Thread gestartet und deswegen auch Thread-> läuft parallel
610 void CUsgDlg::OnMess()
611 {
612   CString sOutStatus;
613   CString sOutX1, sOutY1, sOutZ1;
614   int ON1 = 0;
615   int POS1 = 0;
616   int ON2 = 0;
617   int POS2 = 0;
618
619
620
621   //zebClass cMessung;
622   //Fehler zebClass zurücksetzen
623   cMessung.zebError = false;
624   cMessung.closeDialog = 0;
625   //ZebSignal für initialisierung setzen
626   ZebSignal1 = 3;
627   ZebSignal2 = 3;
628   //zebClass messung;
629   if(zebFirstRun)
630   {
631     //Hardware wird initiiert
632     cMessung.zebHdwInit();
633     if(!cMessung.zebError)
634     {
635       zebFirstRun = false;
636     }
637     else zebFirstRun = true;
638   }
639   if(!cMessung.zebError) {
640     cMessung.zebStart();
641   }
642
643   if (!cMessung.zebError)
644   {
645     while (m_zebControl && !cMessung.closeDialog) {
646
647       //Abfrage ob Hardwarefehler
648       closeDialog = cMessung.closeDialog;
649
650       //wenn Dialog geschlossen wird, soll thread zum einzug von zeb beendet werden
651       if(closeDialog == 1) {
652         m_zebControl = 0; //thread, für dateneinzug
```

```
653         }
654         //keine Messung trozdem Abfrage ob Signal 1 von zebris oder nicht
655         if (cMessung.zebX1)
656         {
657           ZebSignal1 = 1;
658         }
659         else
660         {
661           ZebSignal1 = 0;
662         }
663         //keine Messung trozdem Abfrage ob Signal 2 von zebris oder nicht
664         if (cMessung.zebX2)
665         {
666           ZebSignal2 = 1;
667         }
668         else
669         {
670           ZebSignal2 = 0;
671         }
672
673
674         cMessung.zebGetSingle();
675         ////////////////////////////////////////////////////////
676         //Funktion ZebrisSchalter
677         //Schalter 2, WEITERSCHALTEN!!!!
678         // Schalter 2 gedrückt, ansteigende Flanke, nur wenn vorher ein Signal aufgezeichnet wurde
                (ZebSignalOK==1)
679         if(POS2 == 0 && cMessung.ON2 == 1 && ZebSignalOK == 1)
680         {
681           POS2 = 1;
682           ON2 = 1;
683         }
684         //Schalter 2 losgelassen, abfallende Flanke
685         if(POS2 == 1 && cMessung.ON2 == 0)
686         {
687           POS2 = 0;
688         }
689         // Schalter 2
690         ////////////////////////////////////////////////////////
691
692         if(ON2 ==1)
693         {
694           ON2 =0;
695           pNextStep = 1;              //Dialog wird weiter geschalten
696           ::PlaySound("c:\\windows\\media\\beeb2.wav", NULL, SND_FILENAME | SND_ASYNC );
697         }
698
699         //nur wenn Signale OK kann Signal aufgenommen werden
700         if(ZebSignal1 && ZebSignal2) {
701           //wenn schalter nicht gedrückt-> gedrückt, ansteigende Flanke
702           if(POS1 == 0 && cMessung.ON1 == 1)
703           {
704             POS1 = 1;
705             ON1 = 1;
706           }
707         }
708         //wenn Schalter losgelassen wird, abfallende Flanke
709         if(POS1 == 1 && cMessung.ON1 == 0)
710         {
711           POS1 = 0;
712         }
713
714
715         if(ZebRec || (ON1 == 1))
716         {
717           ON1 = 0;
718           ZebRec = 0; //Variable wieder auf 0 setzen sonst werden ständig werte eingezogen
719           if (cMessung.zebX1) //NOCH EINFÜGEN ABFRAGE OB DIE ANDEREN MESSWERTE AUCH OK SIND!!!!!!
720           {
721             if(xyz_step == 1) //Koordinaten Step 1
722             {
723               pmarker1X1 = cMessung.zebX1;
724               pmarker1Y1 = cMessung.zebY1;
725               pmarker1Z1 = cMessung.zebZ1;
726               pmarker2X1 = cMessung.zebX2;
727               pmarker2Y1 = cMessung.zebY2;
728               pmarker2Z1 = cMessung.zebZ2;
729
730               if(pmarker1X1 && pmarker1Y1 && pmarker1Z1 && pmarker2X1 && pmarker2Y1 && pmarker2Z1)
                    {
731               ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
732               ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
733             }
734           }
```

```
735             if(xyz_step == 2) //Koordinaten Step 2
736             {
737               pmarker1X2 = cMessung.zebX1;
738               pmarker1Y2 = cMessung.zebY1;
739               pmarker1Z2 = cMessung.zebZ1;
740               pmarker2X2 = cMessung.zebX2;
741               pmarker2Y2 = cMessung.zebY2;
742               pmarker2Z2 = cMessung.zebZ2;
743
744               if(pmarker1X2 && pmarker1Y2 && pmarker1Z2 && pmarker2X2 && pmarker2Y2 && pmarker2Z2)
                      {
745                 ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
746                 ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
747               }
748
749             }
750             if(xyz_step == 3) //Koordinaten Step 3
751             {
752               pmarker1X3 = cMessung.zebX1;
753               pmarker1Y3 = cMessung.zebY1;
754               pmarker1Z3 = cMessung.zebZ1;
755               pmarker2X3 = cMessung.zebX2;
756               pmarker2Y3 = cMessung.zebY2;
757               pmarker2Z3 = cMessung.zebZ2;
758
759               if(pmarker1X3 && pmarker1Y3 && pmarker1Z3 && pmarker2X3 && pmarker2Y3 && pmarker2Z3)
                      {
760                 ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
761                 ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
762               }
763
764             }
765             if(xyz_step == 4) //Koordinaten Step 4
766             {
767               pmarker1X4 = cMessung.zebX1;
768               pmarker1Y4 = cMessung.zebY1;
769               pmarker1Z4 = cMessung.zebZ1;
770               pmarker2X4 = cMessung.zebX2;
771               pmarker2Y4 = cMessung.zebY2;
772               pmarker2Z4 = cMessung.zebZ2;
773
774               if(pmarker1X4 && pmarker1Y4 && pmarker1Z4 && pmarker2X4 && pmarker2Y4 && pmarker2Z4)
                      {
775                 ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
776                 ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
777               }
778
779             }
780             if(xyz_step == 5) //Koordinaten Step 5
781             {
782               pmarker1X5 = cMessung.zebX1;
783               pmarker1Y5 = cMessung.zebY1;
784               pmarker1Z5 = cMessung.zebZ1;
785               pmarker2X5 = cMessung.zebX2;
786               pmarker2Y5 = cMessung.zebY2;
787               pmarker2Z5 = cMessung.zebZ2;
788
789               if(pmarker1X5 && pmarker1Y5 && pmarker1Z5 && pmarker2X5 && pmarker2Y5 && pmarker2Z5)
                      {
790                 ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
791                 ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
792               }
793
794             }
795             if(xyz_step == 6) //Koordinaten Step 6
796             {
797               pmarker1X6 = cMessung.zebX1;
798               pmarker1Y6 = cMessung.zebY1;
799               pmarker1Z6 = cMessung.zebZ1;
800               pmarker2X6 = cMessung.zebX2;
801               pmarker2Y6 = cMessung.zebY2;
802               pmarker2Z6 = cMessung.zebZ2;
803
804               if(pmarker1X6 && pmarker1Y6 && pmarker1Z6 && pmarker2X6 && pmarker2Y6 && pmarker2Z6)
                      {
805                 ZebSignalOK = 1;     //Aufgenommenes Signal OK? Dann weiter...
806                 ::PlaySound("c:\\windows\\media\\beeb.wav", NULL, SND_FILENAME | SND_ASYNC );
807               }
808
809             }
810           }
811         }
812         Sleep(1);
813
```

```
814
815       }
816
817
818
819       cMessung.zebStop();
820
821       sOutStatus = cMessung.zebStatus;
822     }
823     //Abfrage ob Hardwarefehler
824     closeDialog = cMessung.closeDialog;
825   }
826   //
827   ////////////////////////////////////////////////////
828
829
830
831
832
833
834
835
836   ////////////////////////////////////////////////////////////////////////////////////////////////
837   //Start Messung
838   void CUsgDlg::OnMessung()
839   {
840     // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
841     //Thread starten für die Messung
842     m_zebMessControl = 1; //Variable für zebris an/aus
843     CWinThread* pThread = AfxBeginThread (thrFunction2, this); //Thread
844
845     OnMessAblauf();    //Aufrus der Funktion die den Messablauf macht
846
847   }
848
849   //Thread Starten
850   UINT CUsgDlg::thrFunction2(LPVOID pParam)
851   {
852     CUsgDlg* pDlg = (CUsgDlg*) pParam;
853     pDlg->OnMAxis();
854
855     return 0;
856   }
857
858
859   ////////////////////////////////////////
860   //   MESSUNG OSGA USGA
861   // Thread Funktion
862   void CUsgDlg::OnMAxis()
863   {
864
865     pos_osg_X.RemoveAll();
866     pos_osg_Y.RemoveAll();
867     pos_osg_Z.RemoveAll();
868     pos_usg_X.RemoveAll();
869     pos_usg_Y.RemoveAll();
870     pos_usg_Z.RemoveAll();
871     //Eigentliche Threadfunktion
872     //Zebris Starten
873     CString sOutStatus;
874
875     int ON1 = 0;
876     int POS1 = 0;
877     int ON2 = 0;
878     int POS2 = 0;
879
880     cMessung.zebStart();          //Zebris starten
881     sOutStatus = cMessung.zebStatus;  //Status auslesen
882
883
884     while (m_zebMessControl) {
885     //keine Messung trozdem Abfrage ob Signal 3 von zebris oder nicht
886       if (cMessung.zebX3 && cMessung.zebY3 && cMessung.zebZ3)
887       {
888         ZebSignal3 = 1;
889       }
890       else
891       {
892         ZebSignal3 = 0;
893       }
894       //keine Messung trozdem Abfrage ob Signal 4 von zebris oder nicht
895       if (cMessung.zebX4 && cMessung.zebY4 && cMessung.zebZ4)
896       {
897         ZebSignal4 = 1;
```

```
898     }
899     else
900     {
901       ZebSignal4 = 0;
902     }
903
904
905     cMessung.zebGetSingle();
906
907       ////////////////////////////////////////////////////
908       //Funktion ZebrisSchalter
909       //Schalter 2, WEITERSCHALTEN!!!!
910       // Schalter 2 gedrückt, ansteigende Flanke, nur wenn vorher ein Signal aufgezeichnet wurde
              (ZebSignalOK==1)
911       if(POS2 == 0 && cMessung.ON2 == 1 && (ZebRec == 2 || ZebRec == 4))
912       {
913         POS2 = 1;
914         ON2 = 1;
915         pPressNext = 1; //bewirkt ein Weiterschalten
916
917       }
918       //Schalter 2 losgelassen, abfallende Flanke
919       if(POS2 == 1 && cMessung.ON2 == 0)
920       {
921         POS2 = 0;
922       }
923       // Schalter 2
924       ////////////////////////////////////////////////////
925
926       if(ON2 ==1)
927       {
928         ON2 =0;
929         pNextStep = 1;              //Dialog wird weiter geschalten
930         //::PlaySound("c:\\windows\\media\\beeb2.wav", NULL, SND_FILENAME | SND_ASYNC );
931       }
932
933       //Überprüfung Signal OK?!?
934       if(ZebRec == 0 && ZebSignal3 == 1 || ZebRec == 1 && ZebSignal3 == 1 || ZebRec == 2 &&
              ZebSignal4 == 1 || ZebRec == 3 && ZebSignal4 == 1)
935       {
936         //wenn schalter nicht gedrückt-> gedrückt, ansteigende Flanke
937         if(POS1 == 0 && cMessung.ON1 == 1)
938         {
939           POS1 = 1;
940           ON1 = 1;
941           pPressRec = 1;
942         }
943       }
944
945       //wenn Schalter losgelassen wird, abfallende Flanke
946       if(POS1 == 1 && cMessung.ON1 == 0)
947       {
948         POS1 = 0;
949       }
950
951     //Aufnahme Starten für Messung OSGA
952     if(ZebRec == 1)
953     {
954       if (cMessung.zebX3 && cMessung.zebY3 && cMessung.zebZ3) //Signal OK?!?
955       {
956         pos_osg_X.Add(cMessung.zebX3);
957         pos_osg_Y.Add(cMessung.zebY3);
958         pos_osg_Z.Add(cMessung.zebZ3);
959       }
960     }
961
962     //Aufnahme Starten für Messung USGA
963     if(ZebRec == 3)
964     {
965       if (cMessung.zebX4 && cMessung.zebY4 && cMessung.zebZ4) // Signal OK?!?
966       {
967         pos_usg_X.Add(cMessung.zebX4);
968         pos_usg_Y.Add(cMessung.zebY4);
969         pos_usg_Z.Add(cMessung.zebZ4);
970       }
971     }
972     Sleep(1);
973
974   }
975   cMessung.zebStop();
976
977   sOutStatus = cMessung.zebStatus;
978
979 }// Thread () ENDE
```

```
980 //////////////////////
981
982
983 ////////////////////////////////////////////////////////////////////////////////////////////////
984 //Ablauf der Messung der Achsen
985 void CUsgDlg::OnMessAblauf()
986 {
987   stepThread = 1; //1 bedeutet Tread durchläuft "if" Abfrage OSGA, 2 -> USGA
988   ZebSignal3 = 0; //kein Signal von Zebris Marker 3
989   ZebSignal4 = 0; //kein Signal von Zebris Marker 4
990   ZebRec = 0;
991   ZebRecStep = 1;
992   //Variable setzen für rechtes oder linkes bein
993   if (m_Proband.m_mFuss == 0)
994   {
995     Achse.Leg = 'L';  //Linkes Bein
996   }
997   else
998   {
999     Achse.Leg = 'R'; //Rechtes Bein
1000  }
1001  //Setzen Verschiedener Variablen, reservieren des Speicherbereichs
1002  m_MessWindow1.pSignal3 = &ZebSignal3; //übergibt den zeiger auf ZebSignal3 (Signal an/aus)
1003  m_MessWindow1.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
1004  m_MessWindow1.pZebRecStep = &ZebRecStep; //Zeiger Schritt aufnahme
1005  m_MessWindow1.closeDialog = &closeDialog; //zeiger auf die variable close dialog (fehler
       abfangen)
1006  m_MessWindow1.pPressRec = &pPressRec;    //Zeiger für Record Button im Aufnahmefenster Messung
1007  m_MessWindow1.pPressNext = &pPressNext;    //Zeiger für Weiter Button
1008  m_MessWindow1.DoModal(); //Öffnet das erste Messfenster
1009
1010  if(ZebRecStep == 2) {
1011    stepThread = 2; //Thread durchläuft die "if" schleife USGA
1012    m_MessWindow2.pSignal4 = &ZebSignal4; //übergibt den zeiger auf ZebSignal3 (Signal an/aus)
1013    m_MessWindow2.pZebRec = &ZebRec; //Zeiger Aufnahmebutton gedrückt
1014    m_MessWindow2.pZebRecStep = &ZebRecStep; //Zeiger Schritt aufnahme
1015    m_MessWindow2.closeDialog = &closeDialog; //zeiger auf die variable close dialog (fehler
         abfangen)
1016    m_MessWindow2.pPressRec = &pPressRec;    //Zeiger für Record Button im Aufnahmefenster Messung
1017    m_MessWindow2.pPressNext = &pPressNext;    //Zeiger für Weiter Button
1018    m_MessWindow2.DoModal();  //Öffnet das 2. Messfenster
1019  }
1020  m_zebMessControl = 0; //Thread Zebris 2 wird gestoppt, Zebris abgeschalten
1021
1022  int test =1;
1023  int test2 =2;
1024  //wenn messung erfolgreich berechnung
1025  if(ZebRecStep == 3) {
1026    //Aufruf der Funktion zum Ermitteln der Punkte zur Berechnung der Achsen
1027    OnCalculate();
1028  }
1029  else MessageBox("Messung␣war␣nicht␣erfolgreich,␣nicht␣alle␣benötigten␣Daten␣wurden␣eingezogen!\
       r\nBerechnung␣der␣Achsen␣wird␣nicht␣durchgeführt!","Messfehler",MB_ICONERROR+
       MB_SETFOREGROUND+MB_OK);
1030
1031
1032 }// Messablauf() ENDE
1033 ////////////////////////////////////////////////////////////////////////////
1034 ////////////////////////////////////////////////////////////////////////////////////
1035
1036
1037 ////////////////////////////////////////////////////////////////////////////////
1038 ////////////////////////////////////////////////////////////////////////////
1039 ////////////////////////////////// F I L T E R ////////////////////////////////
1040 //Filter durchlaufen
1041 void CUsgDlg::callFilter()
1042 {
1043   fpos_osg_X.RemoveAll();
1044   fpos_osg_Y.RemoveAll();
1045   fpos_osg_Z.RemoveAll();
1046   fpos_usg_X.RemoveAll();
1047   fpos_usg_Y.RemoveAll();
1048   fpos_usg_Z.RemoveAll();
1049   int ndata, i;
1050   Fc=1;          //Grenzfrequenz
1051   order=4;          //Pole
1052   dt = 0.01333333333;        //Periodendauer
1053   //Butterworthkoeffizienten berechnen
1054   Butterworth(Fc,dt,order,C,&NSections,&Tg);
1055
1056   //Fiter X Werte USGA
1057   //Filterinitialisieren
1058   Xdc = pos_usg_X[1];    //Startsignal festlegen
1059   FilterInit(Xdc,C,NSections,D);
```

```
1060    //Grösse SignalArray
1061    ndata = pos_usg_X.GetSize();
1062    for (i=0; i<ndata; i++) {
1063      Filter(&temp,pos_usg_X[i],NSections,C,D);
1064      fpos_usg_X.Add(temp);
1065    }
1066
1067    //Filter Y Werte USGA
1068    //Filterinitialisieren
1069    Xdc = pos_usg_Y[1];    //Startsignal festlegen
1070    FilterInit(Xdc,C,NSections,D);
1071    //Grösse SignalArray
1072    ndata = pos_usg_Y.GetSize();
1073    for (i=0; i<ndata; i++) {
1074      Filter(&temp,pos_usg_Y[i],NSections,C,D);
1075      fpos_usg_Y.Add(temp);
1076    }
1077
1078    //Filter Z Werte USGA
1079    //Filterinitialisieren
1080    Xdc = pos_usg_Z[1];    //Startsignal festlegen
1081    FilterInit(Xdc,C,NSections,D);
1082    //Grösse SignalArray
1083    ndata = pos_usg_Z.GetSize();
1084    for (i=0; i<ndata; i++) {
1085      Filter(&temp,pos_usg_Z[i],NSections,C,D);
1086      fpos_usg_Z.Add(temp);
1087    }
1088
1089    ////////////////////////
1090    //FILTERN DER OSG WERTE
1091    //Fiter X Werte OSGA
1092    //Filterinitialisieren
1093    Xdc = pos_osg_X[1];    //Startsignal festlegen
1094    FilterInit(Xdc,C,NSections,D);
1095    //Grösse SignalArray
1096    ndata = pos_osg_X.GetSize();
1097    for (i=0; i<ndata; i++) {
1098      Filter(&temp,pos_osg_X[i],NSections,C,D);
1099      fpos_osg_X.Add(temp);
1100    }
1101
1102    //Filter Y Werte USGA
1103    //Filterinitialisieren
1104    Xdc = pos_osg_Y[1];    //Startsignal festlegen
1105    FilterInit(Xdc,C,NSections,D);
1106    //Grösse SignalArray
1107    ndata = pos_osg_Y.GetSize();
1108    for (i=0; i<ndata; i++) {
1109      Filter(&temp,pos_osg_Y[i],NSections,C,D);
1110      fpos_osg_Y.Add(temp);
1111    }
1112
1113    //Filter Z Werte USGA
1114    //Filterinitialisieren
1115    Xdc = pos_osg_Z[1];    //Startsignal festlegen
1116    FilterInit(Xdc,C,NSections,D);
1117    //Grösse SignalArray
1118    ndata = pos_osg_Z.GetSize();
1119    for (i=0; i<ndata; i++) {
1120      Filter(&temp,pos_osg_Z[i],NSections,C,D);
1121      fpos_osg_Z.Add(temp);
1122    }
1123  }
1124
1125  ////////////////////////////////////////////////
1126  // Butterworth Filter Koeffizienten berechnen
1127  // Fc: Grenzfrequenz, Ts: Periadendauer, n: Pole (1-4)
1128  void CUsgDlg::Butterworth(double Fc,double Ts,int n,Filter_Coef C, int *NSections, double *Tg )
1129  {
1130    int Ns2,i,Modn;
1131    double Arg,Rep,Omega,OmegaSq,temp,W0,W1,m;
1132    double Zero,ONE,TWO,HALF,Pi;
1133
1134    Zero = 0;
1135    ONE  = 1;
1136    TWO  = 2;
1137    HALF = 0.5;
1138    Pi   = 3.1415926535;
1139
1140    Arg=Pi*Ts*Fc;
1141    if (fabs(Arg) > 2.0*Pi) {
1142      m=int(Arg/2.0/Pi);
1143      Arg=Arg-(m*2.0*Pi);
```

```
1144    }
1145    Omega= tan(Arg);
1146    OmegaSq=Omega*Omega;
1147    Modn=(n % 2);
1148    if (Modn==0)
1149    temp=HALF;
1150    else
1151    temp=Zero;
1152    Ns2=n/2;
1153    *NSections=Ns2+Modn;
1154    *Tg=Zero;
1155    if (n>1)
1156      for (i=1; i<Ns2+1; i++) {
1157        Rep=Omega*cos(Pi*(i-temp)/n);
1158        *Tg=*Tg+Ts*Rep/OmegaSq;
1159        W0=TWO*Rep;
1160        W1=ONE +W0+OmegaSq;
1161        C[1][i]=-TWO*(OmegaSq-ONE)/W1;
1162        C[2][i]=-(ONE-W0+OmegaSq)/W1;
1163        C[3][i]=TWO;
1164        C[4][i]=ONE;
1165        C[5][i]=OmegaSq/W1;
1166      }
1167      if (temp == Zero) {
1168        C[1][*NSections]=(ONE-Omega)/(ONE+Omega);
1169        C[2][*NSections]= Zero;
1170        C[3][*NSections]= ONE;
1171        C[4][*NSections]= Zero;
1172        C[5][*NSections]= Omega/(ONE+Omega);
1173        *Tg= *Tg+Ts/(TWO*Omega);
1174      }
1175
1176 }//Butterworth()
1177
1178 ////////////////////////////////////////
1179 //Filter initiieren
1180 // Xdc: Startfrequenz festlegen,
1181 void CUsgDlg::FilterInit(double Xdc, Filter_Coef C, int NSections, Memory_Coef D)
1182 {
1183   double dc,Csum;
1184   int i,j;
1185   dc=Xdc;
1186   for (j=1; j<NSections+1; j++) {
1187     D[2][j]=dc/(1-C[1][j]-C[2][j]);
1188     D[1][j]=D[2][j];
1189     Csum=0;
1190     for (i=1; i<5; i++) Csum=Csum + C[i][j];
1191     dc=C[5][j]*(dc+D[2][j]*Csum);
1192   } //j loop
1193 }//FilterInit()
1194
1195
1196 ////////////////////////////////////////////////////////
1197 //FilterFunktion
1198 void CUsgDlg::Filter(double *Xs,double Xd, int NSections, Filter_Coef C, Memory_Coef D)
1199 {
1200   double x,y,err;
1201   int i;
1202   x=Xd;
1203   for (i=1; i<NSections+1; i++) {
1204     err=x+C[1][i]*D[1][i]+C[2][i]*D[2][i];
1205     y=C[5][i]*(err +C[3][i]*D[1][i]+C[4][i]*D[2][i]);
1206     D[2][i]=D[1][i];
1207     D[1][i]=err;
1208     x=y;
1209   }
1210   *Xs=x;
1211 }//Filter()
1212
1213 // Ende Filterfunktionen
1214 ////////////////////////////////////////////////////////////////
1215 ////////////////////////////////////////////////////////////////
1216 ////////////////////////////////////////////////////////////////
1217
1218
1219
1220
1221 ////////////////////////////////////////////////////////////////////////////////
1222 //
1223 //         Berechnungs routinen
1224 //
1225 ////////////////////////////////////////////////////////////////////////////////
1226
1227 ////////////////////////////////////////////////////////
```

```
1228 // Berechnung mehrerer Achsen,jeweils aus den Umkehrpunkten und dem Mittelpunkt
1229 void CUsgDlg::OnCalcAxis2()
1230 {
1231   ////////////////////////////////////////
1232   //                                    //
1233   //        punkte suchen               //
1234   //                                    //
1235   ////////////////////////////////////////
1236
1237   //Arrays für die Suche der Berechnungspunkt
1238   CArray<double, double>punkt1x;
1239   CArray<double, double>punkt1y;
1240   CArray<double, double>punkt1z;
1241   CArray<double, double>punkt2x;
1242   CArray<double, double>punkt2y;
1243   CArray<double, double>punkt2z;
1244   CArray<double, double>punkt3x;
1245   CArray<double, double>punkt3y;
1246   CArray<double, double>punkt3z;
1247   CArray<int, int>startP;
1248   CArray<int, int>endP;
1249
1250   //Variable für Bewegungsrichtung
1251   CString richtung;
1252   int iMove,changeDir,i;
1253   double xVector;
1254   double ab_min;  //mindestabstand von p1 zu p3
1255   double ab_temp;
1256   double ab_max_temp = 0;
1257
1258   //Bewegungsrichtung, positiv-> von - nach +
1259   //Bewegungsrichtung, negativ-> von + nach -
1260   if (Achse.Leg == 'R')
1261   {
1262     iMove = 1; //bei rechtem Bein, geht die Bewegung von lateral nach medial -> -x nach +x ->
          Bewegungsrichtung positiv
1263   }
1264   else iMove = -1;     //linkes BEIN!
1265
1266   //Array löschen
1267   usg_INC.RemoveAll();  //Inlinationswinkel
1268   usg_DEV.RemoveAll();  //Deviationswinkel
1269   usg_AP_X.RemoveAll(); //Aufpunkt X
1270   usg_AP_Y.RemoveAll(); //Aufpunkt Y
1271   usg_AP_Z.RemoveAll(); //Aufpunkt Z
1272   usg_RV_X.RemoveAll(); //richtungsvektor X
1273   usg_RV_Y.RemoveAll(); //Richtungsvektor Y
1274   usg_RV_Z.RemoveAll(); //Richtungsvektor Z
1275   usg_LEVER.RemoveAll();  //Hebel Achilles
1276   usg_INC1.RemoveAll(); //Inlinationswinkel
1277   usg_DEV1.RemoveAll(); //Deviationswinkel
1278   usg_AP_X1.RemoveAll();  //Aufpunkt X
1279   usg_AP_Y1.RemoveAll();  //Aufpunkt Y
1280   usg_AP_Z1.RemoveAll();  //Aufpunkt Z
1281   usg_RV_X1.RemoveAll();  //richtungsvektor X
1282   usg_RV_Y1.RemoveAll();  //Richtungsvektor Y
1283   usg_RV_Z1.RemoveAll();  //Richtungsvektor Z
1284   usg_LEVER1.RemoveAll(); //Hebel Achilles
1285   usg_DIST1.RemoveAll();  //Abstand OSGA - USGA
1286
1287   //String für Augabe
1288   //sAxis2 = "USGA um OSGA berechnet;;;;;;;;;;USGA um OSGA (ML_MM)\r\n";
1289   //sAxis2 += "Inc.;Dev.;Lever;AP(X);AP(Y);AP(Z);RV(X);RV(Y);RV(Z);;;Inc.;Dev.;Lever;AP(X);AP(Y);
          AP(Z);RV(X);RV(Y);RV(Z)\r\n";
1290
1291   //Arrayeinträge USG Daten
1292   int size_usg = fpos_usg_X.GetSize();
1293   int n = 1;
1294   int vec_n;
1295   int reftime=50;
1296   int count_ref;
1297   UX_VECTOR3 temp_v;
1298   UX_VECTOR3 temp_P1;
1299   UX_VECTOR3 temp_P2;
1300   UX_VECTOR3 temp_P3;
1301   UX_VECTOR3 temp_P1P2;
1302   UX_VECTOR3 temp_P2P3;
1303   UX_VECTOR3 temp_P1P3;
1304
1305   //mindestabstand von min zu max, wobei der abstand mindestens 1/3 des start und endwertes sein
          muss
1306   temp_P1.X=fpos_usg_X[1];
1307   temp_P1.Y=fpos_usg_Y[1];
1308   temp_P1.Z=fpos_usg_Z[1];
```

```
1309    temp_P3.X=fpos_usg_X[size_usg-1];
1310    temp_P3.Y=fpos_usg_Y[size_usg-1];
1311    temp_P3.Z=fpos_usg_Z[size_usg-1];
1312    temp_P1P3 = temp_v.Subtract(temp_P1, temp_P3);
1313    ab_min = temp_P1P3.Magnitude()/3;
1314
1315
1316    //start, p1 und p3 suchen, sortierung egal
1317    for(i=1; i<size_usg; i++) {
1318      //Punkt 1 wird zugewiesen
1319      if(n==1)
1320      {
1321        //dem Array P1 zuordnen
1322        punkt1x.Add(fpos_usg_X[i]);
1323        punkt1y.Add(fpos_usg_Y[i]);
1324        punkt1z.Add(fpos_usg_Z[i]);
1325        //temporären vektor zuordnen
1326        temp_P1.X=fpos_usg_X[i];
1327        temp_P1.Y=fpos_usg_Y[i];
1328        temp_P1.Z=fpos_usg_Z[i];
1329        //Zeilennummer/arraynummer Punkt1
1330        startP.Add(i);
1331
1332        n=2;
1333      }
1334      //Punkt2 wird gesucht
1335      if(n==2)
1336      {
1337        //temporären Vektor zuordnen
1338        temp_P3.X=fpos_usg_X[i];
1339        temp_P3.Y=fpos_usg_Y[i];
1340        temp_P3.Z=fpos_usg_Z[i];
1341        //Abstand/Vektor P1P2 berechnen
1342        temp_P1P3 = temp_v.Subtract(temp_P1, temp_P3);
1343        ab_temp = temp_P1P3.Magnitude();
1344
1345        //Wenn Abstand vom P1 zu P2 größer als der des vorherigen und größer als der mindestabstand
1346        if(ab_temp>=ab_min && ab_temp>=ab_max_temp)
1347        {
1348          ab_max_temp = ab_temp;  //neuer maximal abstand
1349          vec_n = i;          //Arraynummer (eintrag) gespeichert
1350          count_ref=0;        //Zähler refraktär auf 0 setzen
1351        }
1352        //wenn der abstand kleiner als ab_max_temp (umkehrpunkt?!?!)
1353        else if(ab_temp>=ab_min)
1354        {
1355          //durchzählen wieviele messwerte hintereinader kleiner sind
1356          if(count_ref<reftime)
1357          {
1358            count_ref++;
1359          }
1360          //bei mehr als reftime (50 werte)
1361          else
1362          {
1363            //vorher gefunde maxilamabstand/Punkt wird in Array geschrieben
1364            punkt3x.Add(fpos_usg_X[vec_n]);
1365            punkt3y.Add(fpos_usg_Y[vec_n]);
1366            punkt3z.Add(fpos_usg_Z[vec_n]);
1367
1368            //Zeilennummer/Arraynummer Ende
1369            endP.Add(vec_n);
1370
1371            ab_max_temp=0;  //zurücksetzen
1372            count_ref=0;  //zurücksetzen refraktärzeit
1373            n=1;        //erster Punkt suchen/greifen
1374            i=vec_n;     //weitermachen ab diesem Punkt
1375          }
1376        }
1377      }//ende if(n=2)
1378
1379    }//ende for schleife Punkte P1 P3 suchen
1380    ////////////////////////////////////////////////////////////////////
1381
1382    ////////////////////////////////////////////////////////////////////
1383    //Mittelpunkt suchen P2                                           //
1384    ////////////////////////////////////////////////////////////////////
1385    //Länge Array Punkt3x
1386    int sizePoint = endP.GetSize();
1387    int j, countArray;
1388    int arrayStart, arrayEnd;
1389    double ab_max, ab_p1p2, ab_p2p3;
1390    //schleife alle arrayeinträge durchlaufen
1391    for(i=0;i<sizePoint;i++)
1392    {
```

```
1393      //start und endpunkt datensatz
1394      arrayStart=startP[i];
1395      arrayEnd=endP[i];
1396
1397      //Vektor P1P2, P2P3
1398      temp_P1.X = fpos_usg_X[arrayStart];
1399      temp_P1.Y = fpos_usg_Y[arrayStart];
1400      temp_P1.Z = fpos_usg_Z[arrayStart];
1401      temp_P3.X = fpos_usg_X[arrayEnd];
1402      temp_P3.Y = fpos_usg_Y[arrayEnd];
1403      temp_P3.Z = fpos_usg_Z[arrayEnd];
1404
1405      temp_P1P3 = temp_v.Subtract(temp_P1, temp_P3);  //Vektor P1P3
1406      ab_min = temp_P1P3.Magnitude()/3;        //Länge Vektor
1407      ab_max = 0;
1408      ab_max_temp = 0;
1409      //schleife punkte p1-p3 mittelpunkt suchen
1410      for(j=arrayStart; j<arrayEnd; j++)
1411      {
1412        //Vektor P1P2
1413        temp_P1P2.X = fpos_usg_X[j] - fpos_usg_X[arrayStart];
1414        temp_P1P2.Y = fpos_usg_Y[j] - fpos_usg_Y[arrayStart];
1415        temp_P1P2.Z = fpos_usg_Z[j] - fpos_usg_Z[arrayStart];
1416        //Vektor P2P3
1417        temp_P2P3.X = fpos_usg_X[arrayEnd] - fpos_usg_X[j];
1418        temp_P2P3.Y = fpos_usg_Y[arrayEnd] - fpos_usg_Y[j];
1419        temp_P2P3.Z = fpos_usg_Z[arrayEnd] - fpos_usg_Z[j];
1420
1421        ab_p1p2 = temp_P1P2.Magnitude();     //Abstand P1P2
1422        ab_p2p3 = temp_P2P3.Magnitude();     //Abstand P2P3
1423
1424        ab_max_temp = ab_p1p2 + ab_p2p3;     //Abstand gesamt
1425
1426        //wert gefunden zuweisen
1427        //ab_max > vorher gefundener wer, vektor p1p2 und vektor p2p3 sollen in der mitte liegen
1428        if(ab_max_temp >= ab_max && ab_p1p2 > ab_min && ab_p2p3 > ab_min)
1429        {
1430          //neuer maximal wert
1431          ab_max = ab_max_temp;
1432
1433          //Arraynr zuweisen
1434          countArray = j;
1435        }
1436
1437      }//end abschnitte der Daten durchlaufen
1438
1439      //Punkt2 in Array schreiben
1440      punkt2x.Add(fpos_usg_X[countArray]);
1441      punkt2y.Add(fpos_usg_Y[countArray]);
1442      punkt2z.Add(fpos_usg_Z[countArray]);
1443    }//end forschleife arrayeinträge
1444    ////////////////////////////////////////////////////////////////////
1445    //              Ende Punkt P2 suchen                              //
1446    ////////////////////////////////////////////////////////////////////
1447
1448    //Berechnen der Achsen
1449    //Vektoren für die punkte
1450    UX_VECTOR3 tempP1;
1451    UX_VECTOR3 tempP2;
1452    UX_VECTOR3 tempP3;
1453
1454    CArray<double, double>P1X;
1455    CArray<double, double>P1Y;
1456    CArray<double, double>P1Z;
1457    CArray<double, double>P2X;
1458    CArray<double, double>P2Y;
1459    CArray<double, double>P2Z;
1460    CArray<double, double>P3X;
1461    CArray<double, double>P3Y;
1462    CArray<double, double>P3Z;
1463    CArray<int, int>iD;
1464    //Achsarrays
1465    CArray<double, double>usgINC1;
1466    CArray<double, double>usgINC2;
1467    CArray<double, double>usgDEV1;
1468    CArray<double, double>usgDEV2;
1469    CArray<double, double>usgLEVER1;
1470    CArray<double, double>usgLEVER2;
1471    CArray<double, double>usgDIST1;
1472    CArray<double, double>usgANG1;
1473    CArray<double, double>usgANG2;
1474    CArray<double, double>usgAPX1;
1475    CArray<double, double>usgAPX2;
1476    CArray<double, double>usgAPY1;
```

```
1477    CArray<double, double>usgAPY2;
1478    CArray<double, double>usgAPZ1;
1479    CArray<double, double>usgAPZ2;
1480    CArray<double, double>usgRVX1;
1481    CArray<double, double>usgRVX2;
1482    CArray<double, double>usgRVY1;
1483    CArray<double, double>usgRVY2;
1484    CArray<double, double>usgRVZ1;
1485    CArray<double, double>usgRVZ2;
1486    CArray<int, int>usgERROR1;
1487    CArray<int, int>usgERROR2;
1488    double  mINC1, mINC2, mDEV1, mDEV2, mLEVER1, mLEVER2, mDIST1, mANG1, mANG2,
1489        mAPX1, mAPX2, mAPY1, mAPY2, mAPZ1, mAPZ2,
1490        mRVX1, mRVX2, mRVY1, mRVY2, mRVZ1, mRVZ2;
1491    //0 setzen
1492    mINC1=0; mINC2=0; mDEV1=0; mDEV2=0; mLEVER1=0; mLEVER2=0; mDIST1=0; mANG1=0; mANG2=0;
1493    mAPX1=0; mAPX2=0; mAPY1=0; mAPY2=0; mAPZ1=0; mAPZ2=0;
1494    mRVX1=0; mRVX2=0; mRVY1=0; mRVY1=0; mRVZ1=0; mRVZ2=0;
1495
1496    //String für Augabe
1497    sAxis2 = "USGA␣um␣OSGA␣(MLMM)\r\n";//;;;;;;;;;;USGA um OSGA (berechnet)\r\n";
1498    sAxis2 += "Inc.;Dev.;Lever;Drehwinkel;Abstand␣OSGA/USGA;AP(X);AP(Y);AP(Z);RV(X);RV(Y);RV(Z);;\r
          \n";//Inc.;Dev.;Lever;Drehwinkel;AP(X);AP(Y);AP(Z);RV(X);RV(Y);RV(Z)\r\n";
1499
1500    ////////////////////////////////////////////////////////////////////
1501    //Punkte sortieren bei rechtem bein p1<p2<p3, linkes bein p1>p2>p3 //
1502    // Achsen berechnen                                              //
1503    ////////////////////////////////////////////////////////////////////
1504    for(j=0; j<sizePoint; j++)
1505    {
1506        //rechtes bein: p1<p3 alles ok-> zuweisen
1507        //linkes bein: p1>p3 alles ok-> zuweisen
1508        if((punkt1x[j]<punkt3x[j] && Achse.Leg=='R') || (punkt1x[j]>punkt3x[j] && Achse.Leg=='L'))
1509        {
1510          P1X.Add(punkt1x[j]);     //Punkt1 zuweisen
1511          P1Y.Add(punkt1y[j]);
1512          P1Z.Add(punkt1z[j]);
1513          P2X.Add(punkt2x[j]);     //Punkt2 zuweisen
1514          P2Y.Add(punkt2y[j]);
1515          P2Z.Add(punkt2z[j]);
1516          P3X.Add(punkt3x[j]);     //Punkt3 zuweisen
1517          P3Y.Add(punkt3y[j]);
1518          P3Z.Add(punkt3z[j]);
1519
1520          //richtung angeben
1521          if(Achse.Leg=='R') iD.Add(1);      //Inversion
1522          else if(Achse.Leg=='L') iD.Add(1);     //Inversion
1523        }
1524        //rechtes bein: p1>p3 -> vertauschen
1525        //linkes bein: p1<p3 -> vertauschen
1526        else if((punkt1x[j]>punkt3x[j] && Achse.Leg=='R') || (punkt1x[j]<punkt3x[j] && Achse.Leg=='
              L'))
1527        {
1528          P1X.Add(punkt3x[j]);     //Punkt1 zuweisen
1529          P1Y.Add(punkt3y[j]);
1530          P1Z.Add(punkt3z[j]);
1531          P2X.Add(punkt2x[j]);     //Punkt2 zuweisen
1532          P2Y.Add(punkt2y[j]);
1533          P2Z.Add(punkt2z[j]);
1534          P3X.Add(punkt1x[j]);     //Punkt3 zuweisen
1535          P3Y.Add(punkt1y[j]);
1536          P3Z.Add(punkt1z[j]);
1537
1538          //richtung angeben
1539          if(Achse.Leg=='R') iD.Add(-1);       //Eversion
1540          else if(Achse.Leg=='L') iD.Add(-1);    //Eversion
1541        }
1542
1543      //Variable für Fehler auf 0 setzen
1544      usgERROR1.Add(0);
1545      usgERROR2.Add(0);
1546
1547      //Berechnung USGA
1548      Achse.calcUSG(P1X[j],P1Y[j],P1Z[j],P2X[j],P2Y[j],P2Z[j],P3X[j],P3Y[j],P3Z[j]);
1549
1550      //Zuweisung der Werte in Array
1551      //Um die MalleolenAchse
1552      usgINC1.Add(Achse.USG_inc1);     //Inclinationwinkel
1553      usgDEV1.Add(Achse.USG_dev1);     //Deviationswinkel
1554      usgANG1.Add(Achse.USG_winkel);     //Drehwinkel
1555      usgLEVER1.Add(Achse.lever_USG_1); //Hebel
1556      usgDIST1.Add(Achse.Dist);      //Abstand OSGA/USGA
1557      usgAPX1.Add(Achse.USG_APX1);     //Aufpunkt X
1558      usgAPY1.Add(Achse.USG_APY1);     //Aufpunkt Y
```

```
1559      usgAPZ1.Add(Achse.USG_APZ1);      //Aufpunkt Z
1560      usgRVX1.Add(Achse.USG_RVX1);      //Richtungsvektor X
1561      usgRVY1.Add(Achse.USG_RVY1);      //Richtungsvektor Y
1562      usgRVZ1.Add(Achse.USG_RVZ1);      //Richtungsvektor Z
1563
1564
1565      //Um die gemessene OSGA
1566      usgINC2.Add(Achse.USG_Inc);       //Inclinationswinkel
1567      usgDEV2.Add(Achse.USG_Dev);       //Deviationswinkel
1568      usgANG2.Add(Achse.USG_winkel);     //Dregwinkel
1569      usgLEVER2.Add(Achse.Lever_USGA);   //Hebel
1570      usgAPX2.Add(Achse.USG_APX);       //Aufpunkt X
1571      usgAPY2.Add(Achse.USG_APY);       //Aufpunkt Y
1572      usgAPZ2.Add(Achse.USG_APZ);       //Aufpunkt Z
1573      usgRVX2.Add(Achse.USG_RVX);       //Richtungsvektor X
1574      usgRVY2.Add(Achse.USG_RVY);       //Richtungsvektor Y
1575      usgRVZ2.Add(Achse.USG_RVZ);       //Richtungsvektor Z
1576
1577      //Mittelwertberechnung, Summe
1578      //Malleolenachse
1579      mINC1 = mINC1 + usgINC1[j];
1580      mDEV1 = mDEV1 + usgDEV1[j];
1581      mANG1 = mANG1 + usgANG1[j];
1582      mLEVER1 = mLEVER1 + usgLEVER1[j];
1583      mDIST1 = mDIST1 + usgDIST1[j];
1584      mAPX1 = mAPX1 + usgAPX1[j];
1585      mAPY1 = mAPY1 + usgAPY1[j];
1586      mAPZ1 = mAPZ1 + usgAPZ1[j];
1587      mRVX1 = mRVX1 + usgRVX1[j];
1588      mRVY1 = mRVY1 + usgRVY1[j];
1589      mRVZ1 = mRVZ1 + usgRVZ1[j];
1590
1591      //berechnete Achse
1592      mINC2 = mINC2 + usgINC2[j];
1593      mDEV2 = mDEV2 + usgDEV2[j];
1594      mANG2 = mANG2 + usgANG2[j];
1595      mLEVER2 = mLEVER2 + usgLEVER2[j];
1596      mAPX2 = mAPX2 + usgAPX2[j];
1597      mAPY2 = mAPY2 + usgAPY2[j];
1598      mAPZ2 = mAPZ2 + usgAPZ2[j];
1599      mRVX2 = mRVX2 + usgRVX2[j];
1600      mRVY2 = mRVY2 + usgRVY2[j];
1601      mRVZ2 = mRVZ2 + usgRVZ2[j];
1602
1603   }//Ende for Schleife
1604   ////////////////////////////////////////
1605   // Ende Achse berechnen                //
1606   ////////////////////////////////////////
1607   int av = sizePoint;
1608
1609   // MITTELWERT BERECHNUNG
1610   //teilen der summen zur mittelwertberechnung
1611   mINC1 = mINC1/av; //Malleolen Achse
1612   mDEV1 = mDEV1/av;
1613   mANG1 = mANG1/av;
1614   mLEVER1 = mLEVER1/av;
1615   mDIST1 = mDIST1/av;
1616   mAPX1 = mAPX1/av;
1617   mAPY1 = mAPY1/av;
1618   mAPZ1 = mAPZ1/av;
1619   mRVX1 = mRVX1/av;
1620   mRVY1 = mRVY1/av;
1621   mRVZ1 = mRVZ1/av;
1622
1623   mINC2 = mINC2/av; //berechnete Achse
1624   mDEV2 = mDEV2/av;
1625   mANG2 = mANG2/av;
1626   mLEVER2 = mLEVER2/av;
1627   mAPX2 = mAPX2/av;
1628   mAPY2 = mAPY2/av;
1629   mAPZ2 = mAPZ2/av;
1630   mRVX2 = mRVX2/av;
1631   mRVY2 = mRVY2/av;
1632   mRVZ2 = mRVZ2/av;
1633
1634   // STAB UND VAR BERECHNUNG
1635   //Variable Varianz und STAB
1636   double  dVAR_INC1=0,dVAR_INC2=0,
1637        dVAR_DEV1=0, dVAR_DEV2=0,
1638        dVAR_LEVER1=0, dVAR_LEVER2=0, dVAR_DIST1=0,
1639        dVAR_ANG1=0, dVAR_ANG2=0;
1640   double  dSTAB_INC1=0, dSTAB_INC2=0,
1641        dSTAB_DEV1=0, dSTAB_DEV2=0, dSTAB_DIST1=0,
1642        dSTAB_LEVER1=0, dSTAB_LEVER2=0,
```

```
1643        dSTAB_ANG1=0, dSTAB_ANG2;
1644    double quad;
1645    CString sSTAB_INC1, sSTAB_INC2,
1646        sSTAB_DEV1, sSTAB_DEV2,
1647        sSTAB_LEVER1, sSTAB_LEVER2, sSTAB_DIST1,
1648        sSTAB_ANG1, sSTAB_ANG2;
1649
1650    ////////////////////////////////
1651    //BERECHNUNG VARIANZ UND STAB //
1652    ////////////////////////////////
1653    for(j=0; j<sizePoint; j++)  //VARIANZ
1654    {
1655      quad = mINC1-usgINC1[j];  //quadrat
1656      dVAR_INC1 += quad*quad;
1657
1658      quad = mINC2-usgINC2[j];
1659      dVAR_INC2 += quad*quad;
1660
1661      quad = mDEV1-usgDEV1[j];
1662      dVAR_DEV1 += quad*quad;
1663
1664      quad = mDEV2-usgDEV2[j];
1665      dVAR_DEV2 += quad*quad;
1666
1667      quad = mLEVER1-usgLEVER1[j];
1668      dVAR_LEVER1 += quad*quad;
1669
1670      quad = mLEVER2-usgLEVER2[j];
1671      dVAR_LEVER2 += quad*quad;
1672
1673      quad = mDIST1-usgDIST1[j];
1674      dVAR_DIST1 += quad*quad;
1675
1676      quad = mANG1-usgANG1[j];
1677      dVAR_ANG1 += quad*quad;
1678
1679      quad = mANG2-usgANG2[j];
1680      dVAR_ANG2 += quad*quad;
1681    }
1682
1683    //Standardabweichung berechnen
1684    dSTAB_INC1 = sqrt(dVAR_INC1/(av-1));
1685    dSTAB_INC2 = sqrt(dVAR_INC2/(av-1));
1686    dSTAB_DEV1 = sqrt(dVAR_DEV1/(av-1));
1687    dSTAB_DEV2 = sqrt(dVAR_DEV2/(av-1));
1688    dSTAB_ANG1 = sqrt(dVAR_ANG1/(av-1));
1689    dSTAB_ANG2 = sqrt(dVAR_ANG2/(av-1));
1690    dSTAB_LEVER1 = sqrt(dVAR_LEVER1/(av-1));
1691    dSTAB_LEVER2 = sqrt(dVAR_LEVER2/(av-1));
1692    dSTAB_DIST1 = sqrt(dVAR_DIST1/(av-1));
1693
1694    ////////////////////////////////
1695    // Überprüfung auf Ausreisser //
1696    ////////////////////////////////
1697    //int maxError = 1;      //Abweichungen von 3* STABWN werden als Ausreisser bezeichnet!!!!
1698    double maxError = 1.3;
1699    int calcNew = 0;
1700    CString tempString;
1701    for(j=0; j<sizePoint; j++)
1702    {
1703      if( (usgINC1[j]<(mINC1 - maxError*dSTAB_INC1)) ||
1704        (usgINC1[j]>(mINC1 +maxError*dSTAB_INC1)) ||
1705        (usgDEV1[j]<(mDEV1 - maxError*dSTAB_DEV1)) ||
1706        (usgDEV1[j]>(mDEV1 + maxError*dSTAB_DEV1)))
1707
1708      {
1709        usgERROR1[j] = 1;
1710        calcNew = 1;
1711      }
1712
1713    }
1714    // STRINGAUSGABE der ACHSENERGEBNISSE //
1715    // E R G E B N I S S //
1716    //Ergebniss ausgeben in String
1717    for(j=0; j<sizePoint; j++)
1718    {
1719      //Ausgabe in String-> MM - ML Achse
1720      sUSG_INC.Format("%.2f", usgINC1[j]);  //in String umwandeln
1721      sUSG_DEV.Format("%.2f", usgDEV1[j]);
1722      sUSG_LEVER.Format("%.2f", usgLEVER1[j]);
1723      sUSG_ANGLE.Format("%.2f", usgANG1[j]);
1724      sUSG_DIST1.Format("%.2f", usgDIST1[j]);
1725      sUSG_APX.Format("%.2f", usgAPX1[j]);
1726      sUSG_APY.Format("%.2f", usgAPY1[j]);
```

```
1727      sUSG_APZ.Format("%.2f", usgAPZ1[j]);
1728      sUSG_RVX.Format("%.2f", usgRVX1[j]);
1729      sUSG_RVY.Format("%.2f", usgRVY1[j]);
1730      sUSG_RVZ.Format("%.2f", usgRVZ1[j]);
1731      sUSG_INC.Replace(".",","); //. duch , ersetzen
1732      sUSG_DEV.Replace(".",",");
1733      sUSG_LEVER.Replace(".",",");
1734      sUSG_ANGLE.Replace(".",",");
1735      sUSG_DIST1.Replace(".",",");
1736      sUSG_APX.Replace(".",",");
1737      sUSG_APY.Replace(".",",");
1738      sUSG_APZ.Replace(".",",");
1739      sUSG_RVX.Replace(".",",");
1740      sUSG_RVY.Replace(".",",");
1741      sUSG_RVZ.Replace(".",",");
1742      //richtung
1743      if(iD[j]==1) sUSG_DIR ="Inversion";
1744      else if(iD[j]==-1) sUSG_DIR="Eversion";
1745
1746      //Aussreiser klammern
1747      if(usgERROR1[j]==1)
1748      {
1749        tempString = sUSG_INC;
1750        sUSG_INC = "***"+tempString+"***";
1751
1752        tempString = sUSG_DEV;
1753        sUSG_DEV = "***"+tempString+"***";
1754
1755        tempString = sUSG_LEVER;
1756        sUSG_LEVER = "***"+tempString+"***";
1757
1758        tempString = sUSG_ANGLE;
1759        sUSG_ANGLE = "***"+tempString+"***";
1760      }
1761      //ins String schreiben
1762      sAxis2 += sUSG_INC+";"+sUSG_DEV+";"+sUSG_LEVER+";"+sUSG_ANGLE+";"+sUSG_DIST1+";"+sUSG_APX+";"
              +sUSG_APY+";"+sUSG_APZ+";"+sUSG_RVX+";"+sUSG_RVY+";"+sUSG_RVZ+";"+sUSG_DIR+";";
1763
1764
1765      //Ausgabe in String -> berechnete Achse
1766      sUSG_INC.Format("%.2f", usgINC2[j]);  //in String umwandeln
1767      sUSG_DEV.Format("%.2f", usgDEV2[j]);
1768      sUSG_LEVER.Format("%.2f", usgLEVER2[j]);
1769      sUSG_ANGLE.Format("%.2f", usgANG2[j]);
1770      sUSG_APX.Format("%.2f", usgAPX2[j]);
1771      sUSG_APY.Format("%.2f", usgAPY2[j]);
1772      sUSG_APZ.Format("%.2f", usgAPZ2[j]);
1773      sUSG_RVX.Format("%.2f", usgRVX2[j]);
1774      sUSG_RVY.Format("%.2f", usgRVY2[j]);
1775      sUSG_RVZ.Format("%.2f", usgRVZ2[j]);
1776      sUSG_INC.Replace(".",","); //. duch , ersetzen
1777      sUSG_DEV.Replace(".",",");
1778      sUSG_LEVER.Replace(".",",");
1779      sUSG_ANGLE.Replace(".",",");
1780      sUSG_APX.Replace(".",",");
1781      sUSG_APY.Replace(".",",");
1782      sUSG_APZ.Replace(".",",");
1783      sUSG_RVX.Replace(".",",");
1784      sUSG_RVY.Replace(".",",");
1785      sUSG_RVZ.Replace(".",",");
1786
1787      //Aussreiser klammern
1788      if(usgERROR1[j]==1)
1789      {
1790        tempString = sUSG_INC;
1791        sUSG_INC = "***"+tempString+"***";
1792
1793        tempString = sUSG_DEV;
1794        sUSG_DEV = "***"+tempString+"***";
1795
1796        tempString = sUSG_LEVER;
1797        sUSG_LEVER = "***"+tempString+"***";
1798
1799        tempString = sUSG_ANGLE;
1800        sUSG_ANGLE = "***"+tempString+"***";
1801      }
1802      //USGA rotiert um gemessene OSGA in String schreiben
1803      //für Ausgabe beider Achsen // entfernen
1804      //sAxis2 += sUSG_INC+";"+sUSG_DEV+";"+sUSG_LEVER+";"+sUSG_ANGLE+";"+sUSG_APX+";"+sUSG_APY
              +";"+sUSG_APZ+";"+sUSG_RVX+";"+sUSG_RVY+";"+sUSG_RVZ+";\r\n";
1805      sAxis2 += "\r\n";
1806
1807    }//Ende for Schleife Ausgabe in String
1808
```

```
1809    ////////////////////////
1810    // M I T T E L W E R T //
1811    // NEU berechnen      //
1812    // bei Aussreiser     //
1813    ////////////////////////
1814    if (calcNew == 1)
1815    {
1816      //zurücksetzen
1817      mINC1=0; mINC2=0; mDEV1=0; mDEV2=2; mANG1=0; mANG2=0; mLEVER1=0; mLEVER2=0; mDIST1=0;
1818      mAPX1=0; mAPX2=0; mAPY1=0; mAPY2=0; mAPZ1=0; mAPZ2=0;
1819      mRVX1=0; mRVX2=0; mRVY1=0; mRVY2=0; mRVZ1=0; mRVZ2=0;
1820      dVAR_INC1=0; dVAR_INC2=0; dVAR_DEV1=0; dVAR_DEV2=0; dVAR_ANG1=0; dVAR_ANG2=0; dVAR_LEVER1=0;
              dVAR_LEVER2=0; dVAR_DIST1=0;
1821      dSTAB_INC1=0; dSTAB_INC2=0; dSTAB_DEV1=0; dSTAB_DEV2=0; dSTAB_ANG1=0; dSTAB_ANG2=0;
              dSTAB_LEVER1=0; dSTAB_LEVER2=0; dSTAB_DIST1=0;
1822      av=0;
1823      for(j=0; j<sizePoint; j++)
1824      {
1825        if(usgERROR1[j]!=1)
1826        {
1827          //Mittelwertberechnung, Summe
1828          //Malleolenachse
1829          mINC1 = mINC1 + usgINC1[j];
1830          mDEV1 = mDEV1 + usgDEV1[j];
1831          mANG1 = mANG1 + usgANG1[j];
1832          mLEVER1 = mLEVER1 + usgLEVER1[j];
1833          mDIST1 = mDIST1 + usgDIST1[j];
1834          mAPX1 = mAPX1 + usgAPX1[j];
1835          mAPY1 = mAPY1 + usgAPY1[j];
1836          mAPZ1 = mAPZ1 + usgAPZ1[j];
1837          mRVX1 = mRVX1 + usgRVX1[j];
1838          mRVY1 = mRVY1 + usgRVY1[j];
1839          mRVZ1 = mRVZ1 + usgRVZ1[j];
1840
1841          //berechnete Achse
1842          mINC2 = mINC2 + usgINC2[j];
1843          mDEV2 = mDEV2 + usgDEV2[j];
1844          mANG2 = mANG2 + usgANG2[j];
1845          mLEVER2 = mLEVER2 + usgLEVER2[j];
1846          mAPX2 = mAPX2 + usgAPX2[j];
1847          mAPY2 = mAPY2 + usgAPY2[j];
1848          mAPZ2 = mAPZ2 + usgAPZ2[j];
1849          mRVX2 = mRVX2 + usgRVX2[j];
1850          mRVY2 = mRVY2 + usgRVY2[j];
1851          mRVZ2 = mRVZ2 + usgRVZ2[j];
1852
1853          av++;   //Anzahl der Werte für Mittelwertberechnung
1854        }
1855      }
1856      //mittelwert berechnen
1857      mINC1 = mINC1/av; //Malleolen Achse
1858      mDEV1 = mDEV1/av;
1859      mANG1 = mANG1/av;
1860      mLEVER1 = mLEVER1/av;
1861      mDIST1 = mDIST1/av;
1862      mAPX1 = mAPX1/av;
1863      mAPY1 = mAPY1/av;
1864      mAPZ1 = mAPZ1/av;
1865      mRVX1 = mRVX1/av;
1866      mRVY1 = mRVY1/av;
1867      mRVZ1 = mRVZ1/av;
1868
1869      mINC2 = mINC2/av; //berechnete Achse
1870      mDEV2 = mDEV2/av;
1871      mANG2 = mANG2/av;
1872      mLEVER2 = mLEVER2/av;
1873      mAPX2 = mAPX2/av;
1874      mAPY2 = mAPY2/av;
1875      mAPZ2 = mAPZ2/av;
1876      mRVX2 = mRVX2/av;
1877      mRVY2 = mRVY2/av;
1878      mRVZ2 = mRVZ2/av;
1879      //Ende Neuberechnung MITTELWERT
1880
1881      //Zuweisung der Mittelwerte an globale Variablen (Diagram zeichnen)
1882      m_usg_APX = mAPX1;
1883      m_usg_APY = mAPY1;
1884      m_usg_APZ = mAPZ1;
1885      m_usg_RVX = mRVX1;
1886      m_usg_RVY = mRVY1;
1887      m_usg_RVZ = mRVZ1;
1888
1889      ////////////////////////////////////
1890      //NEU BERECHNUNG VARIANZ UND STAB //
```

```
1891      ////////////////////////////////////
1892      for(j=0; j<sizePoint; j++)  //VARIANZ
1893      {
1894        if(usgERROR1[j]!=1)
1895        {
1896          quad = mINC1-usgINC1[j];  //quadrat
1897          dVAR_INC1 += quad*quad;
1898
1899          quad = mINC2-usgINC2[j];
1900          dVAR_INC2 += quad*quad;
1901
1902          quad = mDEV1-usgDEV1[j];
1903          dVAR_DEV1 += quad*quad;
1904
1905          quad = mDEV2-usgDEV2[j];
1906          dVAR_DEV2 += quad*quad;
1907
1908          quad = mLEVER1-usgLEVER1[j];
1909          dVAR_LEVER1 += quad*quad;
1910
1911          quad = mLEVER2-usgLEVER2[j];
1912          dVAR_LEVER2 += quad*quad;
1913
1914          quad = mDIST1-usgDIST1[j];
1915          dVAR_DIST1 += quad*quad;
1916
1917          quad = mANG1-usgANG1[j];
1918          dVAR_ANG1 += quad*quad;
1919
1920          quad = mANG2-usgANG2[j];
1921          dVAR_ANG2 += quad*quad;
1922        }
1923      }//Ende Neuberechnung STAB und VAR
1924
1925      //Standardabweichung berechnen
1926      dSTAB_INC1 = sqrt(dVAR_INC1/(av-1));
1927      dSTAB_INC2 = sqrt(dVAR_INC2/(av-1));
1928      dSTAB_DEV1 = sqrt(dVAR_DEV1/(av-1));
1929      dSTAB_DEV2 = sqrt(dVAR_DEV2/(av-1));
1930      dSTAB_ANG1 = sqrt(dVAR_ANG1/(av-1));
1931      dSTAB_ANG2 = sqrt(dVAR_ANG2/(av-1));
1932      dSTAB_LEVER1 = sqrt(dVAR_LEVER1/(av-1));
1933      dSTAB_LEVER2 = sqrt(dVAR_LEVER2/(av-1));
1934      dSTAB_DIST1 = sqrt(dVAR_DIST1/(av-1));
1935
1936    }
1937
1938    //Fehler <5; <10; <15; >15; für Ausgabe an GUI
1939    //Inklinationswinkel
1940    if (dSTAB_INC1<5) m_iStabInc=1;
1941    else if (dSTAB_INC1<10) m_iStabInc=2;
1942    else if (dSTAB_INC1<15) m_iStabInc=3;
1943    else if (dSTAB_INC1>15) m_iStabInc=4;
1944    //Deviationswinkel
1945    if (dSTAB_DEV1<5) m_iStabDev=1;
1946    else if (dSTAB_DEV1<10) m_iStabDev=2;
1947    else if (dSTAB_DEV1<15) m_iStabDev=3;
1948    else if (dSTAB_DEV1>15) m_iStabDev=4;
1949    //Hebel
1950    if (dSTAB_LEVER1<5) m_iStabLever=1;
1951    else if (dSTAB_LEVER1<10) m_iStabLever=2;
1952    else if (dSTAB_LEVER1<15) m_iStabLever=3;
1953    else if (dSTAB_LEVER1>15) m_iStabLever=4;
1954    //Winkel
1955    if (dSTAB_ANG1<5) m_iStabAng=1;
1956    else if (dSTAB_ANG1<10) m_iStabAng=2;
1957    else if (dSTAB_ANG1<15) m_iStabAng=3;
1958    else if (dSTAB_ANG1>15) m_iStabAng=4;
1959
1960    // STRINGAUSGABE der MITTELWERT
1961    // M I T T E L W E R T E //
1962    //Ausgabe in String-> MIttelwerte
1963    sAxis2 += "Mittelwerte ;;;;;;;;;;;\r\n";//Mittelwert\r\n";
1964    //Mittelwerte der OSGA ML-MM
1965    sUSG_INC.Format("%.2f", mINC1);    //umwandeln in string
1966    sUSG_DEV.Format("%.2f", mDEV1);
1967    sUSG_LEVER.Format("%.2f", mLEVER1);
1968    sUSG_ANGLE.Format("%.2f", mANG1);
1969    sUSG_DIST1.Format("%.2f", mDIST1);
1970    sUSG_APX.Format("%.2f", mAPX1);
1971    sUSG_APY.Format("%.2f", mAPY1);
1972    sUSG_APZ.Format("%.2f", mAPZ1);
1973    sUSG_RVX.Format("%.2f", mRVX1);
1974    sUSG_RVY.Format("%.2f", mRVY1);
```

```
1975    sUSG_RVZ.Format("%.2f", mRVZ1);
1976    sUSG_INC.Replace(".",",");       //. durch , ersetzen
1977    sUSG_DEV.Replace(".",",");
1978    sUSG_LEVER.Replace(".",",");
1979    sUSG_ANGLE.Replace(".",",");
1980    sUSG_DIST1.Replace(".",",");
1981    sUSG_APX.Replace(".",",");
1982    sUSG_APY.Replace(".",",");
1983    sUSG_APZ.Replace(".",",");
1984    sUSG_RVX.Replace(".",",");
1985    sUSG_RVY.Replace(".",",");
1986    sUSG_RVZ.Replace(".",",");
1987
1988    //Zuweisung für Ausgabe an GUI (ML-MM OSGA)
1989    sUSGAV_INC = sUSG_INC;
1990    sUSGAV_DEV = sUSG_DEV;
1991    sUSGAV_LEVER = sUSG_LEVER;
1992    sOut_usgANGLE = sUSG_ANGLE;
1993
1994    //Ausgabe in Datei (CVS File)
1995    sAxis2 += sUSG_INC+";"+sUSG_DEV+";"+sUSG_LEVER+";"+sUSG_ANGLE+";"+sUSG_DIST1+";"+sUSG_APX+";"+
            sUSG_APY+";"+sUSG_APZ+";"+sUSG_RVX+";"+sUSG_RVY+";"+sUSG_RVZ+";;";
1996
1997    sUSG_INC.Format("%.2f", mINC2);    //umwandeln in string
1998    sUSG_DEV.Format("%.2f", mDEV2);
1999    sUSG_LEVER.Format("%.2f", mLEVER2);
2000    sUSG_ANGLE.Format("%.2f", mANG2);
2001    sUSG_APX.Format("%.2f", mAPX2);
2002    sUSG_APY.Format("%.2f", mAPY2);
2003    sUSG_APZ.Format("%.2f", mAPZ2);
2004    sUSG_RVX.Format("%.2f", mRVX2);
2005    sUSG_RVY.Format("%.2f", mRVY2);
2006    sUSG_RVZ.Format("%.2f", mRVZ2);
2007    sUSG_INC.Replace(".",",");       //. durch , ersetzen
2008    sUSG_DEV.Replace(".",",");
2009    sUSG_LEVER.Replace(".",",");
2010    sUSG_ANGLE.Replace(".",",");
2011    sUSG_APX.Replace(".",",");
2012    sUSG_APY.Replace(".",",");
2013    sUSG_APZ.Replace(".",",");
2014    sUSG_RVX.Replace(".",",");
2015    sUSG_RVY.Replace(".",",");
2016    sUSG_RVZ.Replace(".",",");
2017
2018    //für Ausgabe beider Achsen // entfernen
2019    //sAxis2 += sUSG_INC+";"+sUSG_DEV+";"+sUSG_LEVER+";"+sUSG_ANGLE+";"+sUSG_APX+";"+sUSG_APY+";"+
            sUSG_APZ+";"+sUSG_RVX+";"+sUSG_RVY+";"+sUSG_RVZ+";\r\n";
2020    sAxis2 += "\r\n";
2021
2022    // S T A B W //
2023    //UMWANDELN DER STAB IN STRING
2024
2025    sSTAB_INC1.Format("%.2f",dSTAB_INC1);       //MM-ML Achse
2026    sSTAB_DEV1.Format("%.2f",dSTAB_DEV1);
2027    sSTAB_ANG1.Format("%.2f",dSTAB_ANG1);
2028    sSTAB_LEVER1.Format("%.2f",dSTAB_LEVER1);
2029    sSTAB_DIST1.Format("%.2f",dSTAB_DIST1);
2030    sSTAB_INC2.Format("%.2f",dSTAB_INC2);       //berechnete Achse
2031    sSTAB_DEV2.Format("%.2f",dSTAB_DEV2);
2032    sSTAB_ANG2.Format("%.2f",dSTAB_ANG2);
2033    sSTAB_LEVER2.Format("%.2f",dSTAB_LEVER2);
2034    //. durch , ersetzen
2035    sSTAB_INC1.Replace(".",",");
2036    sSTAB_DEV1.Replace(".",",");
2037    sSTAB_ANG1.Replace(".",",");
2038    sSTAB_LEVER1.Replace(".",",");
2039    sSTAB_DIST1.Replace(".",",");
2040    sSTAB_INC2.Replace(".",",");
2041    sSTAB_DEV2.Replace(".",",");
2042    sSTAB_ANG2.Replace(".",",");
2043    sSTAB_LEVER2.Replace(".",",");
2044
2045    //Ausgabe der STABWN in String
2046    sAxis2 += "STAB\r\n";//;;;;;;;;;;STAB\r\n";
2047    sAxis2 += sSTAB_INC1+";"+sSTAB_DEV1+";"+sSTAB_LEVER1+";"+sSTAB_ANG1+";"+sSTAB_DIST1+";;;;;;;\r\
            n";//"+sSTAB_INC2+";"+sSTAB_DEV2+";"+sSTAB_LEVER2+";"+sSTAB_ANG2+";\r\n";
2048
2049 }
2050 // ENDE
2051 ////////////////////////////////////////////////////////
2052
2053
2054 ////////////////////////////////////////////////////////
2055 //Routine für Berechnung instantaner Achsem
```

```
2056 void CUsgDlg::OnCalcAxis()
2057 {
2058   //Variable für Bewegungsrichtung
2059   CString richtung;
2060   int iMove,changeDir;
2061   double xVector;
2062
2063   //Bewegungsrichtung, positiv-> von - nach +
2064   //Bewegungsrichtung, negativ-> von + nach -
2065   if (Achse.Leg == 'R')
2066   {
2067     iMove = 1; //bei rechtem Bein, geht die Bewegung von lateral nach medial -> -x nach +x ->
           Bewegungsrichtung positiv
2068   }
2069   else iMove = -1;     //linkes BEIN!
2070
2071   //Array löschen
2072   usg_INC.RemoveAll();  //Inlinationswinkel
2073   usg_DEV.RemoveAll();  //Deviationswinkel
2074   usg_AP_X.RemoveAll(); //Aufpunkt X
2075   usg_AP_Y.RemoveAll(); //Aufpunkt Y
2076   usg_AP_Z.RemoveAll(); //Aufpunkt Z
2077   usg_RV_X.RemoveAll(); //richtungsvektor X
2078   usg_RV_Y.RemoveAll(); //Richtungsvektor Y
2079   usg_RV_Z.RemoveAll(); //Richtungsvektor Z
2080   usg_LEVER.RemoveAll();  //Hebel Achilles
2081   usg_INC1.RemoveAll(); //Inlinationswinkel
2082   usg_DEV1.RemoveAll(); //Deviationswinkel
2083   usg_AP_X1.RemoveAll();  //Aufpunkt X
2084   usg_AP_Y1.RemoveAll();  //Aufpunkt Y
2085   usg_AP_Z1.RemoveAll();  //Aufpunkt Z
2086   usg_RV_X1.RemoveAll();  //richtungsvektor X
2087   usg_RV_Y1.RemoveAll();  //Richtungsvektor Y
2088   usg_RV_Z1.RemoveAll();  //Richtungsvektor Z
2089   usg_LEVER1.RemoveAll(); //Hebel Achilles
2090
2091   //String für Augabe
2092   sAxis_inst = "USGA␣um␣OSGA␣berechnet;;;;;;;;;;USGA␣um␣OSGA␣(ML,MM)\r\n";
2093   sAxis_inst += "Inc.;Dev.;Lever;AP(X);AP(Y);AP(Z);RV(X);RV(Y);RV(Z);;Inc.;Dev.;Lever;AP(X);AP(Y)
         ;AP(Z);RV(X);RV(Y);RV(Z)\r\n";
2094
2095   //Arrayeinträge USG Daten
2096   double size_usg = fpos_usg_X.GetSize();
2097   int n = 1;
2098   // Abstand zwischen den einzelen Marker Punkte in mm
2099   double v_ab = 10; //20 OK
2100   UX_VECTOR3 V;
2101   double abstand_P1P2, abstand_P2P3;
2102   UX_VECTOR3 temp_P2, temp_P3, temp_P1P2, temp_P2P3, temp_DIR, tempPunkt1, tempPunkt2, tempPunkt3
         ;
2103   UX_VECTOR3 temp_P1(fpos_usg_X[0], fpos_usg_Y[0], fpos_usg_Z[0]);
2104
2105   //Array wird durchlaufen
2106   for (int i = 1; i < size_usg; i++)
2107   {
2108     //suche Wert zwei für Achsberechnung
2109     if (n == 1)
2110     {
2111       temp_P2.X = fpos_usg_X[i];
2112       temp_P2.Y = fpos_usg_Y[i];
2113       temp_P2.Z = fpos_usg_Z[i];                    //Vektor zu Punkt 2
2114       temp_P1P2 = V.Subtract(temp_P1, temp_P2);        //Vektor von Punkt 1 zu Punk2
2115       abstand_P1P2 = temp_P1P2.Magnitude();          //Länge Vektor Punk1 zu Punkt2
2116
2117       //Wenn der Abstand größer als 10mm...
2118       if (abstand_P1P2 > v_ab)
2119       {
2120         //Überprüfung ob Vektor in die richtige Richtung zeigt
2121         xVector = temp_P2.X - temp_P1.X;             //Berechnung der Richtung der X Richtung
2122         changeDir = xVector * iMove;               //Variable für Richtungsänderung
2123         //Richtung hat sich geändert
2124         if (changeDir < 0)
2125         {
2126           temp_P1 = temp_P2;                    //Richtung hat sich geändert, P1 <- P2
2127           iMove = iMove * (-1);                 //Richtung wird umgedreht
2128         }
2129         else
2130         {
2131           n = 2;     //gleiche Richtung weiter zum nächsten Schritt
2132         }
2133       }
2134     }
2135     if (n == 2)
2136     {
```

```
2137        //suche Wert drei für Achsberechnung
2138        temp_P3.X = fpos_usg_X[i];
2139        temp_P3.Y = fpos_usg_Y[i];
2140        temp_P3.Z = fpos_usg_Z[i];                    //Vektor zu Punkt3
2141        temp_P2P3 = V.Subtract(temp_P2, temp_P3);          //Vektor von Punkt2 zu Punk3
2142        abstand_P2P3 = temp_P2P3.Magnitude();
2143        if (abstand_P2P3 > v_ab)
2144        {
2145          //Überprüfung auf Richtungsänderung
2146          xVector = temp_P3.X - temp_P2.X;              //Berechnung Richtung x Vektor
2147          changeDir = xVector * iMove;                 //Variable für Richtungsänderung
2148          //Richtung hat sich geändert
2149          if (changeDir < 0)
2150          {
2151            temp_P1 = temp_P3;                    //P3 -> P1
2152            n = 1;                              //wieder Punkt2 suchen
2153            iMove = iMove * (-1);                 //Richtung umdrehen
2154          }
2155          else
2156          {
2157            //bei Positiver x Richtung Punkt P1 und P3 vertauschen
2158            if ((iMove < 0 && Achse.Leg == 'R') || (iMove > 0 && Achse.Leg == 'L')) //Bewegung nach
                  lateral und rechter Fuss -> Punkte tauschen
2159            {
2160              tempPunkt1 = temp_P3;
2161              tempPunkt2 = temp_P2;
2162              tempPunkt3 = temp_P1;
2163            }
2164            else if ((iMove > 0 && Achse.Leg == 'R') || (iMove < 0 && Achse.Leg == 'L'))  //Kein
                 Punkte tauschen, nur Werte zuweisen
2165            {
2166              tempPunkt1 = temp_P1;
2167              tempPunkt2 = temp_P2;
2168              tempPunkt3 = temp_P3;
2169            }
2170            //alle drei Werte gefunden-> Achsberechnung
2171            //Achse.calcUSG(temp_P1.X, temp_P1.Y, temp_P1.Z, temp_P2.X, temp_P2.Y, temp_P2.Z,
                 temp_P3.X, temp_P3.Y, temp_P3.Z);
2172            Achse.calcUSG(tempPunkt1.X, tempPunkt1.Y, tempPunkt1.Z, tempPunkt2.X, tempPunkt2.Y,
                 tempPunkt2.Z, tempPunkt3.X, tempPunkt3.Y, tempPunkt3.Z);
2173
2174            //Werte neu festlegen Punkt2 -> Punkt1, Punkt3-> Punkt2
2175            temp_P1 = temp_P2;     //Vektor2 wird Vektor1
2176            temp_P2 = temp_P3;     //Vektor3 wird Vektor2
2177
2178            //Werte in Array einlesen /USGA rotiert um die berechnete OSGA
2179            usg_INC.Add(Achse.USG_Inc);    //Inlinationswinkel
2180            usg_DEV.Add(Achse.USG_Dev);    //Deviationswinkel
2181            usg_AP_X.Add(Achse.USG_AP.X); //Aufpunkt X
2182            usg_AP_Y.Add(Achse.USG_AP.Y); //Aufpunkt Y
2183            usg_AP_Z.Add(Achse.USG_AP.Z); //Aufpunkt Z
2184            usg_RV_X.Add(Achse.USG_RV.X); //richtungsvektor X
2185            usg_RV_Y.Add(Achse.USG_RV.Y); //Richtungsvektor Y
2186            usg_RV_Z.Add(Achse.USG_RV.Z); //Richtungsvektor Z
2187            usg_LEVER.Add(Achse.Lever_USGA);//Hebel Achilles
2188
2189            //Werte in Array einlesen /USGA rotiert um die gemessene OSGA (ML-MM)
2190            usg_INC1.Add(Achse.USG_inc1);    //Inlinationswinkel
2191            usg_DEV1.Add(Achse.USG_dev1);    //Deviationswinkel
2192            usg_AP_X1.Add(Achse.USG_APX1);   //Aufpunkt X
2193            usg_AP_Y1.Add(Achse.USG_APY1);   //Aufpunkt Y
2194            usg_AP_Z1.Add(Achse.USG_APZ1);   //Aufpunkt Z
2195            usg_RV_X1.Add(Achse.USG_RVX1);   //richtungsvektor X
2196            usg_RV_Y1.Add(Achse.USG_RVY1);   //Richtungsvektor Y
2197            usg_RV_Z1.Add(Achse.USG_RVZ1);   //Richtungsvektor Z
2198            usg_LEVER1.Add(Achse.lever_USG_1);//Hebel Achilles
2199
2200            //Ausgabe in String
2201            if(iMove<0) richtung = "-1";
2202            else richtung = "1";
2203            sUSG_INC.Format("%.3f",Achse.USG_Inc);
2204            sUSG_DEV.Format("%.3f", Achse.USG_Dev);
2205            sUSG_LEVER.Format("%.3f", Achse.Lever_USGA);
2206            sUSG_APX.Format("%.3f", Achse.USG_AP.X);
2207            sUSG_APY.Format("%.3f", Achse.USG_AP.Y);
2208            sUSG_APZ.Format("%.3f", Achse.USG_AP.Z);
2209            sUSG_RVX.Format("%.3f", Achse.USG_RV.X);
2210            sUSG_RVY.Format("%.3f", Achse.USG_RV.Y);
2211            sUSG_RVZ.Format("%.3f", Achse.USG_RV.Z);
2212            sUSG_INC.Replace(".",",");
2213            sUSG_DEV.Replace(".",",");
2214            sUSG_LEVER.Replace(".",",");
2215            sUSG_APX.Replace(".",",");
2216            sUSG_APY.Replace(".",",");
```

```
2217            sUSG_APZ.Replace(".",",");
2218            sUSG_RVX.Replace(".",",");
2219            sUSG_RVY.Replace(".",",");
2220            sUSG_RVZ.Replace(".",",");
2221            sAxis_inst += sUSG_INC +";"+ sUSG_DEV +";"+ sUSG_LEVER +";"+ sUSG_APX +";"+ sUSG_APY +"
                    ;"+ sUSG_APZ +";"+ sUSG_RVX +";"+ sUSG_RVY +";"+ sUSG_RVZ +";;";
2222
2223            //Ausgabe in String (USGA rotiert um ML-MM)
2224            sUSG_INC.Format("%.3f",Achse.USG_inc1);
2225            sUSG_DEV.Format("%.3f", Achse.USG_dev1);
2226            sUSG_LEVER.Format("%.3f", Achse.lever_USG_1);
2227            sUSG_APX.Format("%.3f", Achse.USG_APX1);
2228            sUSG_APY.Format("%.3f", Achse.USG_APY1);
2229            sUSG_APZ.Format("%.3f", Achse.USG_APZ1);
2230            sUSG_RVX.Format("%.3f", Achse.USG_RVX1);
2231            sUSG_RVY.Format("%.3f", Achse.USG_RVY1);
2232            sUSG_RVZ.Format("%.3f", Achse.USG_RVZ1);
2233            sUSG_INC.Replace(".",",");
2234            sUSG_DEV.Replace(".",",");
2235            sUSG_LEVER.Replace(".",",");
2236            sUSG_APX.Replace(".",",");
2237            sUSG_APY.Replace(".",",");
2238            sUSG_APZ.Replace(".",",");
2239            sUSG_RVX.Replace(".",",");
2240            sUSG_RVY.Replace(".",",");
2241            sUSG_RVZ.Replace(".",",");
2242            sAxis_inst += sUSG_INC +";"+ sUSG_DEV +";"+ sUSG_LEVER +";"+ sUSG_APX +";"+ sUSG_APY +"
                    ;"+ sUSG_APZ +";"+ sUSG_RVX +";"+ sUSG_RVY +";"+ sUSG_RVZ + ";"+richtung+"\r\n";
2243          }
2244        }
2245      }
2246    }
2247
2248    //Mittelwerte berechnen
2249    double temp_av;
2250    usg_av_INC = 0;
2251    usg_av_DEV = 0;
2252    usg_av_LEVER = 0;
2253    usg_av_APX = 0;
2254    usg_av_APY = 0;
2255    usg_av_APZ = 0;
2256    usg_av_RVX = 0;
2257    usg_av_RVY = 0;
2258    usg_av_RVZ = 0;
2259    usg_av_INC1 = 0;
2260    usg_av_DEV1 = 0;
2261    usg_av_LEVER1 = 0;
2262    usg_av_APX1 = 0;
2263    usg_av_APY1 = 0;
2264    usg_av_APZ1 = 0;
2265    usg_av_RVX1 = 0;
2266    usg_av_RVY1 = 0;
2267    usg_av_RVZ1 = 0;
2268
2269    int size_av = usg_INC.GetSize();
2270    for (int ii = 0; ii < size_av; ii++)
2271    {
2272      //Mittelwert USGA um berechnete OSGA
2273      temp_av = usg_INC[ii];
2274      usg_av_INC = usg_av_INC + temp_av;
2275      temp_av = usg_DEV[ii];
2276      usg_av_DEV = usg_av_DEV + temp_av;
2277      temp_av = usg_LEVER[ii];
2278      usg_av_LEVER = usg_av_LEVER + temp_av;
2279      temp_av = usg_AP_X[ii];
2280      usg_av_APX = usg_av_APX + temp_av;
2281      temp_av = usg_AP_Y[ii];
2282      usg_av_APY = usg_av_APY + temp_av;
2283      temp_av = usg_AP_Z[ii];
2284      usg_av_APZ = usg_av_APZ + temp_av;
2285      temp_av = usg_RV_X[ii];
2286      usg_av_RVX = usg_av_RVX + temp_av;
2287      temp_av = usg_RV_Y[ii];
2288      usg_av_RVY = usg_av_RVY + temp_av;
2289      temp_av = usg_RV_Z[ii];
2290      usg_av_RVZ = usg_av_RVZ + temp_av;
2291
2292      //Mittelwert USGA um gemessene OSGA
2293      temp_av = usg_INC1[ii];
2294      usg_av_INC1 = usg_av_INC1 + temp_av;
2295      temp_av = usg_DEV1[ii];
2296      usg_av_DEV1 = usg_av_DEV1 + temp_av;
2297      temp_av = usg_LEVER[ii];
2298      usg_av_LEVER1 = usg_av_LEVER1 + temp_av;
```

```
2299      temp_av = usg_AP_X1[ii];
2300      usg_av_APX1 = usg_av_APX1 + temp_av;
2301      temp_av = usg_AP_Y1[ii];
2302      usg_av_APY1 = usg_av_APY1 + temp_av;
2303      temp_av = usg_AP_Z1[ii];
2304      usg_av_APZ1 = usg_av_APZ1 + temp_av;
2305      temp_av = usg_RV_X1[ii];
2306      usg_av_RVX1 = usg_av_RVX1 + temp_av;
2307      temp_av = usg_RV_Y1[ii];
2308      usg_av_RVY1 = usg_av_RVY1 + temp_av;
2309      temp_av = usg_RV_Z1[ii];
2310      usg_av_RVZ1 = usg_av_RVZ1 + temp_av;
2311      //Ausgabe in String
2312    }
2313    //Mittelwert USGA um berechnete OSGA
2314    usg_av_INC = usg_av_INC / size_av;
2315    usg_av_DEV = usg_av_DEV / size_av;
2316    usg_av_LEVER = usg_av_LEVER / size_av;
2317    usg_av_APX = usg_av_APX / size_av;
2318    usg_av_APY = usg_av_APY / size_av;
2319    usg_av_APZ = usg_av_APZ / size_av;
2320    usg_av_RVX = usg_av_RVX / size_av;
2321    usg_av_RVY = usg_av_RVY / size_av;
2322    usg_av_RVZ = usg_av_RVZ / size_av;
2323
2324    //Mittelwert USGA um gemessene OSGA
2325    usg_av_INC1 = usg_av_INC1 / size_av;
2326    usg_av_DEV1 = usg_av_DEV1 / size_av;
2327    usg_av_LEVER1 = usg_av_LEVER1 / size_av;
2328    usg_av_APX1 = usg_av_APX1 / size_av;
2329    usg_av_APY1 = usg_av_APY1 / size_av;
2330    usg_av_APZ1 = usg_av_APZ1 / size_av;
2331    usg_av_RVX1 = usg_av_RVX1 / size_av;
2332    usg_av_RVY1 = usg_av_RVY1 / size_av;
2333    usg_av_RVZ1 = usg_av_RVZ1 / size_av;
2334
2335
2336    //Ausgabe der Mittelwerte in String
2337    sAxis_inst += "Mittelwerte;;;;;;;;;;Mittelwert\r\n";
2338    sUSG_INC.Format("%.3f", usg_av_INC);
2339    sUSG_DEV.Format("%.3f", usg_av_DEV);
2340    sUSG_LEVER.Format("%.3f", usg_av_LEVER);
2341    sUSG_APX.Format("%.3f", usg_av_APX);
2342    sUSG_APY.Format("%.3f", usg_av_APY);
2343    sUSG_APZ.Format("%.3f", usg_av_APZ);
2344    sUSG_RVX.Format("%.3f", usg_av_RVX);
2345    sUSG_RVY.Format("%.3f", usg_av_RVY);
2346    sUSG_RVZ.Format("%.3f", usg_av_RVZ);
2347    sUSG_INC.Replace(".",",");
2348    sUSG_DEV.Replace(".",",");
2349    sUSG_LEVER.Replace(".",",");
2350    sUSG_APX.Replace(".",",");
2351    sUSG_APY.Replace(".",",");
2352    sUSG_APZ.Replace(".",",");
2353    sUSG_RVX.Replace(".",",");
2354    sUSG_RVY.Replace(".",",");
2355    sUSG_RVZ.Replace(".",",");
2356    sAxis_inst += sUSG_INC +";"+ sUSG_DEV +";"+ sUSG_LEVER +";"+ sUSG_APX +";"+ sUSG_APY +";"+
          sUSG_APZ +";"+ sUSG_RVX +";"+ sUSG_RVY +";"+ sUSG_RVZ +";;";
2357
2358    //Ausgabe der Mittelwerte in String (USGA um gemessene OSGA)
2359    sUSG_INC.Format("%.3f", usg_av_INC1);
2360    sUSG_DEV.Format("%.3f", usg_av_DEV1);
2361    sUSG_LEVER.Format("%.3f", usg_av_LEVER1);
2362    sUSG_APX.Format("%.3f", usg_av_APX1);
2363    sUSG_APY.Format("%.3f", usg_av_APY1);
2364    sUSG_APZ.Format("%.3f", usg_av_APZ1);
2365    sUSG_RVX.Format("%.3f", usg_av_RVX1);
2366    sUSG_RVY.Format("%.3f", usg_av_RVY1);
2367    sUSG_RVZ.Format("%.3f", usg_av_RVZ1);
2368    sUSG_INC.Replace(".",",");
2369    sUSG_DEV.Replace(".",",");
2370    sUSG_LEVER.Replace(".",",");
2371    sUSG_APX.Replace(".",",");
2372    sUSG_APY.Replace(".",",");
2373    sUSG_APZ.Replace(".",",");
2374    sUSG_RVX.Replace(".",",");
2375    sUSG_RVY.Replace(".",",");
2376    sUSG_RVZ.Replace(".",",");
2377    sAxis_inst += sUSG_INC +";"+ sUSG_DEV +";"+ sUSG_LEVER +";"+ sUSG_APX +";"+ sUSG_APY +";"+
          sUSG_APZ +";"+ sUSG_RVX +";"+ sUSG_RVY +";"+ sUSG_RVZ +"\r\n";
2378    //Zuweisung für Ausgabe
2379    sUSGAV_INC.Format("%.1f", usg_av_INC1); sUSGAV_INC.Replace(".",",");
2380    sUSGAV_DEV.Format("%.1f", usg_av_DEV1); sUSGAV_DEV.Replace(".",",");
```

```
2381   sUSGAV_LEVER.Format("%.1f", usg_av_LEVER1); sUSGAV_LEVER.Replace(".",",");
2382 }//ENDE
2383 ///////
2384
2385 ////////////////////////////////////////////////////////////////////////////
2386 //Erneute Berechnung der Parameter USGA OSGA
2387 void CUsgDlg::OnCalcnew()
2388 {
2389   // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
2390   if (m_Proband.m_mFuss == 0)
2391   {
2392     Achse.Leg = 'L';  //Linkes Bein
2393   }
2394   else
2395   {
2396     Achse.Leg = 'R'; //Rechtes Bein
2397   }
2398
2399   //Aufruf der Funktion zum Ermitteln der Punkte zur Berechnung der Achsen
2400   OnCalculate();
2401
2402 }//ENDE
2403 ///////
2404
2405 //Neu Berenchen aus Menu
2406 void CUsgDlg::OnMENUCALCNEW()
2407 {
2408   OnCalcnew();
2409 }
2410
2411
2412 ////////////////////////////////////////////////////////////////////////////////////////
2413 //routine für Berechung der Punkte für calcOsg u. calcUsg (3 Punkte des Einpunktverfahrens)
2414 void CUsgDlg::OnCalculate()
2415 {
2416   //Funktion zum Daten Filtern wird aufgerufen
2417   callFilter();
2418
2419   CString sTemp1X, sTemp1Y, sTemp1Z, sTemp2X, sTemp2Y, sTemp2Z, sTemp3X, sTemp3Y, sTemp3Z;
2420   sKoordinaten = "OSGA;;;;USGA\r\n";
2421   sKoordinaten += "X;Y;Z;;X;Y;Z\r\n";
2422
2423   //Array wird einem String zugewiesen (für die Ausgabe)
2424   double count;
2425   int j = pos_osg_X.GetSize();  //Größe Array OSGA
2426   int k = fpos_usg_X.GetSize(); //Größe Array USGA
2427   if (k > j) count = k;
2428   else count = j;
2429
2430   for (int i = 0; i <= count; i++)
2431   {
2432     //Daten OSGA
2433     if (j > i)
2434     {
2435       //Erstellen eines temporären Vektors aus drei Array Einträgen
2436       UX_VECTOR3 temp_fOSG_XYZ (fpos_osg_X[i], fpos_osg_Y[i], fpos_osg_Z[i]);
2437       //Transformieren des temporären Vektors in das Tibia Koordinatensystem
2438       Achse.transform(&temp_fOSG_XYZ);
2439       //Casten in einen String
2440       sTemp1X.Format("%.3f",temp_fOSG_XYZ.X);
2441       sTemp1Y.Format("%.3f",temp_fOSG_XYZ.Y);
2442       sTemp1Z.Format("%.3f",temp_fOSG_XYZ.Z);
2443       sTemp1X.Replace(".",",");
2444       sTemp1Y.Replace(".",",");
2445       sTemp1Z.Replace(".",",");
2446     }
2447     else
2448     {
2449       sTemp1X = "";
2450       sTemp1Y = "";
2451       sTemp1Z = "";
2452     }
2453
2454     //Daten USGA
2455     if (k > i)
2456     {
2457       //gefilterte Werte
2458       UX_VECTOR3 temp_fUSG_XYZ(fpos_usg_X[i], fpos_usg_Y[i], fpos_usg_Z[i]);
2459       Achse.transform(&temp_fUSG_XYZ);
2460       //Casten in einen String
2461       //Gefilterte Werte
2462       sTemp2X.Format("%.3f",temp_fUSG_XYZ.X);
2463       sTemp2Y.Format("%.3f",temp_fUSG_XYZ.Y);
2464       sTemp2Z.Format("%.3f",temp_fUSG_XYZ.Z);
```

```
2465        sTemp2X.Replace(".",",");
2466        sTemp2Y.Replace(".",",");
2467        sTemp2Z.Replace(".",",");
2468      }
2469      else
2470      {
2471        sTemp2X = "";
2472        sTemp2Y = "";
2473        sTemp2Z = "";
2474      }
2475
2476
2477      sKoordinaten += sTemp1X + ";" + sTemp1Y + ";" + sTemp1Z + ";;" + sTemp2X + ";" + sTemp2Y + ";
          " + sTemp2Z +"\r\n";
2478    }
2479
2480    //Werte zuweisen
2481    //OSGA
2482    step1X3 = fpos_osg_X[1];
2483    step1Y3 = fpos_osg_Y[1];
2484    step1Z3 = fpos_osg_Z[1];
2485
2486    step3X3 = fpos_osg_X[j-1];
2487    step3Y3 = fpos_osg_Y[j-1];
2488    step3Z3 = fpos_osg_Z[j-1];
2489
2490
2491    UX_VECTOR3 p1p3_osg(fpos_osg_X[1] - fpos_osg_X[j-1], fpos_osg_Y[1] - fpos_osg_Y[j-1],
          fpos_osg_Z[1] - fpos_osg_Z[j-1]);
2492
2493    double abstand_p1p3_osg = p1p3_osg.Magnitude();
2494    double abstand_p1p2_osg;
2495    double abstand_p2p3_osg;
2496    double ab_max = 0;
2497    double ab_max_temp = 0;
2498    double ab_max_control= abstand_p1p3_osg/3;
2499    int number;
2500    //Schleife um P2 zu suchen
2501    for (int ii = 2; ii < j; ii++)
2502    {
2503      //Abstand vom Punkt1 zum Punkt2
2504      UX_VECTOR3 p1p2_osg(fpos_osg_X[1] - fpos_osg_X[ii], fpos_osg_Y[1] - fpos_osg_Y[ii],
            fpos_osg_Z[1] - fpos_osg_Z[ii]);
2505      abstand_p1p2_osg = p1p2_osg.Magnitude();
2506
2507      //Abstand vom Punkt2 zum Punkt3
2508      UX_VECTOR3 p2p3_osg(fpos_osg_X[ii]-fpos_osg_X[j-1], fpos_osg_Y[ii]-fpos_osg_Y[j-1],
            fpos_osg_Z[ii]-fpos_osg_Z[j-1]);
2509      abstand_p2p3_osg = p2p3_osg.Magnitude();
2510
2511      //Abstand von p1p2 zu p2p3 wird berechnet
2512      ab_max_temp = abstand_p1p2_osg + abstand_p2p3_osg;
2513
2514      //Gesucht wird größter Betrag von p1p2 + p2p3,
2515      //randbedingung: der abstand von p1p2 und p2p3 muss aber größer als 1/3p1p3 sein
2516      //so wird gewährleistet dass der mittelpunkt auch der mittelpunkt ist
2517      if (ab_max_temp >= ab_max && abstand_p1p2_osg >= ab_max_control && abstand_p2p3_osg >=
            ab_max_control)
2518      {
2519        number = ii;
2520        ab_max = ab_max_temp;
2521        step2X3 = fpos_osg_X[ii];
2522        step2Y3 = fpos_osg_Y[ii];
2523        step2Z3 = fpos_osg_Z[ii];
2524      }
2525    }
2526
2527    //Werte zuweisen
2528    //USGA
2529    step4X4 = fpos_usg_X[1];
2530    step4Y4 = fpos_usg_Y[1];
2531    step4Z4 = fpos_usg_Z[1];
2532
2533    step6X4 = fpos_usg_X[k-1];
2534    step6Y4 = fpos_usg_Y[k-1];
2535    step6Z4 = fpos_usg_Z[k-1];
2536
2537
2538    UX_VECTOR3 p1p3_usg(fpos_usg_X[1] - fpos_usg_X[k-1], fpos_usg_Y[1] - fpos_usg_Y[k-1],
          fpos_usg_Z[1] - fpos_usg_Z[k-1]);
2539
2540    double abstand_p1p3_usg = p1p3_usg.Magnitude();
2541    double abstand_p1p2_usg;
2542    double abstand_p2p3_usg;
```

```
2543    ab_max=0;
2544    ab_max_temp=0;
2545    ab_max_control=abstand_p1p3_usg/3;
2546
2547    //Schleife um P2 zu suchen, bis array usg ende
2548    for ( ii = 2; ii < k; ii++)
2549    {
2550      //abstand vom punkt1 zu punk2
2551      UX_VECTOR3 p1p2_usg(fpos_usg_X[1] - fpos_usg_X[ii], fpos_usg_Y[1] - fpos_usg_Y[ii],
              fpos_usg_Z[1] - fpos_usg_Z[ii]);
2552      abstand_p1p2_usg = p1p2_usg.Magnitude();
2553
2554      //abstand vom punk2 zu punk3
2555      UX_VECTOR3 p2p3_usg(fpos_usg_X[ii]-fpos_usg_X[k-1], fpos_usg_Y[ii]-fpos_usg_Y[k-1],
              fpos_usg_Z[ii]-fpos_usg_Z[k-1]);
2556      abstand_p2p3_usg = p2p3_usg.Magnitude();
2557
2558      //Abstandbetrag p1p2+p2p3 berechnen
2559      ab_max_temp = abstand_p1p2_usg+abstand_p2p3_usg;
2560
2561      //Suche Mittelpunkt:
2562      // - Abstand p1p2+p2p3 am größten
2563      // - abstand p1p2> p1p3/3 -> garantie das der punkt aus der mitte kommt
2564      if (ab_max_temp >= ab_max && abstand_p1p2_usg >= ab_max_control && abstand_p2p3_usg >=
              ab_max_control)
2565      {
2566        number = ii;
2567        ab_max = ab_max_temp;
2568        step5X4 = fpos_usg_X[ii];
2569        step5Y4 = fpos_usg_Y[ii];
2570        step5Z4 = fpos_usg_Z[ii];
2571      }
2572    }
2573
2574    //Berechnung mit calcAxis
2575    //Berechne OSGA
2576    Achse.calcOSG(step1X3, step1Y3, step1Z3, step2X3, step2Y3, step2Z3, step3X3, step3Y3, step3Z3);
2577    //Berechne USGA
2578    Achse.calcUSG(step4X4, step4Y4, step4Z4, step5X4, step5Y4, step5Z4, step6X4, step6Y4, step6Z4);
2579    //Generieren der Strins in calcAxis
2580    Achse.stringOutput();
2581
2582    //Berechnung Instantaner Achsen
2583    //OnCalcAxis();
2584    OnCalcAxis2();
2585
2586    //Ausgabe an GUI
2587    UpdateDialog(output2);
2588
2589 }//ENDE
2590 ///////
2591
2592
2593
2594
2595 ////////////////////////////////////////////////////////////////////////////
2596 // SPEICHERN DIALOGE/ FUNKTIONEN
2597 //
2598 // Rohdaten speichern
2599 void CUsgDlg::OnSAVEDATA()
2600 {
2601    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
2602    //Rohdaten speichern
2603    CString m_sResults, m_strAusgabe;
2604    CFileDialog m_ldFile(FALSE);
2605    double test = 123.312;
2606    m_ldFile.m_ofn.lpstrDefExt = "dat";
2607    m_ldFile.m_ofn.lpstrFilter = "DAT␣File␣(*.dat)\0*.dat\0\0";
2608    // Dialogfeld Öffnen zeigen und Ergebnis auffangen, wenn "speichern" dann IDOK
2609    if (m_ldFile.DoModal() == IDOK)
2610    {
2611
2612      // Gewählten Dateinamen ermitteln
2613      m_sResults = m_ldFile.GetFileName();
2614      m_strAusgabe = "Datei␣'" + m_sResults + "'␣wurde␣erfolgreich␣abgespeichert";
2615      MessageBox(m_strAusgabe,"Meldung");
2616
2617      //Datei Erzeugen
2618      f_out = fopen(m_sResults,"w");
2619      //Speichern Punkt1 Taststift
2620      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X1, pmarker1Y1, pmarker1Z1,
              pmarker2X1, pmarker2Y1, pmarker2Z1);
2621      //Punkt 2
```

```
2622      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X2, pmarker1Y2, pmarker1Z2,
              pmarker2X2, pmarker2Y2, pmarker2Z2);
2623      //Punkt 3
2624      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X3, pmarker1Y3, pmarker1Z3,
              pmarker2X3, pmarker2Y3, pmarker2Z3);
2625      //Punkt 4
2626      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X4, pmarker1Y4, pmarker1Z4,
              pmarker2X4, pmarker2Y4, pmarker2Z4);
2627      //Punkt 5
2628      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X5, pmarker1Y5, pmarker1Z5,
              pmarker2X5, pmarker2Y5, pmarker2Z5);
2629      //Punkt 6
2630      fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pmarker1X6, pmarker1Y6, pmarker1Z6,
              pmarker2X6, pmarker2Y6, pmarker2Z6);
2631
2632      //Ausgabe osg
2633      int size_osg = pos_osg_X.GetSize();
2634      int size_usg = pos_usg_X.GetSize();
2635      int i;
2636      fprintf(f_out,"%d\n",size_osg);
2637      for(i=0;i<size_osg; i++)
2638      {
2639        fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pos_osg_X[i],pos_osg_Y[i],pos_osg_Z[
              i],fpos_osg_X[i],fpos_osg_Y[i],fpos_osg_Z[i]);
2640      }
2641      //Ausgabe USG
2642      fprintf(f_out,"%d\n",size_usg);
2643      for(i=0;i<size_usg; i++)
2644      {
2645        fprintf(f_out,"%6.6f␣␣%6.6f␣%6.6f␣%6.6f␣%6.6f␣%6.6f\n",pos_usg_X[i],pos_usg_Y[i],pos_usg_Z[
              i],fpos_usg_X[i],fpos_usg_Y[i],fpos_usg_Z[i]);
2646      }
2647      //Ausgabe Bein
2648      char leg = Achse.Leg;
2649      CString pName = m_Proband.m_pName;
2650      CString pAnamnese = m_Proband.m_pAnamnese;
2651      CString pBemerkung = m_Proband.m_pBemerkung;
2652      CString pGewicht = m_Proband.m_pGewicht;
2653      CString pBeins = m_Proband.m_pBeins;
2654      CString mDatum = m_Proband.m_mDatum;
2655      CString mBemerkung = m_Proband.m_mBemerkung;
2656      CString pCode = m_Proband.m_pCode;
2657
2658      pName.Replace("␣",",,,");
2659      pAnamnese.Replace("␣",",,,");
2660      pBemerkung.Replace("␣",",,,");
2661      pGewicht.Replace("␣",",,,");
2662      pBeins.Replace("␣",",,,");
2663      mBemerkung.Replace("␣",",,,");
2664      mDatum.Replace("␣",",,,");
2665      pCode.Replace("␣",",,,");
2666      fprintf(f_out,"%c\n",leg);
2667      fprintf(f_out,"%s\n",pName);
2668      fprintf(f_out,"%s\n",m_Proband.m_pCode);
2669      fprintf(f_out,"%s\n",m_Proband.m_pDatum);
2670      fprintf(f_out,"%s\n",pGewicht);
2671      fprintf(f_out,"%d\n",m_Proband.m_pGeschlecht);
2672      fprintf(f_out,"%s\n",m_Proband.m_pFuss);
2673      fprintf(f_out,"%s\n",pAnamnese);
2674      fprintf(f_out,"%s\n",m_Proband.m_pTritt);
2675      fprintf(f_out,"%s\n",pBeins);
2676      fprintf(f_out,"%s\n",pBemerkung);
2677      fprintf(f_out,"%s\n",mDatum);
2678      fprintf(f_out,"%s\n",mBemerkung);
2679
2680      fclose(f_out);
2681      // Dialogfeld aktualisieren/schliessen
2682      UpdateData(FALSE);
2683    }
2684
2685
2686
2687 }
2688 ////////////////////////////////////////////////////
2689 // Speichern als CSV
2690 void CUsgDlg::OnSave()
2691 {
2692    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
2693    CString daten_gesamt, m_sResults, m_strAusgabe;
2694    Achse.stringOutput();
2695    //sAusgabe = Achse.Ausgabe; //Ausgabestring aus calcAxis
2696    daten_gesamt = "\r";
2697    m_strProband = "\r";
2698    m_strProband = m_ProbTemp;
```

```
2699    //daten_gesamt = m_strProband +"\n" +sAchseFinite +"\n" + sAxis_inst+ "\n"+ sKoordinaten ;
2700    daten_gesamt = m_strProband +"\n" +sAchseFinite +"\n" + sAxis2 + "\n"+ sKoordinaten ;
2701
2702    //Dialog für file speichern wird geöffnet
2703    CFileDialog m_ldFile(FALSE);
2704    //m_ldFile(FALSE);
2705    // Anfangsverzeichnis initialisieren
2706    //m_ldFile.m_ofn.lpstrInitialDir = "D:\\temp\\";
2707
2708    //m_ldFile.m_ofn.lpstrDefExt = "csv";
2709    //m_ldFile.m_ofn.lpstrFilter = "CSV File (*.csv)\0*.csv\0\0";
2710    m_ldFile.m_ofn.lpstrDefExt = "csv";
2711    m_ldFile.m_ofn.lpstrFilter = "CSV File (*.csv)\0*.csv\0\0";
2712    // Dialogfeld Öffnen zeigen und Ergebnis auffangen, wenn "speichern" dann IDOK
2713    if (m_ldFile.DoModal() == IDOK)
2714    {
2715
2716      // Gewählten Dateinamen ermitteln
2717      m_sResults = m_ldFile.GetFileName();
2718      m_strAusgabe = "Datei '" + m_sResults + "' wurde erfolgreich abgespeichert";
2719      MessageBox(m_strAusgabe,"Meldung");
2720
2721      //Datei Erzeugen
2722      CFile Datei;
2723      Datei.Open(m_sResults,CFile::modeCreate|CFile::modeWrite);
2724      CArchive ar(&Datei,CArchive::store);
2725
2726      //Inhalt in Datei schreiben
2727      ar<<daten_gesamt;
2728      ar.Close();
2729      //Datei schliesen
2730      Datei.Close();
2731
2732      // Dialogfeld aktualisieren/schliessen
2733      UpdateData(FALSE);
2734    }
2735
2736 }
2737
2738 ////////////////////////////////
2739 // Datei Öffnen
2740 void CUsgDlg::OnOpen()
2741 {
2742    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
2743    CString m_sResults;
2744    int SIZE,i;
2745    char leg;
2746    double tempX1, tempY1, tempZ1, tempX2, tempY2, tempZ2;
2747    pos_osg_X.RemoveAll();
2748    pos_osg_Y.RemoveAll();
2749    pos_osg_Z.RemoveAll();
2750    pos_usg_X.RemoveAll();
2751    pos_usg_Y.RemoveAll();
2752    pos_usg_Z.RemoveAll();
2753    fpos_usg_X.RemoveAll();
2754    fpos_usg_Y.RemoveAll();
2755    fpos_usg_Z.RemoveAll();
2756    fpos_osg_X.RemoveAll();
2757    fpos_osg_Y.RemoveAll();
2758    fpos_osg_Z.RemoveAll();
2759
2760    //Dialog für file speichern wird geöffnet
2761    CFileDialog m_ldFile(TRUE);
2762    // Anfangsverzeichnis initialisieren
2763    //m_ldFile.m_ofn.lpstrInitialDir = "D:\\temp\\";
2764    m_ldFile.m_ofn.lpstrDefExt = "dat";
2765    m_ldFile.m_ofn.lpstrFilter = "DAT File (*.dat)\0*.dat\0\0";
2766    // Dialogfeld Öffnen zeigen und Ergebnis auffangen, wenn "speichern" dann IDOK
2767    if (m_ldFile.DoModal() == IDOK)
2768    {
2769      // Gewählten Dateinamen ermitteln
2770      m_sResults = m_ldFile.GetFileName();
2771      m_sFileName = m_sResults;
2772      MessageBox("Datei '"+m_sResults+"' wurde geladen. Parameter werden berechnet.");
2773
2774      //Daten auslesen
2775      f_in = fopen(m_sResults,"r");
2776
2777      //Einlesen Punkt1 Taststift
2778      fscanf(f_in,"%lf%lf %lf %lf %lf %lf",&pmarker1X1, &pmarker1Y1, &pmarker1Z1, &pmarker2X1, &
            pmarker2Y1, &pmarker2Z1);
2779      //Punkt 2
2780      fscanf(f_in,"%lf %lf %lf %lf %lf %lf",&pmarker1X2, &pmarker1Y2, &pmarker1Z2, &pmarker2X2, &
           pmarker2Y2, &pmarker2Z2);
```

```
2781      //Punkt 3
2782      fscanf(f_in,"%lf␣%lf␣%lf␣%lf␣%lf",&pmarker1X3, &pmarker1Y3, &pmarker1Z3, &pmarker2X3, &
              pmarker2Y3, &pmarker2Z3);
2783      //Punkt 4
2784      fscanf(f_in,"%lf␣%lf␣%lf␣%lf␣%lf",&pmarker1X4, &pmarker1Y4, &pmarker1Z4, &pmarker2X4, &
              pmarker2Y4, &pmarker2Z4);
2785      //Punkt 5
2786      fscanf(f_in,"%lf␣%lf␣%lf␣%lf␣%lf",&pmarker1X5, &pmarker1Y5, &pmarker1Z5, &pmarker2X5, &
              pmarker2Y5, &pmarker2Z5);
2787      //Punkt 6
2788      fscanf(f_in,"%lf␣%lf␣%lf␣%lf␣%lf",&pmarker1X6, &pmarker1Y6, &pmarker1Z6, &pmarker2X6, &
              pmarker2Y6, &pmarker2Z6);
2789
2790      //OSG Daten
2791      fscanf(f_in,"%d",&SIZE);
2792      for(i=0;i<SIZE;i++)
2793      {
2794        fscanf(f_in, "%lf␣%lf␣%lf␣%lf␣%lf",&tempX1,&tempY1,&tempZ1,&tempX2,&tempY2,&tempZ2);
2795        pos_osg_X.Add(tempX1);
2796        pos_osg_Y.Add(tempY1);
2797        pos_osg_Z.Add(tempZ1);
2798      }
2799
2800      //USG Daten
2801      fscanf(f_in,"%d",&SIZE);
2802      for(i=0;i<SIZE;i++)
2803      {
2804        fscanf(f_in, "%lf␣%lf␣%lf␣%lf␣%lf",&tempX1,&tempY1,&tempZ1,&tempX2,&tempY2,&tempZ2);
2805        pos_usg_X.Add(tempX1);
2806        pos_usg_Y.Add(tempY1);
2807        pos_usg_Z.Add(tempZ1);
2808        //fpos_usg_* muss nicht eingelesen werden da die Filterwerte neu berechnet werden
2809      }
2810
2811      leg = 'X';
2812      CString pName, pCode, pDatum, pGewicht, pFuss, pAnamnese,pTritt, pBeins, pBemerkung, mDatum,
              mBemerkung;
2813      int pGeschlecht;
2814      fscanf(f_in,"\n%c",&leg);
2815      fscanf(f_in,"%s\n",pName.GetBuffer(25));
2816      pName.ReleaseBuffer();
2817      pName.Replace(",,,","␣");
2818
2819      fscanf(f_in,"%s\n",pCode.GetBuffer(10));
2820      pCode.ReleaseBuffer();
2821      pCode.Replace(",,,","␣");
2822
2823      fscanf(f_in,"%s\n",pDatum.GetBuffer(12));
2824      pDatum.ReleaseBuffer();
2825      pDatum.Replace(",,,","␣");
2826
2827      fscanf(f_in,"%s\n",pGewicht.GetBuffer(5));
2828      pGewicht.ReleaseBuffer();
2829      pGewicht.Replace(",,,","␣");
2830
2831      fscanf(f_in,"%d\n",&pGeschlecht);
2832
2833      fscanf(f_in,"%s\n",pFuss.GetBuffer(10));
2834      pFuss.ReleaseBuffer();
2835      pFuss.Replace(",,,","␣");
2836
2837      fscanf(f_in,"%s\n",pAnamnese.GetBuffer(30));
2838      pAnamnese.ReleaseBuffer();
2839      pAnamnese.Replace(",,,","␣");
2840
2841      fscanf(f_in,"%s\n",pTritt.GetBuffer(20));
2842      pTritt.ReleaseBuffer();
2843      pTritt.Replace(",,,","␣");
2844
2845      fscanf(f_in,"%s\n",pBeins.GetBuffer(20));
2846      pBeins.ReleaseBuffer();
2847      pBeins.Replace(",,,","␣");
2848
2849      fscanf(f_in,"%s\n",pBemerkung.GetBuffer(250));
2850      pBemerkung.ReleaseBuffer();
2851      pBemerkung.Replace(",,,","␣");
2852
2853      fscanf(f_in,"%s\n",mDatum.GetBuffer(12));
2854      mDatum.ReleaseBuffer();
2855      mDatum.Replace(",,,","␣");
2856
2857      fscanf(f_in,"%s\n",mBemerkung.GetBuffer(250));
2858      mBemerkung.ReleaseBuffer();
2859      mBemerkung.Replace(",,,","␣");
```

```
2860
2861
2862      //Alle Daten eingelesen
2863      ////////////////////////////////////////////////////////////////
2864      //Abfrage Bein
2865      if(leg=='L') {
2866        Achse.Leg = 'L';  //Linkes Bein
2867        m_Proband.m_mFuss = 0;
2868      }
2869      else {
2870        Achse.Leg = 'R';       //Rechtes Bein
2871        m_Proband.m_mFuss = 1;
2872      }
2873      //Setzen der Probanden Daten
2874      m_Proband.m_pFuss = _T(pFuss);
2875      m_Proband.m_pBeins = _T(pBeins);
2876      m_Proband.m_mDatum = _T(mDatum);
2877      m_Proband.m_mBemerkung = _T(mBemerkung);
2878      m_Proband.m_pDatum = _T(pDatum);
2879      m_Proband.m_pBemerkung = _T(pBemerkung);
2880      m_Proband.m_pCode = _T(pCode);
2881      m_Proband.m_pGewicht = _T(pGewicht);
2882      m_Proband.m_pName = _T(pName);
2883      m_Proband.m_pTritt = _T(pTritt);
2884      m_Proband.m_pAnamnese = _T(pAnamnese);
2885      m_Proband.m_pGeschlecht = pGeschlecht;
2886
2887
2888      //Berechnung des Tibiakoordinaten Systems
2889      Achse.coordTibia(pmarker1X1, pmarker1Y1, pmarker1Z1, pmarker2X1, pmarker2Y1, pmarker2Z1,
2890            pmarker1X2, pmarker1Y2, pmarker1Z2, pmarker2X2, pmarker2Y2, pmarker2Z2,
2891            pmarker1X3, pmarker1Y3, pmarker1Z3, pmarker2X3, pmarker2Y3, pmarker2Z3,
2892            pmarker1X6, pmarker1Y6, pmarker1Z6, pmarker2X6, pmarker2Y6, pmarker2Z6);
2893      //Berechnen der OSGA aus den Maleollen punkten
2894      Achse.meshOSG(pmarker1X4, pmarker1Y4, pmarker1Z4,pmarker2X4, pmarker2Y4, pmarker2Z4,
2895          pmarker1X5, pmarker1Y5, pmarker1Z5, pmarker2X5, pmarker2Y5, pmarker2Z5);
2896
2897      //Aufruf der Funktion zum Ermitteln der Punkte zur Berechnung der Achsen
2898      OnCalculate();
2899      GetDlgItem(IDC_MESSUNG)->EnableWindow(TRUE);
2900    }
2901
2902  }
2903  // SPEICHERN
2904  ////////////////////////////////////////////////////////////////////
2905
2906  ////////////////////////////
2907  //Programm zurücksetzen aus Dialog
2908  void CUsgDlg::OnResetDialog()
2909  {
2910    OnReset();
2911  }
2912
2913
2914
2915  //
2916  // Programm zurücksetzen
2917  void CUsgDlg::OnReset()
2918  {
2919    m_Reset.DoModal();// aufrufen des reset dialog
2920    //wenn m_cReset = 2 wird nur mesung gelöscht
2921    if (m_Reset.m_cReset == 2) {
2922      //Mesung löschen
2923      GetDlgItem(IDC_MESSUNG)->EnableWindow(TRUE);
2924    }
2925    //wenn m_cReset = 1 alles löschen
2926    if (m_Reset.m_cReset == 1) {
2927      //Alles löschen
2928      GetDlgItem(IDC_MESSUNG)->EnableWindow(FALSE);
2929      Achse.~calcAxis();
2930    }
2931    UpdateDialog('r'); //reset
2932
2933
2934
2935
2936  }//OnReset()
2937  ////////////////////////
2938
2939
2940  //PROGRAMM BEENDEN
2941  void CUsgDlg::OnOK()
2942  {
2943    m_zebControl = 0; //Thread beenden
```

```
2944    CDialog::OnOK();
2945 }
2946 //Ende
2947 ////////////////////////////////////////////////////////////////////////
2948
2949 //Update Dialog
2950 void CUsgDlg::UpdateDialog(char output)
2951 {
2952
2953    //Zuweisen der Berechneten Werte
2954    sAchseFinite = Achse.Ausgabe;
2955    sOut_usgINC = Achse.sUSG_inc;
2956    sOut_usgDEV = Achse.sUSG_dev;
2957    sOut_usgLEVER = Achse.sUSG_lever;
2958    //sOut_usgANGLE = Achse.sUSG_winkel;
2959    sOut_osgINC = Achse.sOSG_inc;
2960    sOut_osgDEV = Achse.sOSG_dev;
2961    sOut_osgLEVER = Achse.sOSG_lever;
2962    sOut_osgANGLE = Achse.sOSG_winkel;
2963
2964
2965    if(output == 'm')      // Augab der Daten Achse aus 3 Punkten, aus CalcAxis2 //
2966    {
2967      //Ausgabe an Formular/Window
2968      GetDlgItem(IDC_USGINC)->SetWindowText(sUSGAV_INC);
2969      GetDlgItem(IDC_USGDEV)->SetWindowText(sUSGAV_DEV);
2970      GetDlgItem(IDC_USGHEBEL)->SetWindowText(sUSGAV_LEVER);
2971
2972      GetDlgItem(IDC_OSGINC)->SetWindowText(Achse.sOSG_inc_mesh);
2973      GetDlgItem(IDC_OSGDEV)->SetWindowText(Achse.sOSG_dev_mesh);
2974      GetDlgItem(IDC_OSGHEBEL)->SetWindowText(Achse.sLever_OSG_MM);
2975      //Drehwinlkel immer aus CalcAxis
2976      GetDlgItem(IDC_OSGDREH)->SetWindowText(sOut_osgANGLE);
2977      GetDlgItem(IDC_USGDREH)->SetWindowText(sOut_usgANGLE);
2978
2979      //Ausgabe STAB an GUI
2980      //Inklination
2981      if (m_iStabInc == 1) m_stcUsgInc.SetBkColor(RGB(51,102,102));      //Grün
2982      else if (m_iStabInc == 2) m_stcUsgInc.SetBkColor(RGB(102,153,153));    //Hellgrün
2983      else if (m_iStabInc == 3) m_stcUsgInc.SetBkColor(RGB(102,153,255));    //Blau
2984      else if (m_iStabInc == 4) m_stcUsgInc.SetBkColor(RGB(204,0,0));      //Rot
2985      //Deviation
2986      if (m_iStabDev == 1) m_stcUsgDev.SetBkColor(RGB(51,102,102));      //Grün
2987      else if (m_iStabDev == 2) m_stcUsgDev.SetBkColor(RGB(102,153,153));    //Hellgrün
2988      else if (m_iStabDev == 3) m_stcUsgDev.SetBkColor(RGB(102,153,255));    //Blau
2989      else if (m_iStabDev == 4) m_stcUsgDev.SetBkColor(RGB(204,0,0));      //Rot
2990      //Winkel
2991      if (m_iStabAng == 1) m_stcUsgAngle.SetBkColor(RGB(51,102,102));      //Grün
2992      else if (m_iStabAng == 2) m_stcUsgAngle.SetBkColor(RGB(102,153,153));    //Hellgrün
2993      else if (m_iStabAng == 3) m_stcUsgAngle.SetBkColor(RGB(102,153,255));    //Blau
2994      else if (m_iStabAng == 4) m_stcUsgAngle.SetBkColor(RGB(204,0,0));      //Rot
2995      //Hebel
2996      //Inklination
2997      if (m_iStabLever == 1) m_stcUsgLever.SetBkColor(RGB(51,102,102));      //Grün
2998      else if (m_iStabLever == 2) m_stcUsgLever.SetBkColor(RGB(102,153,153));    //Hellgrün
2999      else if (m_iStabLever == 3) m_stcUsgLever.SetBkColor(RGB(102,153,255));    //Blau
3000      else if (m_iStabLever == 4) m_stcUsgLever.SetBkColor(RGB(204,0,0));      //Rot
3001
3002    }
3003    if(output == 'i')
3004    {
3005      //Ausgabe an Formular/Window
3006      GetDlgItem(IDC_USGINC)->SetWindowText(sOut_usgINC);
3007      GetDlgItem(IDC_USGDEV)->SetWindowText(sOut_usgDEV);
3008      GetDlgItem(IDC_USGHEBEL)->SetWindowText(sOut_usgLEVER);
3009
3010      GetDlgItem(IDC_OSGINC)->SetWindowText(Achse.sOSG_inc_mesh);
3011      GetDlgItem(IDC_OSGDEV)->SetWindowText(Achse.sOSG_dev_mesh);
3012      GetDlgItem(IDC_OSGHEBEL)->SetWindowText(Achse.sLever_OSG_MM);
3013      //Drehwinlkel immer aus CalcAxis
3014      GetDlgItem(IDC_OSGDREH)->SetWindowText(sOut_osgANGLE);
3015      GetDlgItem(IDC_USGDREH)->SetWindowText(sOut_usgANGLE);
3016    }
3017
3018
3019    if(output == 'r')
3020    {
3021      //zurücksetzen
3022      //Ausgabe
3023      GetDlgItem(IDC_USGINC)->SetWindowText("--");
3024      GetDlgItem(IDC_USGDEV)->SetWindowText("--");
3025      GetDlgItem(IDC_USGHEBEL)->SetWindowText("--");
3026      GetDlgItem(IDC_OSGDREH)->SetWindowText("--");
3027        GetDlgItem(IDC_USGDREH)->SetWindowText("--");
```

```
3028      GetDlgItem(IDC_OSGINC)->SetWindowText("--");
3029      GetDlgItem(IDC_OSGDEV)->SetWindowText("--");
3030      GetDlgItem(IDC_OSGHEBEL)->SetWindowText("--");
3031    }
3032
3033
3034    UpdateData(FALSE); //Ergebnisse ans Dialogfeld ausgeben
3035
3036    OnDrawGraph();   //Graph zeichnen
3037  }
3038
3039
3040
3041  void CUsgDlg::On3punkt()
3042  {
3043    // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
3044
3045  }
3046
3047  void CUsgDlg::OnMittel()
3048  {
3049    // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
3050    output2 = 'm';
3051    UpdateDialog(output2);
3052  }
3053
3054  void CUsgDlg::OnPunkt3()
3055  {
3056    // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
3057    output2 = 'i';
3058    UpdateDialog(output2);
3059  }
3060
3061  void CUsgDlg::OnProperties()
3062  {
3063    // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
3064    //m_3D.ShowPropertyPages();
3065  }
3066
3067  void CUsgDlg::OnProperty()
3068  {
3069    m_3D.ShowPropertyPages();
3070  }
3071
3072  void CUsgDlg::OnView()
3073  {
3074    // TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung hier einfügen
3075      if(m_iTrackMode == 1)
3076    {
3077      m_iTrackMode = 2;
3078      m_3D.SetTrackMode(m_iTrackMode);
3079      GetDlgItem(IDC_VIEW)->SetWindowText("Rot");
3080    }
3081    else if(m_iTrackMode == 2)
3082    {
3083      m_iTrackMode = 3;
3084      m_3D.SetTrackMode(m_iTrackMode);
3085      GetDlgItem(IDC_VIEW)->SetWindowText("Pan");
3086    }
3087    else if(m_iTrackMode == 3)
3088    {
3089      m_iTrackMode = 1;
3090      m_3D.SetTrackMode(m_iTrackMode);
3091      GetDlgItem(IDC_VIEW)->SetWindowText("Zoom");
3092    }
3093  }
3094  //Diagramm Rotiert in die Sagitalebene
3095  void CUsgDlg::OnYz()
3096  {
3097    m_3D.SetTrackMode(4);
3098  }
3099  //Diagramm Rotiert in Transversalebene
3100  void CUsgDlg::OnXy()
3101  {
3102    m_3D.SetTrackMode(5);
3103  }
3104  //Diagramm Rotiert in Frontalebene
3105  void CUsgDlg::OnXz()
3106  {
3107    m_3D.SetTrackMode(6);
3108  }
3109  //Menu Zoom
3110  void CUsgDlg::OnAnsichtZoom()
3111  {
```

```
3112    m_iTrackMode = 1;
3113    m_3D.SetTrackMode(m_iTrackMode);
3114    GetDlgItem(IDC_VIEW)->SetWindowText("Zoom");
3115  }
3116  //Menu Rotation
3117  void CUsgDlg::OnAnsichtRot()
3118  {
3119    m_iTrackMode = 2;
3120    m_3D.SetTrackMode(m_iTrackMode);
3121    GetDlgItem(IDC_VIEW)->SetWindowText("Rot");
3122  }
3123  //Menu Pan
3124  void CUsgDlg::OnAnsichtPan()
3125  {
3126      m_iTrackMode = 3;
3127      m_3D.SetTrackMode(m_iTrackMode);
3128      GetDlgItem(IDC_VIEW)->SetWindowText("Pan");
3129  }
3130  //Button perspektive othogonal
3131  void CUsgDlg::OnView2()
3132  {
3133    if (m_iView2 == 0)
3134    {
3135      m_iView2 = 1;
3136      m_3D.SetProjection(m_iView2);
3137      GetDlgItem(IDC_VIEW2)->SetWindowText("orthographic");
3138    }
3139    else if (m_iView2 == 1)
3140    {
3141      m_iView2 = 0;
3142      m_3D.SetProjection(m_iView2);
3143      GetDlgItem(IDC_VIEW2)->SetWindowText("perspective");
3144    }
3145  }
3146  //Menu Graph
3147  void CUsgDlg::OnAnsichtSagitalebeneyz()
3148  {
3149    OnYz();
3150  }
3151  //Menu Graph
3152  void CUsgDlg::OnAnsichtTransversalebenexy()
3153  {
3154    OnXy();
3155  }
3156  //Menu Graph()
3157  void CUsgDlg::OnAnsichtFrontalebenexz()
3158  {
3159    OnXz();
3160  }
3161  //Menu Graph()
3162  void CUsgDlg::OnAnsichtEigenschaften()
3163  {
3164    OnProperty();
3165  }
3166  //Menu Graph(), kopiert die Abbildung in die Zwischenablage
3167  void CUsgDlg::OnAnsichtGraphkopieren()
3168  {
3169    OnCopy2CB();
3170  }
3171  void CUsgDlg::OnFootType()
3172  {
3173    if(m_iFootType==4)
3174    {
3175      m_iFootType=1;
3176      GetDlgItem(IDC_FOOTTYPE)->SetWindowText("Fläche");
3177    }
3178    else if(m_iFootType==1)
3179    {
3180      m_iFootType=2;
3181      GetDlgItem(IDC_FOOTTYPE)->SetWindowText("2D Fuss XY");
3182    }
3183    else if(m_iFootType==2)
3184    {
3185      m_iFootType=3;
3186      GetDlgItem(IDC_FOOTTYPE)->SetWindowText("2D Fuss YZ");
3187    }
3188    else if(m_iFootType==3)
3189    {
3190      m_iFootType=4;
3191      GetDlgItem(IDC_FOOTTYPE)->SetWindowText("Gitter");
3192    }
3193
3194    //Funktion Graph zeichnen
3195    OnDrawGraph();
```

```
3196 }
3197 //button Copy2CB
3198 void CUsgDlg::OnCopy2CB()
3199 {
3200   m_3D.CopyToClipboard();
3201   MessageBox("Schaubild␣wurde␣in␣die␣Zwischenablage␣kopiert!",MB_OK);
3202 }
3203
3204 ////////////////////////////////
3205 // String normalization (remove all unused characters)
3206 static char *NormalizeString(char *s)
3207 {
3208   while (*s == '␣') s++;
3209
3210   ULONG l = strlen(s);
3211   for (ULONG i=(l-1); i>0; i--)
3212     if (s[i] < 33) s[i] = 0;
3213     else break;
3214
3215   return s;
3216 }
3217
3218 static void ParseFloat(char *s, float *a, float *b, float *c)
3219 {
3220   int i, j;
3221   int x, y, z;
3222
3223   s = NormalizeString(s);
3224   x = 0; y = -1; z = -1;
3225   j = (int)strlen(s);
3226   for (i=0; i<j; i++)
3227   {
3228     if ((s[i] == '␣') || (s[i] == '\n'))
3229     {
3230       s[i] = 0;
3231       if (y == -1) y = i+1;
3232       else
3233       if (z == -1) z = i+1;
3234     }
3235   }
3236   *a = (float)atof(&s[x]);
3237   *b = (float)atof(&s[y]);
3238   *c = (float)atof(&s[z]);
3239 }
3240
3241
3242
3243 ////////////////////////////////
3244
3245
3246
3247
3248 //Funktion zum zeichnen des Graphen
3249 void CUsgDlg::OnDrawGraph()
3250 {
3251   float f_osg1x, f_osg1y, f_osg1z, f_usg1x, f_usg1y, f_usg1z;
3252   int m_iDrawLeg;
3253   int m_iFootLenght;
3254
3255   //Fusslänge in integer konvertieren
3256   m_iFootLenght = atoi(m_Proband.m_pFuss);
3257   //linkes oder rechtes Bein
3258   if(m_Proband.m_mFuss == 1) m_iDrawLeg=-1; //rechts
3259   if(m_Proband.m_mFuss == 0) m_iDrawLeg=1;    //links
3260
3261
3262   m_3D.ClearGraph();
3263
3264   //Beschriftung Graph
3265   //m_3D.SetCaption(m_sFileName);
3266   m_3D.SetCaption(m_Proband.m_pCode);
3267   //Osg Achse zeichen
3268   m_3D.AddElement();
3269   m_3D.SetElementLineColor(0, RGB(255,0,0));
3270   m_3D.SetElementType(0, 0);
3271   m_3D.SetElementLineWidth(0,3);
3272
3273   //berechnen der Punkte der OSGA
3274   for(int i = -12; i <=4; i++)
3275   {
3276     f_osg1x = Achse.mOSG_AP.X + (i * 10 * Achse.mOSG_RV.X);
3277     f_osg1y = Achse.mOSG_AP.Y + (i * 10 * Achse.mOSG_RV.Y);
3278     f_osg1z = Achse.mOSG_AP.Z + (i * 10 * Achse.mOSG_RV.Z);
3279
```

```
3280      m_3D.PlotXYZ(f_osg1x, f_osg1y, f_osg1z,0);
3281    }
3282
3283
3284    //Malleolen zeichnen
3285    m_3D.AddElement();
3286    m_3D.SetElementPointColor(1, RGB(0,0,0));
3287    m_3D.SetElementPointSize(1,5);
3288    m_3D.SetElementType(1, 1);
3289    m_3D.PlotXYZ(Achse.ML.X, Achse.ML.Y, Achse.ML.Z, 1);
3290    m_3D.PlotXYZ(Achse.MM.X, Achse.MM.Y, Achse.MM.Z, 1);
3291
3292    //USG Achse zeichnen
3293    m_3D.AddElement();
3294    m_3D.SetElementLineColor(2, RGB(0,51,255));
3295    m_3D.SetElementType(2,0);
3296    m_3D.SetElementLineWidth(2,3); ///
3297
3298    for(int ii = -50; ii < 50; ii++)
3299    {
3300      f_usg1x = m_usg_APX +(ii * 10 * m_usg_RVX);
3301      f_usg1y = m_usg_APY + (ii * 10 * m_usg_RVY);
3302      f_usg1z = m_usg_APZ + (ii * 10 * m_usg_RVZ);
3303
3304      m_3D.PlotXYZ(f_usg1x, f_usg1y, f_usg1z, 2);
3305    }
3306
3307    //fuss zeichnen
3308    //Variablen definieren
3309    FILE *f_in;
3310    CString v;
3311    CArray<double, double>m_X;
3312    CArray<double, double>m_Y;
3313    CArray<double, double>m_Z;
3314    CArray<double, double>m_nX;
3315    CArray<double, double>m_nY;
3316    CArray<double, double>m_nZ;
3317    char buffer[512];
3318    char *str;
3319    int count=1;
3320    int count2=3;
3321    int c1,c2,c3;
3322    float x,y,z, p1x, p1y, p1z, p2x,p2y,p2z,p3x,p3y,p3z,p1nx,p1ny,p1nz,p2nx,p2ny,p2nz,p3nx,p3ny,
          p3nz;
3323
3324    //rotation
3325    float rot=-85;
3326    float rot1 = 3.141592/180* rot;
3327
3328    //eichung streckung
3329    float ex=3.8;
3330    float ey=3.8;
3331    float ez=3.8;
3332    //streckung (angabe in mm, fusslänge wird in mm angegeben)
3333    float s_x=m_iFootLenght;
3334    float s_y=m_iFootLenght;
3335    float s_z=m_iFootLenght;
3336
3337    //fuss spiegeln bei rechtem fuss
3338    ex=ex*m_iDrawLeg;
3339
3340
3341
3342    //File einlesen
3343    ////////////////////////////
3344    //öffnen absolut pfad
3345    ////////////////////////////
3346    f_in = fopen("c:\\programme\\zebusg\\man_foot_l2.obj","r"); //cube.obj
3347
3348    while(!feof(f_in))
3349    {
3350      fgets(buffer, 512, f_in);
3351      str = NormalizeString(buffer);
3352
3353      if (str[0] == '#' || str[0] == 'g' || str[0] == 0) continue;
3354          for (i=0; i<strlen(str); i++)
3355      {
3356        if (str[i] == ' ')
3357        {
3358          str[i] = 0;
3359          if (!strcmp(str, "v")) // Vertex coordinates
3360          {
3361            float x1,y1,z1;
3362
```

```
3363            ParseFloat(&str[i+1], &x, &y, &z);
3364            //rotation um z achse
3365            x1 = x*cos(rot1) + y*sin(rot1);
3366            y1 = x*(-sin(rot1)) + y*cos(rot1);
3367
3368            //streckung
3369            if(m_iFootType == 3) x1 = 0;     //2D Darstellung YZ Ebene
3370            else x1 = (x1+0.08)*s_x*ex;
3371            y1 = (y1+0.119)*s_y*ey;
3372            //Darstellung Fuss oder 2D Fuss
3373            if(m_iFootType == 2) z1 = 0;  //2D Darstellung  XY Ebene
3374            else z1 = (z+0.9)*s_z*ez;
3375
3376            m_X.Add(x1);
3377            m_Y.Add(y1);
3378            m_Z.Add(z1);
3379
3380            count++;
3381
3382            break;
3383          }
3384
3385          else
3386          if (!strcmp(str, "vn")) // Normal coordinates
3387          {
3388            ParseFloat(&str[i+1], &x, &y, &z);
3389            m_nX.Add(x);
3390            m_nY.Add(y);
3391            m_nZ.Add(z);
3392            break;
3393          }
3394
3395          else
3396          if (!strcmp(str, "f"))
3397          {
3398            ParseFloat(&str[i+1], &x, &y, &z);
3399            c1 = (int)x;
3400            c2 = (int)y;
3401            c3 = (int)z;
3402
3403            m_3D.AddElement();
3404            m_3D.SetElementLineColor(count2,RGB(150,150,150));
3405            m_3D.SetElementLineWidth(count2,1);
3406
3407            if (m_iFootType==1 || m_iFootType==2 || m_iFootType==3) m_3D.SetElementType(count2, 3);
                    //Fläche 3D, 2D
3408            if (m_iFootType==4) m_3D.SetElementType(count2,0);     //Gitternetz
3409
3410            m_3D.SetElementPointSize(count2,5);
3411            m_3D.SetElementSurfaceFill(count2,1);
3412            m_3D.SetElementSurfaceFlat(count2,0);
3413            m_3D.SetElementLightingAmbient(count2,20);//grundhelligkeit
3414            m_3D.SetElementLightingDiffuse(count2,0);//beleuchtung
3415            m_3D.SetElementLightingSpecular(count2,0);//glänzend
3416            m_3D.SetElementLight(count2,1);
3417            m_3D.SetLightCoordinates(count2,-350,250,300);  //37,100,46
3418            m_3D.SetElementMaterialEmission(count2,10);
3419            m_3D.SetElementMaterialDiffuse(count2,0);
3420            m_3D.SetElementMaterialSpecular(count2,0);
3421            m_3D.SetElementMaterialAmbient(count2,0);
3422            m_3D.SetElementMaterialEmission(count2,10);
3423
3424            p1x = m_X[c1-1];
3425            p1y = m_Y[c1-1];
3426            p1z = m_Z[c1-1];
3427            p2x = m_X[c2-1];
3428            p2y = m_Y[c2-1];
3429            p2z = m_Z[c2-1];
3430            p3x = m_X[c3-1];
3431            p3y = m_Y[c3-1];
3432            p3z = m_Z[c3-1];
3433
3434            //Normalen Vertex Vectoren
3435            p1nx = m_nX[c1-1];
3436            p1ny = m_nY[c1-1];
3437            p1nz = m_nZ[c1-1];
3438            p2nx = m_nX[c2-1];
3439            p2ny = m_nY[c2-1];
3440            p2nz = m_nZ[c2-1];
3441            p3nx = m_nX[c3-1];
3442            p3ny = m_nY[c3-1];
3443            p3nz = m_nZ[c3-1];
3444
3445            m_3D.PlotXYZ(p1x,p1y,p1z,count2);
```

```
3446            m_3D.PlotXYZ(p2x,p2y,p2z,count2);
3447            m_3D.PlotXYZ(p3x,p3y,p3z,count2);
3448
3449            //test normalen vektor
3450              float v1x,v1y,v1z,v2x,v2y,v2z;
3451              float nx,ny,nz,nnx,nny,nnz;
3452              float vLen;
3453
3454              // Calculate vectors
3455              v1x = p1x - p2x;
3456              v1y = p1y - p2y;
3457              v1z = p1z - p2z;
3458
3459              v2x = p2x - p3x;
3460              v2y = p2y - p3y;
3461              v2z = p2z - p3z;
3462
3463              // Get cross product of vectors
3464              nx = (v1y * v2z) - (v1z * v2y);
3465              ny = (v1z * v2x) - (v1x * v2z);
3466              nz = (v1x * v2y) - (v1y * v2x);
3467
3468              // Normalise final vector
3469              vLen = sqrt( (nx * nx) + (ny * ny) + (nz * nz) );
3470              nnx = nx / vLen;
3471              nny = ny / vLen;
3472              nnz = nz / vLen;
3473
3474
3475            if(m_iFootType==1)
3476            {
3477
3478              m_3D.PlotXYZ(p1nx,p1ny,p1nz,count2);
3479              m_3D.PlotXYZ(p2nx,p2ny,p2nz,count2);
3480              m_3D.PlotXYZ(p3nx,p3ny,p3nz,count2);
3481            }
3482            else if(m_iFootType==2)
3483            {
3484              m_3D.SetElementLight(count2,0);
3485              m_3D.PlotXYZ(0,0,1,count2);
3486              m_3D.PlotXYZ(0,0,1,count2);
3487              m_3D.PlotXYZ(0,0,1,count2);
3488            }
3489            else if(m_iFootType==3)
3490            {
3491              m_3D.SetElementLight(count2,0);
3492              m_3D.PlotXYZ(0,1,0,count2);
3493              m_3D.PlotXYZ(0,1,0,count2);
3494              m_3D.PlotXYZ(0,1,0,count2);
3495            }
3496            else if(m_iFootType==4)
3497            {
3498              m_3D.SetElementLight(count2,0);
3499            }
3500
3501            count2++;
3502
3503            break;
3504          }
3505        }
3506        else continue;
3507      }///////
3508
3509    }//######
3510
3511
3512    m_3D.SetRange(-150,150,0,300,0,300);
3513
3514  }
```

G Funktionen und Bedienung weiterer Benutzeroberflächen zur Steuerung des Messprogramms

G.1 Dialog - Probandendaten

Im Dialog „Probandendaten“ werden alle Stammdaten zum Probanden aber auch Angaben zur Messung eingegeben werden. Diese Daten können jederzeit durch einen erneuten Aufruf des Dialogs eingesehen oder verändert werden. Die exportierte CSV Datei enthält die Probanden- und Messdaten einer durchgeführten Messung . Wird im Feld *Fusslänge* die Länge in *mm* eingetragen, wird nach erfolgreich durchgeführter Messung ein dreidimensionales Fußmodell in der grafischen Darstellung der Ergebnisse im Hauptdialog angezeigt. Einziges Pflichtfeld ist das Feld *Fuß* im Bereich *Messung*. Wird dort keine Angabe gemacht, bleibt der Wert *rechts* voreingestellt und alle Ergebnisse werden bezogen auf eine rechten Fuß in einem *Rechtssystem* berechnet. Abbildung 69 auf Seite 188 zeigt den Dialog „Probandendaten“.

G.2 Dialog - Koordinaten festlegen

Der Dialog *Koordinaten festlegen* besteht eigentlich aus 6 Dialogen. In vier dieser Dialoge werden Punkte, die zur Berechnung des *Tibiakoordinatensystems* benötigt werden, aufgezeichnet. Die Berechnung erfolgt anschließend wie in Kapitel 3.4.3 auf Seite 25 beschrieben. In den weiteren Dialogen werden zusätzlich noch jeweils ein Punkt auf Malleolus lateralis und Malleolus medialis erfasst. Alle sechs Oberflächen bestehen aus den selben Schalt- und Ausgabeflächen.

1 Kontrollanzeige für die Ultraschallsignale von Marker 1 und Marker 2. Diese Schaltflächen sind rot wenn kein Ultraschallsignal empfangen werden kann. Sie nehmen die Farbe grün an wenn ein störungsfreies Signal empfangen wird.

2 Schaltfläche für das Aufzeichnen von Ultraschallsignal bzw. anatomisch markanter Punkte. Diese Schaltfläche wird erst aktiv, wenn von beiden Ultraschallmarkern ein Signal empfangen wird. Wird diese Schaltfläche mehrmals gedrückt, wird das vorangegangene Datensatzpaar überschrieben. Der am Taststift angebrachte Taster *T1* übernimmt ebenfalls diese Funktion.

3 Schaltfläche *weiter*, schließt den aktuellen und öffnet den darauffolgenden Dialog. Diese Schaltfläche wird erst aktiv wenn bereits ein Ultraschallsignal aufgezeichnet wurde. Diese Funktion kann auch vom am Taststift angebrachten Taster *T2* gesteuert werden.

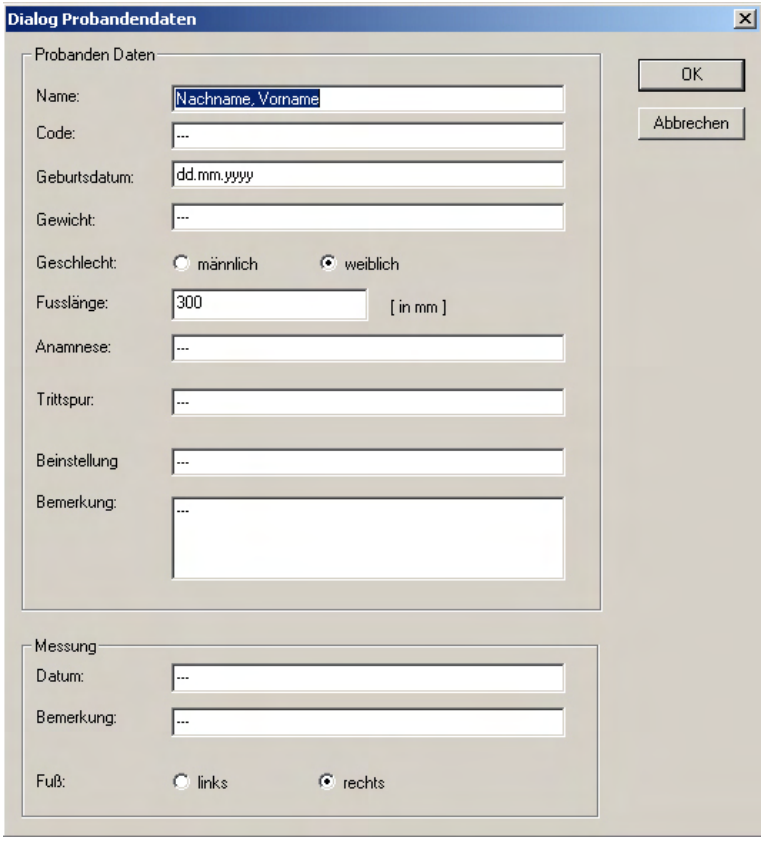

Abbildung 69: Dialog Probandendaten. Einziges Pflichtfeld ist das Feld „Fuß“ im Bereich „Messung“. Voreingestellt ist *rechts*. Alle hier eingegebenen Daten werden in der exportierten *CSV* Datei abgespeichert.

4 Schaltfläche *Abrechen* dient zum Abbruch der Routine *Koordinaten festlegen*.

5 Programm Erklärung und Messanweisungen.

6 Grafische Erläuterung der Messanweisungen.

G.3 Dialog - Messung

Der Dialog *Messung* setzt sich aus einem Dialog zur Bestimmung der OSGA und einem weiteren zur Bestimmung der USGA Parameter. Beide Dialoge sind annähernd identisch. Sie verfügen über die selben Schalt- und Ausgabeflächen, unterscheiden sich aber in ihren Messanweisungen.

1 Kontrollanzeige des Ultraschallsignals von Marker 3 im OSGA Dialog, bzw. des Marker 4 im USGA Dialog.

Abbildung 70: Dialog Koordinatensystem festlegen. Abgebildet ist der erste von insgesamt sechs Dialogen die zur Erfassung anatomischer Punkte dienen.

2 Schaltfläche zum starten bzw. stoppen des Dateneinzugs des jeweiligen Messvorgangs.

3 Schaltfläche *weiter* schließt den Messdialog zur Berechnung der OSGA Parameter und öffnet den USGA Dialog.

4 Schaltfläche *Abbrechen* bricht den Messvorgang ab. Es werden keine Sprunggelenkachsenparameter berechnet.

5 Messanweisungen.

6 Grafische Erläuterung der Messanweisungen.

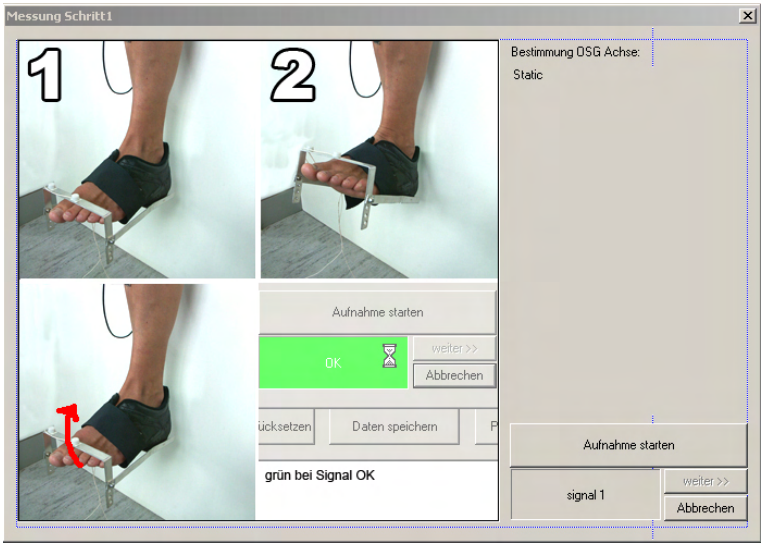

Abbildung 71: Dialog, Messung. Abgebildet ist der erste von zwei Dialogen die zur Erfassung der Daten zur Berechnung der Parameter der OSGA und USGA benötigt werden. Beide Dialoge sind identisch aufgebaut, verfügen also über die selben Schalt- und Ausgabeflächen.

H Taststift zur Erfassung anatomischer Punkte

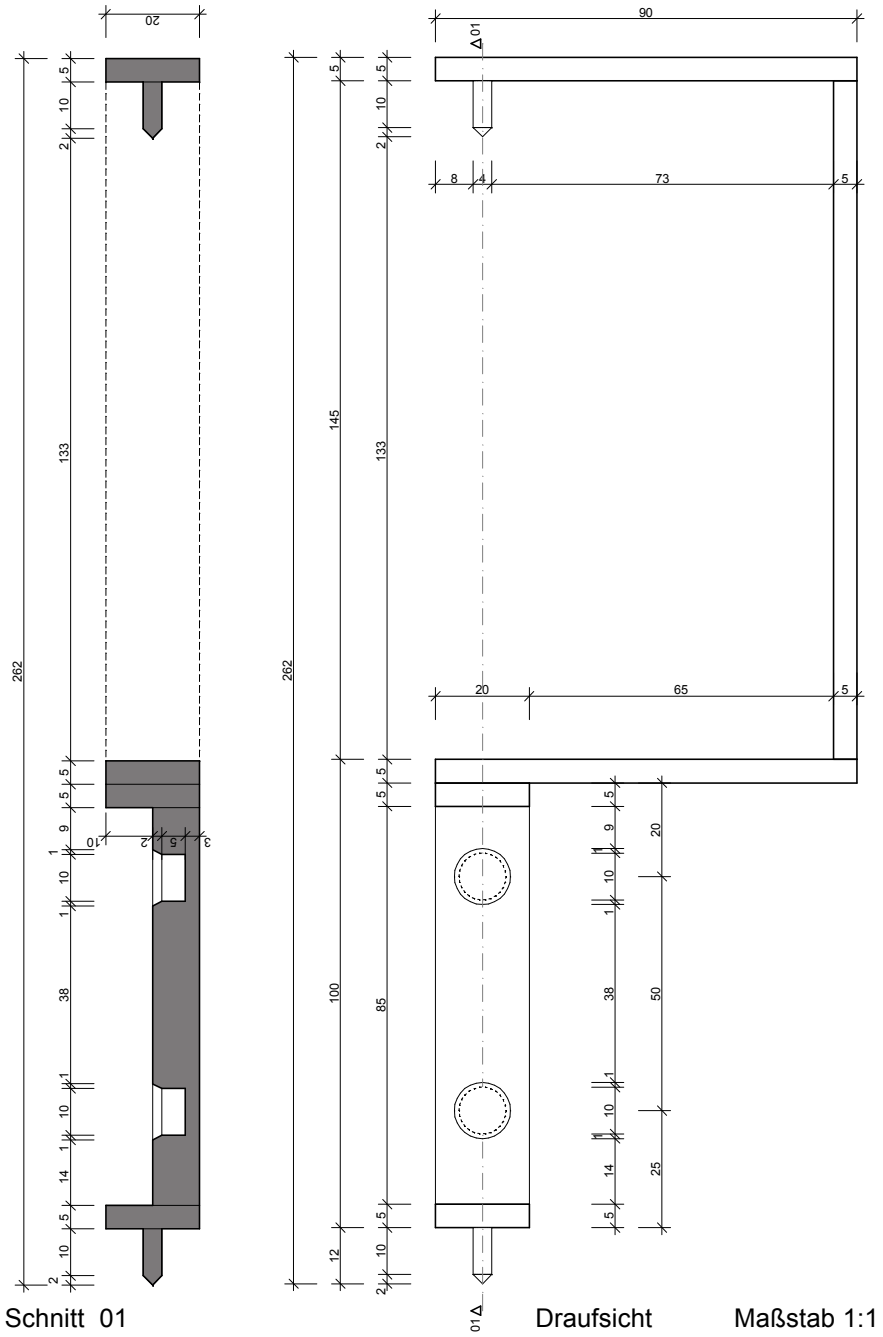

Abbildung 72: Konstruktionsplan des Taststiftes

I Quellcode Auszug aus dem 'Programm Modell virtuell'

```
1  CString m_sResults, m_strAusgabe;
2 CFileDialog m_ldFile(FALSE);
3   m_ldFile.m_ofn.lpstrDefExt = "dat";
4   m_ldFile.m_ofn.lpstrFilter = "DAT File (*.dat)\0*.dat\0\0";
5   // Dialogfeld ÷ffnen zeigen und Ergebnis auffangen, wenn "speichern" dann IDOK
6   if (m_ldFile.DoModal() == IDOK)
7   {
8
9     // Gew?hlten Dateinamen ermitteln
10    m_sResults = m_ldFile.GetFileName();
11    m_strAusgabe = "Datei '" + m_sResults + "' wurde erfolgreich abgespeichert";
12    MessageBox(m_strAusgabe,"Meldung");
13    //Datei Erzeugen
14    f_out = fopen(m_sResults,"w");
15
16    float i;
17    float x1,y1,x2,y2,x3,y3;
18    int max = 100;          //maximaler Abstand von der Rotationsachse
19    float winkel_max = 10;
20    float winkel;
21    for(winkel = 0.5; winkel<winkel_max; winkel=winkel+0.5) {
22
23      for(int n = 1; n<=max; n++) {
24        i= n*0.1;    //F?ngt bei 0.1 an
25
26        //berechnung erster Punkt
27        y1 = i*cos(0);
28        x1 = i*sin(0);
29        //berechnung 2. punkt
30        y2 = i*cos((winkel/2)*3.141592654/180);
31        x2 = i*sin((winkel/2)*3.141592654/180);
32        //berechnung 3. punkt
33        y3 = i*cos(winkel*3.141592654/180);
34        x3 = i*sin(winkel*3.141592654/180);
35        //hier schleife f¸r berechnung
36        Achse.calcOSG(x1,y1,0,x2,y2,0,x3,y3,0);
37
38        //Speichern
39        fprintf(f_out,"%6.1f  %6.6f %6.1f %6.6f %6.6f %6.6f %6.6f %6.6f %6.6f\n",
40          winkel, Achse.OSG_winkel, i,
41          Achse.OSG_APX, Achse.OSG_APY, Achse.OSG_APZ,
42          Achse.OSG_RVX, Achse.OSG_RVY, Achse.OSG_RVZ);
43
44      }
45    }
46    fclose(f_out);
47    // Dialogfeld aktualisieren/schliessen
48    UpdateData(FALSE);
49  }
```

J Lebenslauf

Harald Hochwald

Persönliche Daten

06.03.1970	geboren in Geislingen an der Steige

Schulbildung

1976-1980	Friedrich-Kammerer Schule (Grundschule), Ehningen
1980-1986	Theodor-Heuss-Realschule, Gärtringen
1992-1995	Kolleg am Stifts-Gymnasium, Sindelfingen, Abschluss Abitur

Berufsausbildung

1986-1989	Ausbildung zum Fernmeldeelektroniker, Telekom Stuttgart

Zivildienst

1990-1992	Rettungssanitäter, Rotes Kreuz, Sindelfingen

Studium

1997-2003	Studium an der Universität Stuttgart, Fachrichtung Physik und Sport, LA/ MA
2002	Abschluss mit dem ersten Staatsexamen
2003	Abschluss Magister Atrium

Berufstätigkeit

seit 1999	Geschäftsführender Gesellschafter der fastmovinbytes GbR
2001-2002	Wissenschaftliche Hilfskraft am Institut für Sportwissenschaft derUniversität Stuttgart
seit 2002	Wissenschaftlicher Angestellter am Institut für Sportwissenschaft der Universität Stuttgart

K Eidesstattliche Erkärung

Ich versichere an Eides statt, dass ich die vorliegende Arbeit selbständig ohne unerlaubte Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, habe ich als solche einzeln kenntlich gemacht.

Danksagung

Zum Schluss möchte ich allen danken, die mich bis hierher auf verschiedene und vielfältige Weise begleitet und unterstützt haben.
Zunächst gilt mein Dank meinem Doktorvater Herrn Professor Wilfried Alt, der mir diese Arbeit ermöglichte und mich in jeder Phase der Arbeit sehr sachkundig und richtungsweisend begleitete. Ebenfalls möchte ich mich bei Herrn Professor Wagner für die Übernahme des Zweitgutachten bedanken. Bei Herrn Professor Schoder möchte ich mich für die Unterstützung in der Anfangsphase meines Promotionsvorhabens bedanken.
Dem Arbeitsbereich Biomechanik des Instituts für Sportwissenschaft der Universität Stuttgart möchte ich herzlich für die freundliche und hilfsbereite Arbeitsatmosphäre danken. Danke, Julia, Wowo, Steffen, Dieter und Rolf. Besonderen Dank an Gunter für seine Unterstützung vor allem in der Anfangsphase der Arbeit.
Ich möchte mich bei Herrn Doktor Mauch und Herrn Jäger von der Sportklinik Bad Canstatt für die Zusammenarbeit und Hilfe bei der Erstellung der MRT-Aufnahmen bedanken. Ein weiterer Dank geht an Herrn Brunner und Herrn Abrecht von der Firma Zebris® für die unkomplizierte Unterstützung ihre Hardware betreffend. Den Mechanik Werkstätten der Universität Stuttgart danke ich für die Anfertigung des Taststiftes.
Danke Syn und Franziska für die Durchsicht des Manuskripts, danke an Harald und Steffen für die Korrektur des englischen Abtracts.
Ein weiterer Dank geht an alle anderen, die mich mit Tips und Anregungen versorgt haben bzw. meine Arbeit in irgendeiner Form unterstützt haben: Volker, Jan, Fabi, Andy, Christoph, Claus, Maren, Benni, allen Unbekannten aus verschiedensten Internet C^{++} sowie LaTeX Foren und alle die ich jetzt vergessen habe.
Anschließend ist es mein Wunsch denen zu danken, die mich seit langer Zeit begleiten. Meiner Freundin Jutta, meinem Studienfreund Syn (danke für die fachlichen und weniger fachlichen Diskussionen), sowie meinem Freund und Bandkollegen Flo. Zuletzt möchte ich mich ganz besonders bei meinen Eltern bedanken die mich immer, in jeder Phase meines Lebens auf jede erdenkliche Art unterstützt haben.

Die Arbeit wurde im Rahmen des Projekts „Entwicklung und Anwendung eines Verfahrens zur Bestimmung der Sprunggelenkachsen in vivo“ (VF 0404/01/65/04-05) erstellt, das vom BISP gefördert wurde.