Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Kollisionsberechnung für die Bewegungssteuerung mobiler robotischer Manipulatoren

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Thomas Kiesling

Betreuer: Dr.-Ing. Steffen Müller,
FG Neuroinformatik und Kognitive Robotik

Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 15.08.2019 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

Erklärung: „Hiermit versichere ich, dass ich diese wissenschaftliche Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet.“

Ilmenau, 15.08.2019 .....

Thomas Kiesling

Inhaltsverzeichnis

Abkürzungsverzeichnis

AABB	Axis Aligned Bounding Box
FCL	Flexible Collision Library
k-DOP	k-Discrete Oriented Polygon
MIRA	Middleware for Robotic Applications
NDT	Normal Distribution Transform
OBB	Oriented Bounding Box
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SONARO	Smarte Objektübernahme und -übergabe für die nutzerzentrierte mobile Assistenzrobotik
TIAGo	Roboterplattform der Firma PAL Robotics

Kapitel 1

Einleitung

Der Einsatz von Robotern ist aus der heutigen Zeit nicht mehr wegzudenken. Sie können in unterschiedlichsten Situationen und Umgebungen eingesetzt werden und werden in verschiedensten Disziplinen genutzt. Die Wissenschaft, die sich mit der Erstellung, Gestaltung und Programmierung von Robotern befasst, ist die Robotik. Diese Wissenschaft integriert unterschiedlichste Ansätze aus anderen Disziplinen miteinander, unter anderem aus den Bereichen Maschinenbau, Elektrotechnik, Mathematik und Informatik.

> *„Die Assoziationen, die der Begriff Robotik in der Öffentlichkeit manches Mal auslöst, orientiert sich dabei oft noch an den klassischen Robotersystemen. Ein weit verbreitetes Bild ist oft noch immer eine große Produktionsanlage, in der mehrere Schwerlastroboter hochpräzise, aufeinander abgestimmte und unveränderliche Aufgaben ausführen.“* ([Klein et al., 2018, S. 2])

Die Roboter, die mit diesem Bild assoziiert sind, sind Industrieroboter, die in der Produktion eingesetzt werden ([Klein et al., 2018]) . Hier werden Roboter genutzt, um die Menschen in ihren Tätigkeiten zu unterstützen bzw. ihre Tätigkeit in Gefahrenbereichen oder in schwierigen Umgebungen zu übernehmen. Industrieroboter sind so konzipiert, dass sie ihren programmierten Bewegungsablauf autonom durchführen können. Diese Roboter arbeiten in speziell abgesteckten und begrenzten Arbeitsbereichen, in denen sich keine Personen aufhalten dürfen. Somit können keine Personen aufgrund des Einsatzes eines Roboters zu Schaden kommen. Weitere Einsatzmöglichkeiten für Roboter sind unter anderem im Bereich der Wissenschaft, bei denen sie zu

Forschungs- und Experimentierzwecken eingesetzt werden, sowie in der Gesellschaft, z.B. als Serviceroboter.
Ein weiterer Bereich, in dem Roboter mittlerweile vermehrt eingesetzt werden, ist das Gesundheitswesen. Hier können Roboter in verschiedenen Bereichen zum Einsatz kommen, wie zum Beispiel zur Rehabilitation, zur Unterstützung des (Pflege-)Personals oder auch im privaten Bereich ([Klein et al., 2018]). In der m&i-Fachklinik Bad Liebenstein wurde zum Beispiel der Reha-Roboter Roreas getestet, der bei Patienten, die einen Schlaganfall erlitten, zum Einsatz kommt und sie im Laufen lernen unterstützen soll ([Gross et al., 2017], [Hillienhof, 2016]). Um so einen Einsatz zu ermöglichen, werden Sicherheitszäune bzw. -bereiche, wie sie es bei Industrierobotern gibt, aufgehoben. Mensch und Roboter interagieren dann miteinander. Bei einer sogenannten Mensch-Roboter-Kollaborationen arbeiten Menschen und Roboter im gleichen Raum ([Onnasch et al., 2016]). Gerade dann spielen Sicherheitsaspekte eine wesentliche Rolle, da eine Gefährdung des Menschens immer ausgeschlossen werden soll. Ist der Arbeitsbereich vermischt, so besteht auch immer die Gefahr, dass unvorhergesehene Hindernisse im Weg des Roboters stehen könnten, ein vergessener Rollwagen im Flur zum Beispiel oder auch ein Patient. Damit so eine Gefahrensituation gar nicht erst entsteht, besteht insbesondere bei der Mensch-Roboter-Kollaboration ein erhöhter Anspruch an die Kollisionserkennung. Nur so kann vermieden werden, dass es zu Zusammenstößen mit Hindernissen, Personen und auch mit sich selbst kommt.

Ziel der Arbeit

In dieser Arbeit wird sich mit der Thematik Kollisionsberechnung auseinandergesetzt. Eingebettet in das Forschungsprojekt *Smarte Objektübernahme und -übergabe für die nutzerzentrierte mobile Assistenzrobotik (SONARO)*, leistet diese Masterarbeit ihren Beitrag in der Kollisionsberechnung der Bewegungssteuerung robotischer Manipulatoren. Im Vordergrund steht dabei die Frage, wie die Berechnung der Bewegungssteuerung robotischer Manipulatoren erfolgen kann. Ziel der hier vorliegenden Arbeit ist dementsprechend die Entwicklung einer Kollisionserkennung und deren Implementierung in das im Projekt verwendete Framework. Hierfür wird sich mit unterschiedlichen Möglichkeiten der Implementierung bereits vorhandener Bibliotheken auseinandergesetzt und deren Einbindung in das Projekt geprüft. In Anlehnung daran wird dann ein eigener

Ansatz zur Kolllisionsberechnung entwickelt. Im Bereich der Kollisionserkennung gibt es üblicherweise zwei unterschiedliche Standardverfahren: Zum einen die Kollisionstest, die besagen ob eine Kollision stattfindet oder nicht und zum anderen Abstandsberechnungen. Anders als bei den Kollisionstest berechnen die Abstandsberechnungen den kleinsten Abstand zwischen dem Roboter und einem möglichen Hindernis [Ebert, 2003]. Diese Berechnungen sind aufwändiger als die Kollisionstest, lohnen sich allerdings, wenn bei der Kollisionsvermeidung die Zusatzinformation über den Abstand weiter genutzt werden kann. In der vorliegenden Arbeit soll zusätzlich zur eigentlichen Kollisionserkennung ebenfalls der jeweilige Abstand zu den Objekten berechnet werden.

Aufbau der Arbeit

Die Arbeit ist folgendermaßen gegliedert: In Kapitel 2 werden die wesentliche mathematische Grundlagen, die zur Berechnung der Kollisionserkennung benötigt werden, aufgeführt. Der Stand der Technik und der aktuelle Forschungsstand wird im Kapitel 3 zusammen mit den verwendeten Simulationsumgebungen vorgestellt und erläutert. Es folgt in Kapitel 4 die Umsetzung des eigenen Ansatzes. Es wird beschrieben, welche Konzepte eingesetzt und wie diese konkret umgesetzt wurden. Zudem wird auch der entwickelte Ansatz bewertet. Es folgt in Kapitel 5 ein zusammenfassendes Fazit und ein Ausblick auf weitere mögliche zu erarbeitende Aspekte.

Kapitel 2

Theoretische Grundlagen

Die vorliegende Arbeit setzt sich mit der Kollisionsberechnung auseinander. Eine Kollision ist „a configuration of two objetcts occupying the same point in space“[van den Bergen, 2004, S. 1]. Befinden sich demnach zwei Objekte im gleichen Bereich bzw. am gleichen Punkt, findet eine Kollision dieser Objekte statt. Um diese mögliche Kollision zu berechnen, muss unter anderem dieser Punkt, an dem beide miteinander kollidieren könnten, beachtet werden.

Im Verlauf des Kapitels wird zunächst auf die wesentlichen mathematischen Grundlagen eingegangen, insbesondere Abstandsberechnugen, die bei den Kollisionstests verwendet werden. Im Anschluss daran wird ein allgemeiner Überblick über die Vorgehensweisen von Methoden zur Kollisionsberechung gegeben, sowie auf verschiedene Probleme, die berücksichtigt werden müssen, eingegangen. Danach folgt die Beschreibung des aktuellen Forschungsstands des Fachgebiets Neuroinformatik und Kognitive Robotik der TU Ilmenau. Zuletzt werden die verwendete Software sowie die untersuchten Physik-Engines näher beschrieben.

2.1 Mathematische Grundlagen

Im Folgenden werden die mathematischen Grundlagen beschrieben, auf denen die Kollisions- und Abstandstests der hier vorliegenden Arbeit aufgebaut sind.

2.1.1 Euklidischer Raum

Für die Berechnungen des euklidischen Abstands im dreidimensionalen Raum sind Vektoren essentiell. Anhand von Vektoren kann die Beziehung zwischen zwei Punkten p und q beschrieben werden. Die Länge eines Vektor $\underline{\mathbf{v}}$ ist durch seinen Betrag $\|\underline{\mathbf{v}}\|$ definiert ([Verth und Bishop, 2008, S. 44ff.]).

$$\underline{\mathbf{v}} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \qquad \|\underline{\mathbf{v}}\| = d = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{2.1}$$

Die Distanz $d(p,q)$ zwischen den zwei Punkten p und q entspricht somit der Länge des Vektors [van den Bergen, 2004].

$$d(p,q) = \|p - q\| \tag{2.2}$$

Mithilfe des Skalarproduktes zweier Vektoren $\underline{\mathbf{v}}$ und $\underline{\mathbf{w}}$ mit dem Winkel θ kann die Lage der Vektoren zueinander bestimmt werden. Das zeigt die geometrische Deutung des Skalarproduktes in der folgenden Gleichung. Zur vollständige Herleitung siehe [Verth und Bishop, 2008, S. 47ff.].

$$\underline{\mathbf{v}} \cdot \underline{\mathbf{w}} = v_x w_x + v_y w_y + v_z w_z = \|v\| \|w\| \cos\theta \tag{2.3}$$

Für das Ergebnis ergeben sich die folgenden Beziehungen für den Winkel θ zwischen den Vektoren:

$$\underline{\mathbf{v}} \cdot \underline{\mathbf{w}} = \begin{cases} < 0 & \text{stumpfer Winkel} \\ = 0 & \text{rechter Winkel (orthogonal)} \\ > 0 & \text{spitzer Winkel} \end{cases} \tag{2.4}$$

Des Weiteren entspricht das Skalarprodukt der quadratische Länge des Vektors, denn es gilt:

$$\underline{\mathbf{v}} \cdot \underline{\mathbf{v}} = \|\underline{\mathbf{v}}\|^2 \tag{2.5}$$

Eine weiteres wichtiges Element der Vekotorrechnung ist das *Kreuzprodukt*. Hierbei

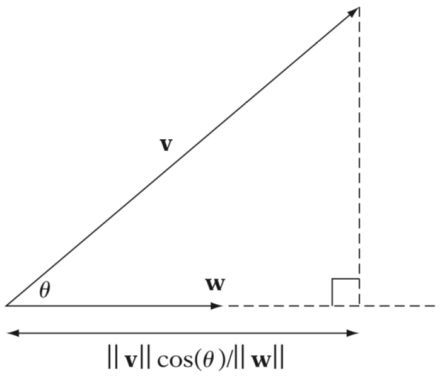


Abbildung 2.1: Grafische Darstellung der Projektion des Vektors v auf den Vektor w ([Verth und Bishop, 2008, S. 53ff]).

wird aus den exisitierenden Vektoren $\underline{\mathbf{v}}$ und $\underline{\mathbf{w}}$ ein neuer Vektor $\underline{\mathbf{v}} \times \underline{\mathbf{w}}$ gebildet, der orthogonal zu beiden Vektoren ist. Für das Kreuzprodukt gilt die Formel 2.6, eine genau Herleitung dieser findet sich in [Verth und Bishop, 2008, S.53ff].

$$\underline{\mathbf{v}} \times \underline{\mathbf{w}} = (v_y w_z - w_y v_z, v_z w_x - w_z v_x, v_x w_y - w_x v_y) \tag{2.6}$$

Für die Abstandsberechnungen ist es wichtig, einen Vektor auf einen anderen projizieren zu können. Für die Projektion des Vektors $\underline{\mathbf{v}}$ auf den Vektor $\underline{\mathbf{w}}$ ergibt sich nach Abbildung 2.1 die Formel 2.7, vergleiche hierzu [Verth und Bishop, 2008, S.50ff]. Der projizierte Punkt liegt parallel und somit orthogonal zum Ursprungsvektor.

$$proj_{\underline{\mathbf{w}}}\, \underline{\mathbf{v}} = \frac{\underline{\mathbf{v}} \cdot \underline{\mathbf{w}}}{\|\underline{\mathbf{w}}\|^2}\, \underline{\mathbf{w}} \tag{2.7}$$

2.1.2 Geometrische Abstandsberechnungen

Bei den Kollisionsberechnungen wird sich auf geometrische Abstandsberechnungen bezogen. Im Folgenden werden die wesentlichsten Abstandsberechnungen aufgeführt. Diese bilden die Grundlage für die Kollisionstest aus Kapitel 4.3.

Punkt zu Punkt

Die Abstandsberechnung zwischen zwei Punkten entspricht der Länge des Vektors zwischen diesen Punkten. Somit kann die Formel 2.2 oder für die quadratische Länge die Formel 2.5 verwendet werden.

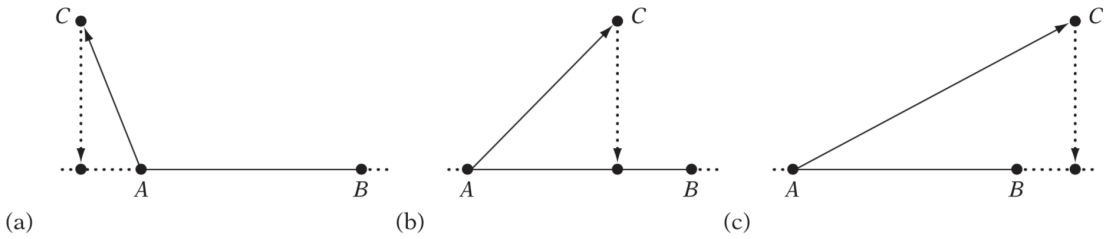


Abbildung 2.2: Quelle: [Ericson, 2005, S. 128] Fallunterscheidungen des Punktes zum Streckensegment. (a) Außerhalb des Segments bei Punkt A, (b) Innerhalb der Segments, (c) Außerhalb des Segments bei Punkt B

Punkt zu Strecke

Für die Berechnung des Abstandes eines Punktes zu einer Strecke muss zuerst der nächste Punkt auf der Strecke zum zu prüfenden Punkt gefunden werden. Zur Berechnung des nächsten Punktes werden drei mögliche Fälle betrachtet (siehe Abbildung 2.2). Anhand der Winkelinterpretation des Skalarproduktes (Formel 2.4), kann die Lage des Punktes für den Fall (a) bestimmt werden. Daraus ergibt sich der Abstand aus der Länge des Vektors zwischen Startpunkt der Linie und dem Punkt. Für Fall (b) und (c) wird der Punkt auf die Strecke projiziert. Ist der Streckenteil des projizierten Punktes länger als die Strecke an sich, so liegt der Punkt hinter dem Endpunkt der Strecke und die Distanz ergibt sich aus der Länge des Vektors zwischen den Endpunkt der Strecke und dem Punkt. Anderenfalls liegt der Punkt auf der Strecke. Ist dies der Fall ergibt sich der Abstand aus dem Vektor zwischen projizierten Punkt und dem Orignialpunkt.

Das eben beschriebene Vorgehen für die Abstandsberechnung eines Punktes zu einer Strecke wird in dem nachfolgen Algorithmus noch einmal verdeutlicht.

Punkt zu Ebene

Eine Ebene wird durch zwei nicht parallele Vektoren $\underline{\mathbf{v}}$ und $\underline{\mathbf{w}}$ aufgespannt. Das Ergebnis des Kreuzproduktes aus diesen Vektoren, ergibt den Normalenvektor $\underline{\mathbf{n}}$. Für den Abstand eines Punktes P, muss dieser auf den Normalenvektor projiziert werden. Die Länge des Vektors $\underline{\mathbf{n}}$ zum projizierten Punkt entspricht den Abstand zur Ebene.

Algorithmus 2.1: Berechnung der minimalen Distanz zwischen einem Punkt und einer Strecke.

Input : Punkt P
Startpunkt der Strecke S
Endpunkt der Strecke E
Output : Minimale Distanz zwischen Punkt und Strecke d_{min}

if $\underline{\mathbf{v}}_{P-S} \cdot \underline{\mathbf{v}}_{E-S} \leq 0$ **then**
  $d_{min} = \|\underline{\mathbf{v}}_{P-S}\|$
else
  $\underline{\mathbf{v}}_{Proj_P} = proj_{\underline{\mathbf{v}}_{E-S}} \underline{\mathbf{v}}_{P-S}$ // Projektion von $\underline{\mathbf{v}}_{P-S}$ in $\underline{\mathbf{v}}_{E-S}$
  if $\|\underline{\mathbf{v}}_{Proj_P}\| \geq \|\underline{\mathbf{v}}_{E-S}\|$ **then**
    $d_{min} = \|\underline{\mathbf{v}}_{P-E}\|$
  else
    $d_{min} = \|\underline{\mathbf{v}}_{P-Proj_P}\|$
return d_{min}

Strecke zu Strecke

Bei der Kollision zweier Strecken müssen unterschiedliche Aspekte überprüft werden: Zum einen muss getestet werden, ob die beiden Strecken parallel zueinander sind. Dies kann anhand des Skalarproduktes der Richtungsvektoren erfolgen. Sind die beiden Strecken parallel zueinander, so gibt es keine Kollision und der Abstand kann anhand eines orthogonalen Vektors zwischen den Strecken berechnet werden. Liegen ein Start- oder Endpunkt zwischen dem Start- und Endpunkt der anderen Strecke, dann ist der Abstand zwischen beiden Strecken gleich der Länge des orthogonalen Vektors zwischen ihnen. Anderenfalls muss der nächste Start- bzw. Endpunkt bestimmt werden und darauf die Abstandsberechnung zwischen Punkt und Strecke angewandt werden (vgl. Abbildung 2.3).

Zum anderen muss für denn Fall, dass die Strecken nicht parallel zueinander liegen, getestet werden, wo sich der Schnittpunkt beider aus den Strecken resultierenden Geraden befindet. Ist der Schnittpunkt der Strecke gleich der Schnittpunk der Geraden, so liegt eine Kollision vor. Der Abstand ist gleich 0. Andernfalls müssen die Punkte die den Strecken am nächsten liegen bestimmt werden. Liegen beide Punkte auf den Streckensegmenten entspricht deren Abstand, den Abstand bei Strecken. Befindet sich

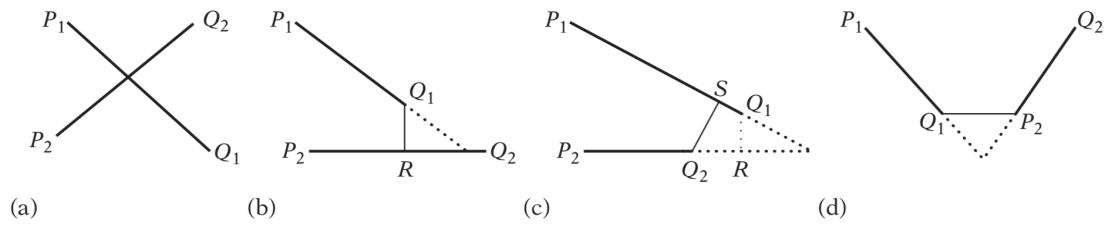


Abbildung 2.3: Quelle: [Ericson, 2005, S. 148] Fallunterscheidungen der Berechnung der nähsten Punkte zwischen zwei Strecken. (a) Schnittpunkt; (b) und (c) nur ein Punkt liegt auf einer Strecke der beiden Strecken; (d) beide Endpunkte

nur ein Punkt auf der Linie müssen dieser auf die andere Gerade projiziert werden um den Abstand zu berechnen [Ericson, 2005].

Abstand Strecke zu Ebene

Ist der Richtungsvektor der Strecke und der Normalenvektor der Strecke orthogonal, so sind bei parallel zueinander. Der Abstand zwischen Strecke und Ebene kann mit der Projektion eines beliebigen Punktes der Strecke auf den Normalenvektor der Ebene bestimmt werden. Sind Strecke und Ebene nicht parallel, so muss der Schnittpunkt der Geraden mit der Ebene bestimmt werden. Die Gerade entspricht der Strecke, nur mit einer nicht begrenzten Länge. Anhand des Schnittpunktes mit der Ebene kann nun der Punkt auf der Strecke festgestellt werden, der am nächsten zu der Ebene liegt. Der Abstand kann somit auf das Punkt-Ebene-Problem zurückgeführt werden.

Abstand Ebene zu Ebene

Sind zwei Ebenen parallel existiert ein Abstand zwischen ihnen. Andernfalls schneiden sich die beiden Ebenen. Mit dem Skalarprodukt der beiden Normalenvektoren kann nachgewiesen werden ob die Ebenen parallel sind. Liegen zwei parallele Ebenen vor, kann ein beliebiger Punkt von Ebene 1 auf den Normalenvektor der Ebene 2 projiziert werden. Entsprechend dem Verfahren zur Punkt-zu-Ebene-Berechnung.

2.2 Grundverfahren der Kollisionsberechnung

Die Berechnung auf Kollisionsfreiheit von Objekten ist besonders im dreidimensionalen Raum sehr komplex. Betrachtet man nur statische Objekte (nicht bewegende Objekte) ist eine einfache Berechnung möglich. Im Gegensatz dazu gestaltet sich die Berechnung bei dynamischen Objekten (bewegende Objekte) komplexer, da hier zusätzlich die Bewegung einkalkuliert und betrachtet werden muss.
Das nachfolgende Kapitel beschreibt die Anforderungen und Probleme, die im Allgemeinen von einer Kollisionserkennung berücksichtigt und gelöst werden müssen. Außerdem werden verschiedene Verfahren betrachtet, durch die der Komplexitätsgrad verringert werden kann.

2.2.1 Problemstellungen

Bei einer Kollisionserkennung können zwei wesentliche Grundprobleme identifiziert werden, die sich auf Komplexität und Berechnungsdauer auswirken. Dazu zählt zum einen die Anzahl der Objekte, die gegeneinander auf Kollisionsfreiheit geprüft werden müssen. Beispielsweise ist der Rechenaufwand bei einer Kollisionserkennung von drei Objekten deutlich geringer als bei zehn Objekten, die auf Kollision geprüft werden müssen. Das zweite Grundproblem ist die Komplexität der Objekte an sich. Die Berechnung zweier primitiven Geometrien, wie z.B. Kugeln, ist im Vergleich zu zwei Objekten mit komplexeren Geometrien, wie beispielsweise Polygone, deutlich geringer und weniger schwierig. Dies wirkt sich dementsprechend auch auf die Berechnungsdauer aus.
Bei der Betrachtung von dynamischen Objekten kommen zudem weitere Problemstellungen hinzu: So muss einerseits auf die Bewegungsreihenfolge von Objekten geachtet werden, ob sich die Objekte nacheinander (sequential) bewegen oder ob die Bewegung zur gleichen Zeit (simultaneous) erfolgt. Die simultane Bewegung von Objekt wird auch als natürliche Bewegung bezeichnet. Sie wird in dieser Arbeit als Bewegungsart angenommen. Die Abbildung 2.4 zeigt die Auswirkung beider Bewegungsarten auf das Kollisionsverhalten. Die oben dargestellten Fälle beschreiben die sequentielle und die unteren entsprechend die simultane Bewegungsreihenfolge.

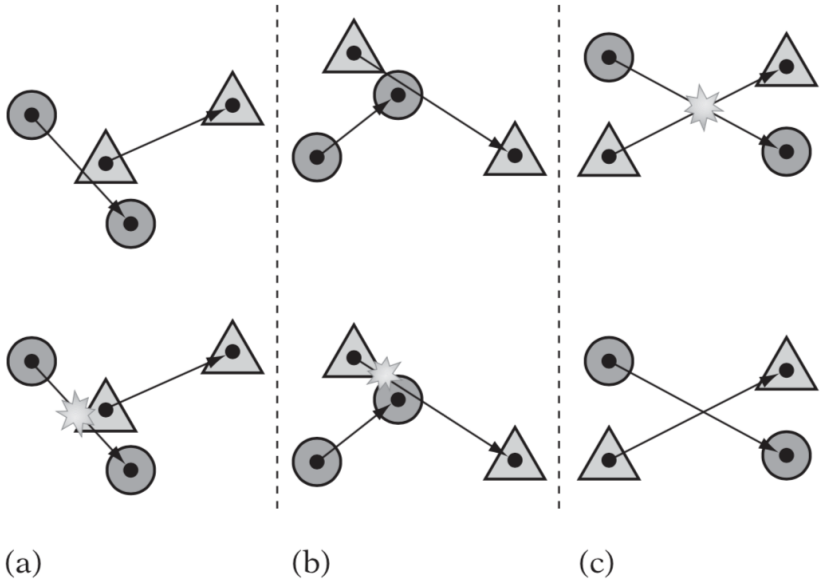


Abbildung 2.4: Quelle: [Ericson, 2005, S. 55 Figure 2.5] Auswirkung von Bewegungsarten auf das Kollisionsverhalten
(a) oben: Kollisionsfrei bei parallel Bewegung. unten: Kollision, bei Bewegung nacheinander.
(b) oben: Kollisionsfrei bei parallel Bewegung. unten: Kollision, bei Bewegung nacheinander.
(c) oben: Kollision bei parallel Bewegung. unten: Kollisionsfrei, bei Bewegung nacheinander.

Eine weitere Problemstellung bei dynamischen Objekten stellt andererseits die Bewegungsgeschwindigkeit und die Abtastungsrate der Kollisionserkennung dar. Von der Szene wird zu diskreten Zeitpunkten eine Momentaufnahme gemacht, die auf Kollisionsfreiheit getesten werden soll. Die Bewegungsänderung der Objekte in einer Szene werden nur zu diesen Zeitpunkten (z.B. alle 100ms) erkannt und geprüft. Bewegen sich die Objekte relativ langsam, kann eine Berechnung auf Kollision sehr zuverlässig durchgeführt werden. Gibt es hingegen Objekte, die sich sehr schnell bewegen und dazu noch relativ klein sind, können Kollisionen zwischen den Zeitpunkten übersehen werden. In der Literatur ([Ericson, 2005]) wird oft auf das Beispiel von einer Pistolenkugel und einem Blatt Papier Bezug genommen. Das Phänomen, bei dem Kollisionen es zwischen zwei Abtastungen nicht entdeckt werden, nennt man **Tunneling** Abb. 2.5 .

(a) Zeitschritt $t = 0$ **(b)** Zeitschritt $t = 1$

Abbildung 2.5: Darstellung des Tunnelingproblems
Unter (a) befindet sich die Kugel vor dem Hindernis und in (b) befindet sich die Kugel nach dem Hindernis, ohne dass eine Kollision erkannt wurde, da die Abtastung zu gering ist.

Am Anfang einer Kollisionsberechung muss zunächst geklärt werden, was genau getestet werden soll. Dazu sind die folgenden drei Punkte zu beachten.

1. Nur Prüfung auf Berührung oder Überschneidung von zwei Objekten (Intersection Test)
2. Prüfung an welcher Stelle sich die beiden Objekte berühren oder schneiden. (Intersection Finding)
3. Prüfung zu welchen Zeitpunkt sich die Objekte berühren oder schneiden (Intervall Tests)

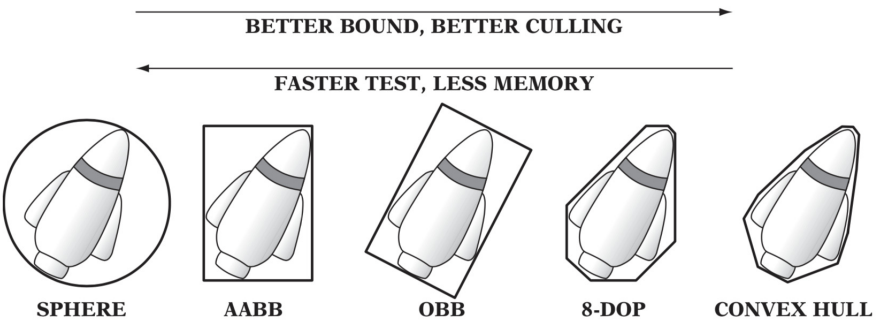


Abbildung 2.6: Quelle: [Ericson, 2005, S. 77, Figure 4.2] Übersicht verschiedener Bounding Volumes mit Bezug auf die Geschwindigkeit der Testbarkeit und die Genauigkeit beim Objektfitting.

Zur Umsetzung einer Kollisionsberechung gibt es verschiedene Verfahren, die die eben genannten Probleme angehen.

2.2.2 Polygonbasierte Verfahren

Anhand der polygonbasierten Verfahren wird ein geometrisch komplexes Objekte von einem geometrisch, einfachen Hüllkörper (Bounding Volume) umschlossen. Mit Hilfe der einfacheren Geometrien lassen sich zwei Objekte schneller auf Kollisionsfreiheit überprüfen. Denn je einfacher die Geometrie desto einfacher und schneller ist die Kollisionsberechnung. Berücksichtigt werden muss dennoch, dass bei einfachen Geometrien oft die Objekteinhüllung sehr ungenau ist. So ist beispielsweise ein Stift bzw. ein länglicher Gegenstand mit einer Kugel als Hüllkörper sehr ungünstig beschrieben, da die Kugel sehr viel Hohlraum beinhaltet, der beim Schnitt mit einer anderen Geometrie trotzdem zu einer Kollision führen kann. Deshalb ist bei der Wahl des Hüllkörpers wesentlich, den richtigen Kompromiss zwischen Genauigkeit der Abdeckung des Hüllkörper und der Geschwindigkeit der Kollisionstests zu finden. Die Abbildung 2.6 zeigt verschiedene Hüllkörper-Geometrien und deren Bezug auf Genauigkeit der Einhüllung und ihre Testgeschwindigkeiten.
Die aufgeführten Hüllkörper-Geometrien bzw. Bounding Volumes sind:

- Kugel-Form (Sphere)
- Box (AABB und OBB)
- Zylinder (Cylinder)

- 8-DOP
- Konvexe Hülle

Das Bounding Volume **Sphere**, die Kugel-Form, ist die einfachste Geometrie der Bounding Volumes. Im Vergleich zu den anderen Bounding Volumes muss der Hüllkörper nicht bei einer Rotation des umhüllten Objektes neu berechnet werden. Jedoch ist das Volumen nicht für alle Objekte geeignet. Objekte die viel länger als breit sind erzeugen viel Blindraum (unbenutzter Raum) und führt zu vielen Kollisionserkennungen im Freiraum. Daher ist das Volumen nicht unbedingt für präzise Bewegungen geeignet.

Die Box als Hüllkörper kann auf zwei verschiedene Arten verwendet werden. Zum einen kann ein Objekt von einer Box umschlossen werden, die nach den Achsen des Weltkoordinatensystems ausrichtetet und konstruiert ist. Dabei handelt es sich um eine sogenannte **Axis Aligned Bounding Box (AABB)**. Diese Box muss bei jeder Rotation des Objektes neu berechnet werden, wodurch sich das Objektfitting verschlechtert. Zum anderen gibt es noch die am **Oriented Bounding Box (OBB)**. Diese Box ist nicht raumfest, sondern an den Körper, die die Box umschliesst, angepasst. Der entscheidende Unterschied zur AABB ist, dass die Box bei einer Rotation nicht mehr neu erzeugt werden muss, sondern zusammen mit dem Objekt rotiert. Somit kann das Objektfitting beibehalten werden.

Weitere Hüllkörper-Volumen sind sogenannte **Hüllkörper-Polyeder**. Hierbei wird das Objekt von Ebenenpaaren umschlossen. Der Abstand des Normalenvektors der Ebenenpaare zum Bezugskoordinatensystem wird dabei gespeichert. Die Hüllkörper-Polyeder werden auch als **k-Discrete Oriented Polygon (k-DOP)** bezeichnet. Der Buchstabe k steht dabei für die Anzahl der Seitenebenen.

Die komplexeste Hüllkörper-Geometrie ist die **konvexe Hülle**. Mittels einer polygonalen Beschreibung wird das Objekt umfasst. Eine konvexe Hülle ist so definiert, dass eine Gerade zwischen zwei beliebigen Punkten innerhalb des Hüllkörpers, den Hüllkörper nicht verlässt. Der wesentliche Vorteil ist das gute Objektfitting. Nachteilig ist hierbei aber der hohe Speicherverbrauch für die Hüllkörperbeschreibung und die komplexe Geometrie.

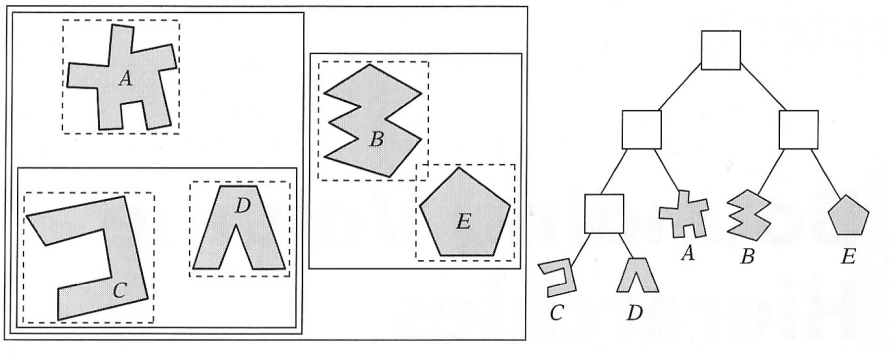


Abbildung 2.7: Quelle: [Ericson, 2005, S. 236, Figure 6.1] Darstellung einer Hüllkörperhierarchie mit fünf einfachen Geometrien. Verwendet wurden AABB-Boxen als Hüllkörper

Hüllkörperhierachien (Bounding Volume Hierarchy)

Bei einer Hüllkörperhierachie (Bounding Volume Hierarchy) handelt es sich um eine baumförmige Struktur mit deren Hilfe Kollision erkannt werden soll ([Ericson, 2005],[Gottschalk, 2000]). Hierbei wird mit groben Hüllkörpern gearbeitet. Wird dabei eine Kollision erkannt, so verwendet man genauere Hüllkörper, um eine Kollision zu bestimmen. Wird beim ersten Test keine Kollision erkannt, kann auch keine Kollision mit darunterliegenden Objekten passieren und eine weitere Testung ist daher nicht notwendig. Dieses Vorgehen erleichtert und beschleunigt den Berechnungsprozess der Kollision. Jedem Knoten wird ein Hüllkörper zugeordnet. Dieser umfasst alle nachfolgenden Objekte und ist der Grund für die baumartige Struktur, die dadurch entsteht 2.7. Werden diese Objekte in dieser Struktur organisiert, ist es nur dann notwendig Kindknoten darauf zu testen, ob sie sich gegenseitig schneiden, wenn sich ihre Elternknoten schneiden ([Gottschalk, 2000],[Petersen, 2007]).

Konstruktion von Hüllkörperhierachien

Zur Konstruktion von Hüllkörperhierachien gibt es drei verschiene Ansätze, die genutzt werden können. Die betrachteten Hüllkörper-Ansätze sind:

- Der Top-Down-Ansatz
- Der Bottom-up-Ansatz
- Die Insertion-Methode

Top-Down-Ansatz

Beim Top-Down-Ansatz wird rekursiv gearbeitet. Zuerst wird ein Hüllkörper gebildet, der die gesamte Geometrie umfasst. Im Anschluss daran wird diese Menge in kleiner Teilmengen geteilt, die wiederum jeweils mit einem Hüllkörper umgeben werden Abb. 2.8a. Dies wird solange entsprechend ausgeführt, bis entweder in jeder Teilmenge nur noch ein Objekt enthalten ist oder bis ein definiertes Abbruchkriterium erreicht wurde ([Ericson, 2005],[Petersen, 2007]). Beim Top-Down-Verfahren gibt es folgende Einflussgrößen, die berücksichtigt werden müssen, um einen Baum zu erstellen ([Aichele, 2015, S. 39]):

- „Die Partionierungsstrategie (Median, Minimierung des Volumens der Kind-Hüllkörper, Minimierung der Überlappung der Kind-Hüllkörper)
- Die Abbruch-Kriterien (Anzahl umschlossener Primitive, minimales Volumen der Kind-Hüllkörper, Tiefe der Hierarchie)
- Die Auswahl der partitionierenden Hyper-Ebene und der Zuordnung verbleibender Primitive“

(a) Top-Down-Ansatz **(b)** Bottom-Up-Ansatz

Abbildung 2.8: Zwei Verfahren zur Konstruktion von Hüllkörperhierarchien
Im Vergleich der beiden Methoden ist die Bottom-up-Methode die einfachere von beiden. Allerdings kann es mit diesem Verfahren zu unnötig großen und somit ineffizienten Hüllkörpern kommen. Das Top-Down-Verfahren dagegen erzeugt genauere und angepasstere Hüllkörper, da die Objekte immer mehr in kleinere Bereiche geteilt werden.([Aichele, 2015]).

Bottom-up-Ansatz

Anders als beim Top-Down-Ansatz wird beim Bottom-up-Verfahren mit der gesamtem Menge an Objekten die Erstellung des Baumens begonnen. Hierbei handelt es sich um eine iterative Methode [Aichele, 2015]. Die einzelnen Objekte werden als ein

Blatt des Baumes definiert und jedes Objekt wird mit einem Hüllkörper umgeben. Im nächsten Schritt werden zwei oder mehr Objekte zu einem Knoten zusammengefasst und ein neuer gemeinsamer Hüllkörper wird berechnet. Dieses Vorgehen wird solange durchgeführt, bis nur noch ein Knoten vorhanden ist, der sämtliche Objekte umfasst und als Wurzel des Baumes gilt Abb. 2.8b. Wichtigstes Auswahlkriterium dieses Ansatzes ist„die Wahl des Paares aus Knoten, die in einer Iteration durch einen neuen Eltern-Knoten kombiniert werden sollen“([Aichele, 2015, 39]).

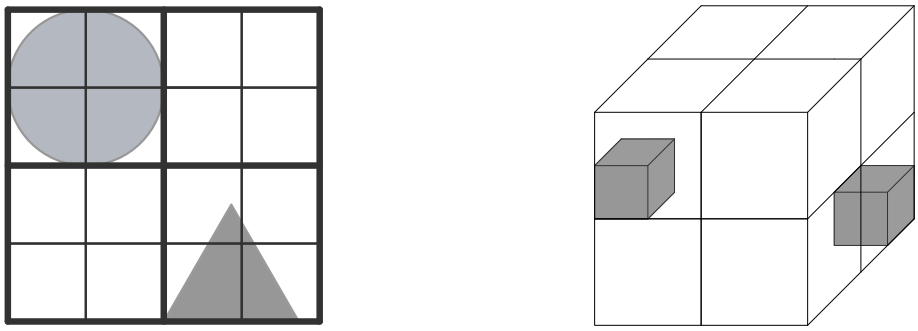
Die Insertion Methode

Bei der Insertion-Methode wird „one object at a time, starting from an empty tree“([Ericson, 2005, S. 251]) hinzugefügt. Es handelt es sich demnach um ein Vorgehen, bei dem stufenweise Objekte hinzugefügt werden. Begonnen wird mit einem leerem Baum bzw. leeren Hierarchie, in die sukzessiv neue Objekte hinzugefügt werden. Die Objekte werden jeweils dort in die bestehende Hierarchie eingefügt, sodass „das Volumen der Hierarchie durch das Einfügen eines neuen Elements möglichst wenig wächst.“[Aichele, 2015, 39]. Die Knoten werden demnach danach bemessen, wie sehr sich das Volumen vergrößern würde. Objekte, die von einem größeren Hüllkörper umgeben sind, werden daher eher weiter oben im Baum eingefügt, während Objekte, die kleiner sind in andere Hüllkörper integriert werden können. Dieses Vorgehen eignet sich insbesondere zur Aktualisierung dynamischer Bäume, da zu Beginn der Konstruktion der Hierarchie nicht alle Objekte bereits vorhanden sein müssen ([Ericson, 2005],[Petersen, 2007]).

2.2.3 Raumparitionierungsverfahren

Eine andere Möglichkeit Objekte auf Kollision miteinander zu prüfen, bieten die Raumpartitionierungsverfahren, bei denen rekursiv vorgegangen wird ([Aichele, 2015]). Dabei ist das Ziel den Raum in verschiedene bzw. disjunkte, gleichgroße Teilräume zu unterteilen. In die daraus entstehenden Teilräume werden die vorhandenen Objekte, je nach ihrer Lage im Raum, eingeordnet. Mit Hilfe dieser Teilräume kann herausgefunden werden, ob sich Objekte schneiden können oder nicht. Befinden sich zwei Objekte in unterschiedlichen Teilräumen besteht Kollisionsfreiheit, da diese räumlich voneinander getrennt bzw. disjunkt sind. In diesem Fall muss kein weiter Test auf Kollision erfolgen.

Es müssen somit nur die Objekte genauer auf Kollision miteinander getestet werden, die sich im gleichen Teilraum befinden([Aichele, 2015]). Eine weitere Unterteilung der Teilräume kann weitere Kollisionstests ersparen. Weitere Unterteilungen können beliebig oft erfolgen. Es entsteht dadurch eine Baumstruktur, die in Quadtrees (2D) oder in Octrees (3D) gespeichert werden (Abbildung 2.9).



(a) Quadtree **(b)** Octree

Abbildung 2.9: Übersicht Raumpartionierung
a) Beispieldarstellung des Gittermusters eines Quadtrees mit Objekten b) Beispieldarstellung eines Octrees mit Objekten

Eines der häufig genutzten Verfahren zur räumlichen Partitionierung von Objekten wird das Separating-Axis-Theorem (SAT) verwendet. Hier wird durch die Erzeugung einer Ebene zwischen zwei Objektpaaren die räumliche Trennung gezeigt. Für die Aufteilung der Objekte in die Sektoren oder generell für Berührungs- oder Schnitttests kann das Seperating Axis Theorem verwendet werden. Das Theorem besagt, dass sich zwei Objekte genau dann nicht schneiden, wenn eine Hyperebene zwischen diese beiden konvexen Geometrien vorhanden ist. Denn dann liegen beide Objekte in unterschiedlichen, voneinander getrennten Halbräumen und überschneiden sich nicht.

Polygonbasierteverfahren im Vergleich zu Raumpartitionierungsverfahren
Ein Unterschied zwischen diesen beiden Verfahren ist, dass sich bei Hüllkörperhierachien zwei Hüllkörper in unterschiedlichen Verzweigungen schneiden. Beim Raumpartitionierungsverfahren dagegen entstehen eindeutig abgegrenzte, disjunkte Bereiche. Vorteil der Hüllkörperhierarchien ist, im Gegensatz zum Raumpartitionierungsverfahren ist, dass die Hüllkörper an die Objekte angepasst sind und daher diese effizienter umhüllen. Des Weiteren sind mit diesem Verfahren einfache Kollisionstests möglich. Vorteil

der Octreestruktur dagegen ist insbesondere ihre geringe benötigte Speicherkapazität ([Aichele, 2015],[Petersen, 2007]).

2.2.4 Bildraumbasierte Verfahren

Eine weitere Möglichkeit um Kollisionen von Objekten zu berechnen, bieten Bildraumbasierte Verfahren. Dieser Verfahren ermitteln mögliche Überschneidungen auf Basis von Ansichten, die aus verschiedenen Perspektiven erzeugt werden ([Aichele, 2015]). Das bedeutet, dass die Überprüfung auf eine mögliche Kollision nicht dadurch erfolgt, dass die einzelnen Objekte mittels Hüllkörper umgeben und diese Hüllkörper auf mögliche Überschneidung getestet werden, sondern es wird getestet, ob sich zwei Objekte aus unterschiedlichen Perspektiven heraus überlappen und ob sich daraus Schnittvolumen ergeben ([Aichele, 2015]). Besonders eignet sich dieses Verfahren für rasterbare Geometrien und unterstützen daher u.a. verformbare Geometrien. Vorteil dieses Verfahrens ist, dass zumindest teilweise auf Hüllkörperhierarchien und somit auf umfangreiche Berechnungen verzichtet werden kann. Allerdings ist die Art Kollisionserkennung möglicherweise nicht so präzise ist, da ihre „geometrische Genauigkeit [...] durch die Auflösung der Bildpuffer, die zur Rasterisierung der verwendeten Geometrien verwendet werden beschränkt“([Aichele, 2015, S. 177]) ist. Aus diesem Grund können möglicherweise Kollisionen zwischen Objekten nicht erkannt werden, insbesondere, wenn beispielsweise sehr kleine Objekte mit wenigen Pixeln im Bildpuffer abgebildet werden, da sie von der Kamera sehr weit entfernt liegen.
Da dieses Verfahren in der vorliegenden Arbeit keine wesentliche Rolle spielt, wird im weiteren Verlauf nicht genauer darauf eingegangen. Für detailliertere Informationen sei auf die Dissertation von Fabian Aichele ([Aichele, 2015]) verwiesen.

2.2.5 Broad- und Narrow-Phase

Mit den Hüllkörpern können die Problemstellungen aus Kapitel 2.2.1 angegangen werden. Für das Problem mit der Anzahl der Objekte, die auf eine mögliche Kollision geprüft werden müssen, wird oft ein zweiphasiger Ansatz als Optimierung gewählt. In der ersten Phase werden die zu prüfenden Objekte in Gruppen unterteilt, z.B. auf Basis der Distanz zueinander. Somit kommen nur noch die Objekte aus den einzelnen Gruppen für weitere Kollisionstests in Frage. Objekte aus verschiedenen Gruppen

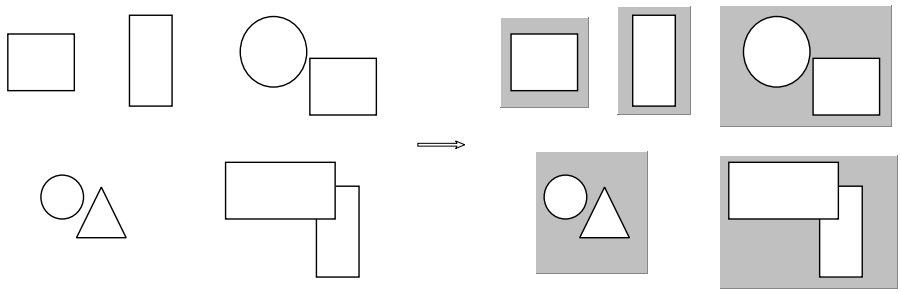


Abbildung 2.10: Darstellung freier Objekten (links) zu sortierten Objekten nach der Broadphase (rechts)

können sich in diesem Zeitschritt nicht berühren bzw. überschneiden. Somit können anhand einer Vorweg-Sortierung der Objekte unnötige Kollisionstests ausgeschlossen werden. Diese erste Phase wird deshalb als Vorfilter-Phase oder auf englisch als Broad-Phase bezeichnet.

Die einzelnen Objekte der Szene werden dabei üblicherweise von Hüllkörpern umschlossen und in einer räumlichen Partitionierung angeordnet. Anschließend wird geprüft welche Hüllkörper sich gemeinsam in einer oder mehreren Regionen der Partitionierung befinden. Alternativ können die Objekte auch anhand des Seperating Axis Theorem sortiert werden. Ein weiterer Anwendungsfall der Phase ist die Aufteilung von komplexer Objektgeometrie in eine Hüllkörperhierarchie. Die oberste Ebene ist eine einfache Geometrie, die das komplette Objekt umschließt. Danach wird das Objekt in jeder Hierarchie-Ebene immer feiner aufgelöst, so dass das komplexe Objekt in der niedrigsten Ebene von einer Vielzahl von Hüllkörper-Geometrien entsprechend genau dargestellt werden kann. Für die Kollisionstests werden die einzelnen Ebenen des Hierarchiehüllkörpers berücksichtigt. Das heißt, schneidet sich ein Objekt mit der obersten Ebene, dann muss ein weitere Tests mit der nächsten Ebene ausgeführt werden. Schneidet oder berührt kein Objekt die oberste Hierarchie-Ebene, dann besteht auch keine Kollision mit den darunter liegenden Ebenen und die Objekte kollidieren nicht miteinander. Ziel der Phase ist es unnötige Paartests für die zweite Phase (Narrow-Phase) schon vorzeitig auszuschließen, wie die Abbildung 2.10 veranschaulicht.

In der zweiten Phase der sogenannten Objektpaar-Phase oder auf englisch der Narrow-Phase werden die eigentlichen Kollisionstests durchgeführt. Mit diesem zweiphasigen Ansatz kann eine Menge an rechenaufwändigen Kollisionstests übersprungen werden.

Ziel der Phase ist es, Kollision zwischen Objektpaaren zu erkennen und kollisionsfreie Tests schon frühzeitig abzubrechen.
Die zwei Phasen werden bei den reinen Hüllkörper-Verfahren meist mit Hüllkörper-Hierarchien umgesetzt. Dabei werden einzelne Hüllkörper von einem größeren Hüllkörper eingeschlossen. Wird bei einem Kollisionstest der größere Hüllkörper nicht geschnitten so schneidet sich der Körper auch nicht mit den Hüllkörpern innerhalb. Die Tests bei diesen Phasen sind vorallem Überschneidungstests.

Kapitel 3

Stand der Technik

3.1 Simulationsumgebung und Frameworks

Im nachfolgenden Kapitel erfolgt die Beschreibung des aktuellen Forschungsstands des Fachgebiets Neuroinformatik und Kognitive Robotik der TU Ilmenau. Im Anschluss werden die verwendete Software sowie die untersuchten Physik-Engines näher beschrieben. Die eingesetzten Simulationsumgebungen und Frameworks die in dieser Arbeit verwendet werden, werden vorgestellt. Des Weiteren wird auf verwendete Karten und die Kollisionsberechnung im zweidimensionalen Raum eingegangen.

3.1.1 Robot Operating System

ROS steht für Robot Operation System und ist ein flexibles Framework zum Schreiben von Robotersoftware. „It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.“([ROS, o.D.]) Sinn und Zweck von ROS ist es, dass die Erstellung einer robusten, universellen Robotersoftware vereinfacht wird, da die Erstellung einer universellen Robotersoftware in unterschiedlichsten Umgebungen alleine kaum zu bewerkstelligen wäre. Die kollaborative Robotiksoftwareentwicklung wird durch ROS gefördert und ermöglicht, dass viele Personen zur Erstellung von Karten etwas beitragen können ([ROS, o.D.]).
Gazebo ist eine Open-Source Simulation, die im Rahmen des Player/Stage Projekts entwickelt wurde. Es handelt sich dabei um eine dynamische 3D-Simulation, die für

eine Vielzahl von an Robotern entwickelt wurde ([Gerkey et al., 2003]). Während Player als Schnittstelle zwischen den Sensoren und Stellgliedern des Roboters und der Software dient und Stage eine Vielzahl an mobilen Sensoren, Robotern und Objekten, in einer zweidimensionalen Umgebung simuliert ([Sta, o.D.]), bietet Gazebo dagegen bietet die Möglichkeit eine Vielzahl von Robotern in Innen- und Außenbereichen zu in dreidimensionalen Umgebungen zu simulieren. In der Arbeit wird Gazebo zur Simulation eingesetzt.

3.1.2 MIRA Framework

MIRA bietet, wie ROS, ein Framework zur Erstellung einer Robotersoftware und wird in Zusammenarbeit der Technischen Universität Ilmenau und der MetraLabs GmbH entwickelt. Es handelt sich hierbei um ein in C++ geschriebenes, plattformübergreifendes Framework. Geboten werden eine Basissoftware, mehrere verschiedene Grundfunktionen, sowie unterschiedliche Tools zur Entwicklung von Softwaremodulen ([MIR, o.D.b]). Ziel von MIRA ist, den sogenannten modularen Softwareentwicklungsprozess zu unterstützen. Hierbei handelt es sich um ein Softwaredesign, das zur Entwicklung von komplexen Anwendungen gedacht ist. Die Funktionalität eines Programms wird dabei in einzelne, voneinander unabhängige und austauschbare Module zerlegt. Jedes einzelne Modul erfüllt immer nur eine Funktion und umfasst daher alle Aspekte, die für die Ausführung dieser Funktion notwendig sind. Jedes Modul enthält einzelne Schnittstellen, die wiederum von den anderen Modulen zum Austausch von Daten genutzt werden können ([MIR, o.D.b]). MIRA bietet in diesem Zusammenhang eine Basissoftware, durch die es möglich wird, entwickelte Module so zusammenzustellen, dass die entsprechende komplexe Anwendung gebildet werden kann. „This middleware handles the communication between the modules efficiently and transparently“ ([MIR, o.D.b]), dient demnach zur Vermittlung der Kommunikation zwischen den einzelnen Modulen.

Im Vergleich zu Robot Operating System (ROS) bietet Middleware for Robotic Applications (MIRA) einige Vorteile. Darunter zählt unter anderem die Möglichkeit komplexere Datenstrukturen und Pointer an andere Komponenten zu übertragen. Außerdem besitzt MIRA im Vergleich zu ROS eine vollständige Dezentralisierung der Komponenten und somit existiert kein *single point of failure*. Unter ROS läuft die gesamte Kommunikation

über den ROS-Master-Knoten. Ein weiterer Vorteil ist die bessere Netzwerksicherheit, da verschiedene Authentifikationsstufen zwischen der Verbindung mit einem anderen Framework zur Verfügung stehen und die Kommunikation nur in eine Richtung (Master-Framework zu Client-Framework) möglich ist ([MIR, o.D.a]).

3.1.3 Physik Engines und Bibliotheken

Die Problemstellung, verschiedene Objekte auf eine Kollision miteinander zu testen, ist nicht neu. In der Computersimulation und vor allem bei Computerspielen müssen Objekte in sehr kurzer Zeit auf Kollision geprüft werden. Dazu gibt es verschiedene sogenannte Physik-Engines (engl. Dynamic Engines) in denen die Berechnung von Kollisionen nur einer kleiner Teil ist. Die Idee zu dieser Arbeit war es, Bibliotheken zur Kollisionsberechnung aus schon existierenden Physik-Engines zu verwenden und in MIRA zu verwenden. Nach längerer Recherche standen Bullet und Flexible Collision Library (FCL) in der engeren die Auswahl.

Bullet

Bullet ist eine OpenSource Physik-Engine, die auf verschiedenen Plattformen (Windows, Linux, Mac OSX, Android und andere) verwendet werden kann. Sie ist in portable C++ geschrieben und setzt sich mit Kollisionserkennung, Starrkörper-Dynamik, sowie der Dynamik von verformbaren Körpern auseinander. Hauptsächlich kommt die Engine in Computerspielen, -visualisierungen und Robotersimulationen zum Einsatz. Die Engine ist modular aufgebaut, sodass auch nur einzelne Teilbibliotheken, wie z.B. die Kollisionserkennung, verwendet werden können. Die Bullet-Kollisionserkennung bietet eine Vielzahl an Hüllkörpern, die zur Beschreibung der Objekte dienen. Darunter zählen beispielsweise Box, Kugel und Zylinder. Die verschiedenen Hüllkörper werden nach dem Testverfahren aus der Matrix in Tabelle 3.1 untereinander getestet.
Zur Berechnung auf Kollision müssen die Objekte in der Hüllkörperbeschreibung in eine konfigurierte *CollisionWorld* geladen werden. Zur Konfiguration stehen die Art der Broad- und Narrowphase zur Verfügung, so wie einige weitere Optionen. Die Berechnung der Objekte auf Kollision erfolgt mittels Callback-Funktionen ([Coumans, 2015]).
In der vorliegenden Arbeit konnte diese vielversprechende Bibliothek nicht verwendet werden, da sich diese nicht erfolgreich in das MIRA-Framework integrieren lies.

	box	**sphere**	**convex, cylinder, cone, capsule**	**compound**	**triangle mesh**
box	box box	sphere box	gjk	compound	concave convex
sphere	sphere box	sphere sphere	gjk	compound	concave convex
convex, cylinder, cone, capsule	gjk	gjk	gjk or SAT	compound	concave convex
compound	compound	compound	compound	compound	compound
triangle mesh	concave convex	concave convex	concave convex	compound	gimpact

Tabelle 3.1: Übersicht der Kollisionstests der Bullet Bibliothek ([Coumans, 2015])

Flexible Collision Library

FCL ist eine OpenSource-Kollisionserkennungsbibliothek, die besonders für Objekte, die aus Dreiecken bestehen, geeignet ist. Sie kann unter Windows und Linux verwendet werden. Neben den eigentlichen Kollisionstests können auch Abstandstests, für einen minimalen Abstand zwischen zwei Objekten, oder auch Toleranztests durchgeführt. Dabei wird überprüft, ob sich die Objekte innerhalb oder außerhalb eines Toleranzabstandes befinden. Zur Berechnung können die Objekte in eine Vielzahl an Hüllkörper überführt werden. Darunter zählen beispielsweise Box, Kugel, Ellipsoid und Zylinder. Um die verschiedenen Objekte untereinander zu testen, müssen diese angelegt und in einen *CollisionManager* registriert werden. Je nach Art des Managers wird ein unterschiedliches Broadphaseverfahren genutzt. Nach Abschluss der Objektregistration wird die Vorfilterung durchgeführt. Dann können die eigentlichen Kollisions- oder Abstandstest durchgeführt werden ([GitHub, 2019]).
Wie bereits bei der Bullet-Engine konnte diese Bibliothek nicht funktionsfähig in das MIRA-Framework integriert werden. Aus diesem Grund wurde in dieser Arbeit ein

eigener Ansatz entwickelt.

3.2 Forschungsstand

Das Fachgebiet Neuroinformatik und Kognitive Robotik der Technischen Unitverität Ilmenau entwickelt und verwendet für ihre Robotikprojekt verschiedene Module unter anderem zur Ansteuerung und Lokalisierung eines Roboters. Der folgende Abschnitt gibt einen Überblick der Module, die essentiell für die in dieser Arbeit entstanden Toolbox ist.
Die Lokalisierung eines Roboters erfolgt im Fachgebiet mit einer Normal Distribution Transform (NDT) Karte. Diese Karte dient als Hinderniskarte und somit als eine Informationsquelle der zu entwickelnden Kollisionserkennung. Der Bewegungsplaner benötigt die Ergebnisse der Kollisionserkennung um eine optimale Bewegung zu planen und auszuführen. Zuletzt wird auf die derzeit existierende Kollisionserkennung eingegangen.

3.2.1 Normal Distribution Transform Map

Zur Lokalisierung von Robotern wird oft ein Simultaneous Localization and Mapping (SLAM) - Verfahren verwendet, welches die aktuelle Position des Roboters in der gleichzeitig erstellten Umgebungskarte verankert. Normale SLAM - Verfahren erstellen oft nur statische Umgebungskarten. Assistenzroboter bewegen ich in dynamischen Umgebungen. Deshalb muss der Algorithmus so angepasst werden, dass die Karte ständig aktualisiert werden kann, da sich die Objekte im Bezug auf die letzte Erfassung bewegt haben können.
Der vom Fachgebiet verwendete Algorithmus erstellt eine dreidimensionale Belegtheitswahrscheinlichkeitskarte (Voxel-Karte), die als Octree-Darstellung gespeichert wird. Die Belegtheitswerte werden je Scan des Sensors angepasst. Der Initialwert beträgt 0.5, der Belegtheitswahrscheinlichkeit liegt zwischen 0 und 1. Für jeden Freiraum wird der Wert verringert und für jedes erkannte Objekt der Wert erhöht. Zusätzlich zu der Belegung der einzelnen Zellen wird bei der NDT - Karte die Normalverteilung der Belegung versehen. Somit kann die Genauigkeit der Hinderniserkennung verbessert werden, da die Beschreibung nicht mehr von der Geometrie der Zelle abhängig sondern mit einem Elipsoid der Normalverteilung beschrieben werden kann ([Einhorn und

Gross, 2013]).

3.2.2 Evolutionary Motion Planner

In der Robotik ist die Bewegungsplanung meist in lokalen und globalen Planer aufgeteilt. Wobei der globale Planer den kompletten Pfad plant und der lokale Planer die Ansteuerungen optimiert und versucht dem globalen Pfad zu folgen.
Der Evolutionary Motion Planner ist ein objektbasierter Ansatz der Bewegungsplanung. Dabei werden eine Vielzahl von Trajektorien geplant und anhand von Restriktionen und dessen Kosten bewertet. Die Restriktionen können beispielsweise der Abstand zum Hindernis , die Orientierung zum Ziel oder die Fahrtrichtung sein. Je nach Wichtigkeit der Restriktion zum Gesamtziel können diese unterschiedlich stark gewichtet werden. Die optimale Trajektorie ergibt sich aus den besten Kostenverhältnis der Trajektorienschar. Eine große Anzahl von Trajektorien werden zu einem Planungszeitpunkt berechnet. Somit bleibt die Umgebung immer gleich.
Der lokale Planer plant einen gewissen Zeithorizont vorraus, damit die optimale Trajektorie gefunden werden kann. Viele Anfragen mit unterschiedichen Trajektorien für den Roboter finden in einer sich für diesen Zeitschritt nicht nicht ändernden Umgebung statt. Um bei einer Anfrage Zeit zu sparen, ist es möglich, die erhaltenen Eingangsdaten der Karte aufwändiger zu berechnen. Zum Zeitpunkt der eigentlichen Anfrage müssen nicht mehr die gesamten Daten berechnet werden. Ein Beispiel hierfür ist die Distanzstransformation für 2D Navigation. Die Kosten der Distanztransformation ist im Verhältnis recht hoch. Ist diese allerdings erfolgt, so ist die spätere Anfrage für eine bestimmte Roboterposition nur ein Punkt und somit sehr gering. Zur Berechnung des Optimums wird ein evolutionärer Algorithmus verwendet. Damit ist eine neue Trajektorie immer abhängig von der letzten Generation des Zeitschritts und die neue Population ist die beste (optimierte) Generation des vorherigen Schrittes ([Mueller et al., 2017]).

3.2.3 CollisionDetection 2D

Für das aktuelle System wird im Fachgebiet eine zweidimensionale Kollisionserkennung verwendet. Dabei wird der Umriss des dreidimensionalen Roboters auf auf den Boden als sogenannter *Footprint* projiziert. Für die Kollisionsberechnung wird der Footprint

von einen Kreis umschlossen. Eine Kollision ist dann vorhanden sobald sich ein Objekt mit dem Kreis schneidet.
Zur einfachen Bewegung des Roboters durch den Raum ist dieser Ansatz in der Berechnung gut geeignet. Da die Roboterplattform der Firma PAL Robotics (TIAGo) einen extra Greifarm besitzt, erweitern sich die Anwendungsszenarien. Aus diesem Grund ist die zweidimensionale Kollisionserkennung dafür ungeeignet und es wird im Laufe dieser Arbeit eine dreimensionale Kollisionserkennung entwickelt.

Kapitel 4

Eigener Ansatz

Im Kapitel 3.1.3 wurde verschiedene Bibliotheken zur Berechnung von Kollisionen und Abständen vorgestellt. Da die beiden Bibliotheken nicht mit der Middleware for Robotic Applications (MIRA) zusammen in Betrieb genommen werden konnten wurde ein eigener Ansatz zur Kollisionsberechnung erstellt.
Im folgenden Kapitel wird die Vorgehensweise dieses eigenen Ansatzes beschrieben. Dabei wird zuerst ein Überblick des Gesamtsystems gegeben. Es folgt eine genauere Beschreibung der verwendeten Hüllkörper sowie deren Berechnungen eingegangen. Im Anschluss daran werden die einzelnen Abstandstests beschrieben. Zuletzt wird das Software Design erläutert und anhand von einfachen Kollisionstest dargestellt und bewertet.

4.1 Versuchsaufbau

Für die Daten, die für diese Arbeit benötigt werden, gibt es für die Roboterplattform TIAGo eine von der Herstellerfirma PAL Robotics entwickelte Robot Operating System (ROS) Simulation [1]. Mithilfe dieser Simulation kann der Roboter über Konsolenbefehle oder anhand von Skripten in einem virtuellen Büro gesteuert werden. Die Abbildung 4.1 zeigt den Roboter in der Gazebo-Simulation des virtuellen Büros vor einem Tisch stehend. Die Simulationsdaten werden mittels eines vom Fachgebiet bereitgestellten Adapters, aus der ROS/Gazebo-Umgebung an das MIRA-Framework übertragen. Wie in Kaptiel 3.2 beschrieben, müssen für die Kollisionserkennung die erzeugt NDT-Karte

[1] http://wiki.ros.org/Robots/TIAGo

und die aktuelle Roboterkonfiguration bereitgestellt werden. Die Abbildung 4.1 gibt eine Übersicht der benötigten Daten von den jeweiligen Simulationen. Dabei werden die Tiefenbilddaten (Depth Image) für die Erzeugung der NDT-Map und der Roboter-State für die aktuelle Roboterkonfiguration gebraucht. Die RGB-Bilddaten werden nur zur Anzeige benötigt. Da derzeit noch kein Modul zur Ansteuerung des Roboters existiert, werden die Bewegungskommandos, wie eben beschrieben, über die Konsole ausgeführt.

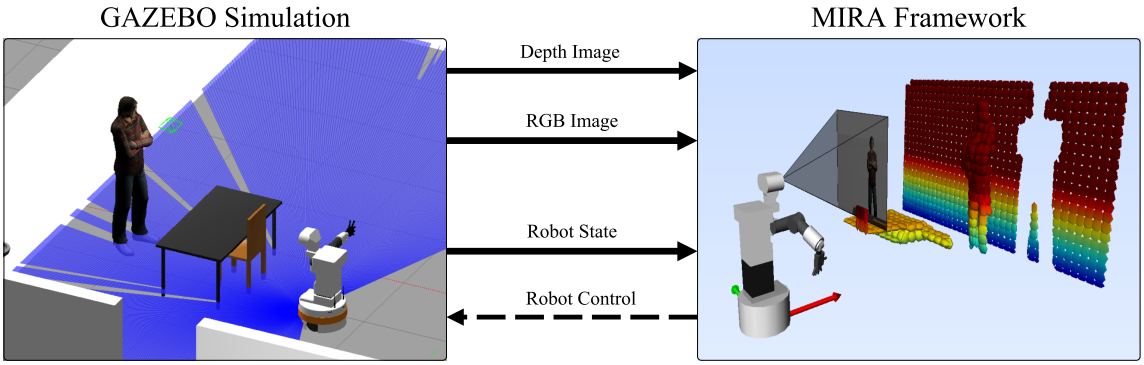


Abbildung 4.1: Übersicht der verwendeten Simulationsumgebung und den Transport der Daten zwischen GAZEBO und MIRA.

Das Kollisionserkennungsmodul, das im Zuge dieser Arbeit entstanden ist, kann aus den erzeugten Simulationsdaten die benötigten Informationen gewinnen und sein Berechnungen durchführen. Da die Normal Distribution Transform (NDT)-Karte aus achsenorientierten Boxen besteht und das Robotermodell anhand von primitiven Geometrien beschrieben ist, wird die Kollisionsberechnung mit diesen primitiven Hüllkörpern durchgeführt.

4.2 Verwendete Hüllkörper

Wie bereits beschrieben, bezeichnet der Begriff Hüllkörper meist einfache geometrische Körper, die komplexe dreidimensionale Gegenstände einschließen bzw. umhüllen. In der Kollosionsberechnung werden sie eingesetzt, um Testgeschwindigkeiten bei der Berechnung zu erhöhen. Im Rahmen dieser Arbeit werden folgende Hüllkörper verwendet, um die Bewegungen des Roboters und des Roboterarms auf Kollision zu berechnen:

- Kugel-Form (Sphere)
- Box (AABB und OBB)
- Zylinder (Cylinder)

Kugelform (Sphere)

Spheres sind, wie bereits aufgezeigt, die am einfachsten zu speichernden Hüllkörper. Der Speicheraufwand ist deshalb so gering, da nur Zentrum und Radius der Kugel aufgenommen werden müssen. Ein großer Vorteil sind die Überschneidungstests der Kugeln, die mir geringem Kostenaufwand erbracht werden können sowie ihre Konstanz gegenüber von Rotation. Bewegen sich Objekte bei dynamischen Anwendungen im Raum, muss die Kugel nicht neu berechnet, sondern einfach nur die Verschiebung auf den Mittelpunkt der Kugel genutzt werden([Ericson, 2005]). Die Berechnung der Kollision zweier Kugeln ist relativ einfach: Ist der Abstand zweier Kugelzentren größer als die Summe ihrer Radien (r_s), so schneiden sich die beiden Kugeln nicht.

$$d > r_s1 + r_s2 \tag{4.1}$$

Damit wird herausgefunden, ob eine Kollision generell stattfindet oder nicht. Die Distanz zwischen zwei Kugeln wird in Kapitel 4.3.1 näher beschrieben. Nachteil der Kugel als Hüllkörper ist, dass die Hüllkörpereffizienz bei vielen Objekten nicht gegeben ist. Gerade bei länglichen Objekten ist die Kugel als Hüllkörper eher ungeeignet ([Petersen, 2007]).

Box (AABB und OBB)

Axis Aligned Bounding Box (AABB) sind Quader, die sich beim Einhüllen der Objekte, nicht an der Ausrichtung des Objekts selbst, sondern an den Achsen des Weltkoordinatensystems orientieren. Auch bei AABBs ist ein Kollisionstest verhältnismäßig einfach durchzuführen. Es müssen nur die Eckpunkte des AABB für jede Achse einzeln verglichen werden 4.1.

Auch hier wird nur herausgefunden, ob sich die beiden Hüllquader überhaupt schneiden. Die Distanzberechnung findet sich hierfür in Kapitel 4.3.6 Oriented Bounding Boxs (OBBs) sind im Vergleich dazu komplexer in ihrer Berechnung, da sie nicht achsenorientiert sind und somit nicht nur drei Achsen sondern maximal 15 Achsenkombinationen getestet werden müssen.

Konvertierung der Hüllkörperklassen

Algorithmus 4.1: Überschneidungstest AABB

Input : AABB1
AABB2
Output : Kollision / keine Kollision

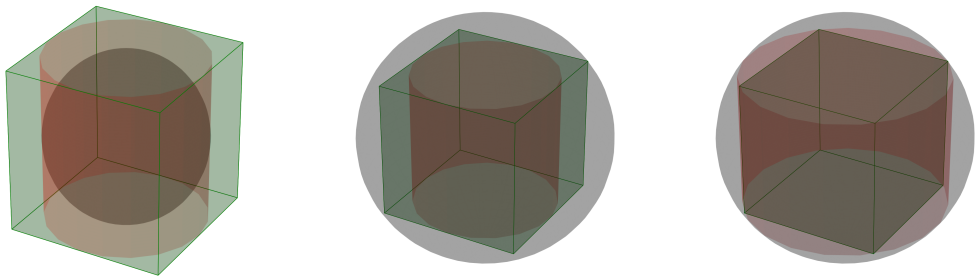
```
1  if ( AABB1.maxCorner.x < AABB2.minCorner.x )  OR
2  ( AABB2.minCorner.x > AABB1.maxCorner.x ) then
3  |  return keine Kollision
4  if ( AABB1.maxCorner.y < AABB2.minCorner.y )  OR
5  ( AABB2.minCorner.y > AABB1.maxCorner.y ) then
6  |  return keine Kollision
7  if ( AABB1.maxCorner.z < AABB2.minCorner.z )  OR
8  ( AABB2.minCorner.z > AABB1.maxCorner.z ) then
9  |  return keine Kollision
10 return Kollision
```

Die Hüllkörper werden in eigenen Umsetzung als eigene Klassen angelegt. Dabei können die Hüllkörper jeweils in eine andere Hüllkörperklasse konvertiert werden. Zu beachten ist hierbei aber, dass die Hüllkörper so konvertiert werden, dass die Hüllkörpereffzienz der Ausgangsprimitive nicht verbessert, sondern eher verschlechtert wird (vgl. Abb. 4.2). So wird beispielsweise bei der Konvertierung einer Kugel zu einem Zylinder der Zylinderkörper um die Kugel erstellt. Diese Konvertierung hat somit nur Einfluss auf die Berechnung der Hüllkörper untereinander und nicht auf das Objektfitting (vgl. Abb. 4.2a).

Somit kann für ein Objekt, das nur eine Box als Hüllkörper besitzt, eine Kugel als Hüllkörper erzeugt werden, um einen schnelleren Kollisionstest zu ermöglichen.

4.3 Kollisionstests

Die Kollisionstest zwischen den einzelnen Hüllkörpern sind anhand ihrer Definitionen aufgeteilt und können untereinander abgetestet werden. Für die berechneten Kollisions-



(a) Kugel **(b)** Zylinder **(c)** Box

Abbildung 4.2: Konvertierung der verschiedenen Hüllkörper untereinander.
Bei der Konvertierung eines Hüllkörper zu einen anderen, wird der Ausgangshüllkörper als einzuhüllendes Objekt genommen. Unter a) ist der Ausgangshüllkörper eine Kugel, die in die anderen Hüllkörper überführt wird. Bei b) ist der Ausgangshüllkörper ein Zylinder und bei c) eine Box.

distanzen ($D_{Collision}$) gilt immer die Interpretation von 4.2.

$$D_{Collision} = \begin{cases} > 0 & \text{Keine Kollision, Distanz entspricht Freiraum} \\ = 0 & \text{Kontakt, Kein Freiraum} \\ < 0 & \text{Kollision der Objekte miteinander} \end{cases} \tag{4.2}$$

4.3.1 Kugel gegen Kugel

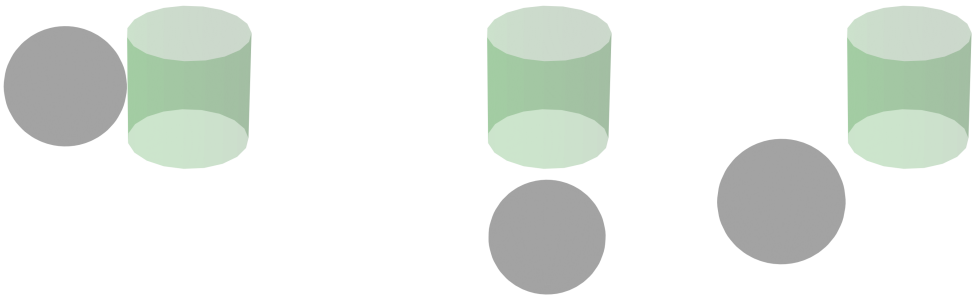
Zwei Kugeln auf Kollision zu testen ist der einfachste und somit auch schnellste Test. Hierbei wird die Distanz der beiden Mittelpunkte der Kugeln berechnet und die Summe der Radien davon abgezogen. Somit ergibt sich die folgende Formel 4.3:

$$D_{Collision} = d_{s12} - r_{s1} - r_{s2} \tag{4.3}$$

Dabei ist d_{s12} die Distanz zwischen den beiden Mittelpunkten der Kugeln, r_{s1} der Radius der ersten Kugel und r_{s2} der Radius der zweiten Kugel.

4.3.2 Kugel gegen Zylinder

Der Kollisionstest einer Kugel gegen einen Zylinder kann auf den Abstandstest zwischen Punkt und Strecke aus Kapitel 2.1.2 zurückgeführt werden. Zur Berechnung des Abstands zwischen Kugel und Zylinder müssen drei mögliche Fälle betrachtet werden (vgl. Abbildung 4.3):



(a) Fall 1 **(b)** Fall 2 **(c)** Fall 3

Abbildung 4.3: Übersicht der möglichen unterschiedlichen Lagen einer Kugel um einen Zylinder
a) Kugel liegt seitlich des Zylinders. b) Kugel liegt komplett unterhalb des Zylinders. c) Kugel liegt schräg zum Zylinder.

Anhand dieser Fälle muss die Berechnung der Kollisionsdistanz angepasst werden, da nicht in jedem Fall der Radius der Kugel r_s oder der Radius des Zylinders r_c von dem berechneten Abstand d_{sc} subtrahiert werden müssen. Somit ergibt sich für den ersten Fall (Abbildung 4.3a) der Kollisionsabstand nach der Formel 4.4. Hier müssen beide Radien berücksichtigt werden, solange sich der Kugelmittelpunkt (c_s) im Bereich der Zylinderhöhe (h_c) befindet.

$$D_{Collision} = d_{sc} - r_s - r_c \qquad 0 \leq c_s \leq h_c \tag{4.4}$$

Im zweiten Fall (vgl. Abb. 4.3b) liegt die Kugel unterhalb des Zylinders. Solange sich der Kugelmittelpunkt nicht außerhalb des Zylinderradius befindet fließt nur der Kugelradius bei der Abstandsberechnung mit ein. Daraus ergibt sich die Formel 4.5. Dieses Verhalten ist symmetrisch und kann ebenso auf die Kugellage oberhalb des

Zylinder angewandt werden.

$$D_{Collision} = d_{sc} - r_s \qquad -r_c \leq c_s \leq r_c \qquad h_s < c_s > h_s \tag{4.5}$$

Im dritten Fall (vgl. Abb.4.3c) ist die Berechnung des Kollisionsabstandes anhand der Radien nicht mehr trivial. Deshalb wird hier zur Abstandsberechnung der Zylinder als „Pille“(der Radius des Zylinders vergrößert die Enden um Halbkugeln) angenommen.Aus diesem Grund kann die Formal 4.4 auch hier angewendet werden. Der hier entstehende minimale Kollisionsdistanz-Offset wird in dieser Arbeit hingenommen.

4.3.3 Kugel gegen Box

Die Distanzberechnung einer Kugel zu einer Box ist eine erweiterte Form der Punkt-zu-Ebenen Berechnung aus Kapitel 2.1.2. Eine Box ist über ihr Zentrum, der Weite, der Breite und der Tiefe definiert. Mit diesen Informationen lässt sich die Ebene um das Zentrum mittels der Weite und Breite aufspannen. Die Größenbeschreibungen der Box sind dabei Vektoren mit je nur einem Element ungleich 0 und stehen orthogonal zueinander. Die Tiefe entspricht dann dem Normalenvektor der Ebene.
Durch die Projektion des Kugelzentrums P_s auf die einzelnen Vektoren, die die Ebene beschreiben, kann der Punkt P_b auf der Box, der am nächsten zur Kugel steht, bestimmt werden. Ist die Entfernung des projizierten Punktes zum Zentrum der Box größer als der Abstand der halben Seitenlänge, dann liegt der Punkt außerhalb der Box auf dieser Achse. Andernfalls liegt er innerhalb. Dies wird durch den unten aufgeführten Pseudocode 4.2 verdeutlicht. Die Distanz zur Kugel entspricht der Länge des Vektors zwischen P_s und P_b. Somit ergibt sich die Formel 4.6 für den Kollisionsabstand $D_{Collision}$.

$$D_{Collision} = \|P_s - P_b\| - r_s \tag{4.6}$$

4.3.4 Zylinder gegen Zylinder

Bei der Abstandsberechnung von zwei Zylindern wird auf die in Kapitel 2.1.2 beschrieben Abstandsberechnung von zwei Linien zurückgegriffen. Da Zylinder unterschiedliche Lagen zueinander einnehmen können, werden beide Zylinder zur Vereinfachung als „Pillen“angesehen. Dadurch entstehen eventuell etwas kleinere Abstandsergebnisse als

Algorithmus 4.2: Abstandsberechnung Kugel Box

Input : Sphere
Box

Output : $D_{Collision}$

1 $\underline{\mathbf{v}} = \begin{pmatrix} Box.Width \\ 0 \\ 0 \end{pmatrix}$, $\underline{\mathbf{w}} = \begin{pmatrix} 0 \\ Box.Height \\ 0 \end{pmatrix}$, $\underline{\mathbf{n}} = \begin{pmatrix} 0 \\ 0 \\ Box.Depth \end{pmatrix}$

2 $x = min(Box.Width * 0.5\ , proj_{\underline{\mathbf{v}}} P_{Sphere})$

3 **if** $proj_{\underline{\mathbf{v}}} P_{Sphere}.x < 0$ **then**

4 $\quad x = -x$

5 $y = min(Box.Height * 0.5\ , proj_{\underline{\mathbf{w}}} P_{Sphere})$

6 **if** $proj_{\underline{\mathbf{w}}} P_{Sphere}.y < 0$ **then**

7 $\quad y = -y$

8 $z = min(Box.Depth * 0.5\ , proj_{\underline{\mathbf{n}}} P_{Sphere})$

9 **if** $proj_{\underline{\mathbf{n}}} P_{Sphere}.z < 0$ **then**

10 $\quad z = -z$

11 $P_{Box} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

12 **return** $\|P_{Sphere} - P_{Box}\| - Sphere.Radius$

diese tatsächlich sind. Da der hier entstehenden möglichen Berechnungsfehler allerdings zu einer früheren und nicht zu einer zu späten Kollisionserkennung führt, wird der Fehler akzeptiert. Außerdem bedeutet die genaue Lagenerkennung weitere Berechnungsschritte. Somit ergibt sich die Formel 4.7 aus den Abstand der Strecken (d_{c12}) und den beiden Radien (r_{c1} und r_{c2}) der Zylinder.

$$D_{Collision} = d_{c12} - r_{c1} - r_{c2} \tag{4.7}$$

4.3.5 Zylinder gegen Box

Zur Berechnung des Abstandes zwischen einem Zylinder und einer Box wird sich auf die im Kapitel 2.1.2 beschriebenen Berechnungen zwischen einer Strecke und einer Ebene bezogen. Anhand der Mittelachse des Zylinders kann eine Strecke erzeugt werden, die mit der Ebene, die durch die Box aufgespannt ist, auf Kollision getestet wird. Zudem wird der Abstand zueinander berechnet. Dazu muss zunächst überprüft werden, ob der Zylinder und die Box parallel liegen oder ein Schnittpunkt mit der Ebene existiert. Der nächste Punkt der Strecke des Zylinders zur Ebene muss auf den Normalenvektor der Ebene projiziert werden, um den Abstand zur Ebene zu erhalten. Die drei orthogonal liegenden Vektoren der Ebene entsprechen gleichzeitig der Größenbeschreibung der Box, weshalb der Punkt ebenso auf die anderen beiden Vektoren projiziert werden muss. Das weitere Vorgehen entspricht dann dem zur Berechnung des Abstandes Kugel zu Box 4.2. Demnach ergibt sich die Distanz zwischen Zylinder und Box anhand der Länge des Vektors der am nächsten liegenden Punkte zwischen beiden Objekten und dem Radius des Zylinders r_c, wie in der Formel 4.8 dargestellt.

$$D_{Collision} = \|P_c - P_b\| - r_c \tag{4.8}$$

4.3.6 Box gegen Box

Um den Abstand zwischen zwei Boxen zu bestimmen, werden alle acht Eckpunkte der Box auf die Vektoren, die die Ebene aufspannen, projiziert. Damit kann der Punkt P_{b1} mit dem kleinsten Abstand zur Ebene gefunden werden. Dieses Verfahren muss für die anderen Box erneut wiederholt werden. Die Distanz zwischen den Boxen entspricht dem Vektors zwischen den gefundenen minimalen Abstandspunkte P_{b1} und P_{b2}.

$$D_{Collision} = \|P_{b1} - P_{b2}\| \tag{4.9}$$

4.4 Software Design

Das Software Design der in dieser Arbeit erstellten Toolbox orientiert sich an dem existierender Physik-Engines, wie beispielsweise Bullet. Anhand des *Collision Manager* können die verschiedenen Objekte hinzugefügt und entfernt werden, sowie die Kollisionsergebnisse abgerufen werden. Jedes Objekt wird intern in eine *Collision Scene* umgewandelt die eine Hüllkörperhierarchie-Baumstruktur enthält. Dabei ist jeder Knoten des Baumes ein *Collision Node* der unter anderem die Hüllkörpergeometrien enthält. Die Abbildung 4.4 veranschaulicht diesen Aufbau.

4.4.1 Collision Node

Die *Collsion Node* ist des kleinste Element der Kollisionserkennung wie sie in dieser Arbeit umgesetzt wurde (Abbildung 4.4). In den Node sind die Informationen zu dem Elter-Node und möglichen Kinder-Nodes gespeichert. Da das Robotermodell aus Joints (Gelenken) und Links (Verbindungsgeometrien) besteht, werden die beiden Komponenten in einem Node zusammengefasst. Somit beschreibt ein Node für das Beispiel des Roboters ein Joint und dessen Links, die davon zu eventuellen Kindern oder Endpunkten abgehen. Da in den Links mehrere Geometrien enthalten seien können, werden diese zu einem Hüllkörper, dem Node-Volumen, zusammengefasst.
Für die hierarchische Kollisionsberechung besitzt jeder Node zusätzlich zu seinen Geometrien noch die Information über die Hüllkörper seiner Kinder. Zusammen mit dem eignen Node-Volumen und den Node-Volumen aller darunter liegender Nodes wird Branch-Volumen erzeugt. Die Abbildung 4.5 verdeutlicht noch einmal den Zusammenhang zwischen den beiden Volumen die jeder Node besitzt. Hierbei wird ein Node aus

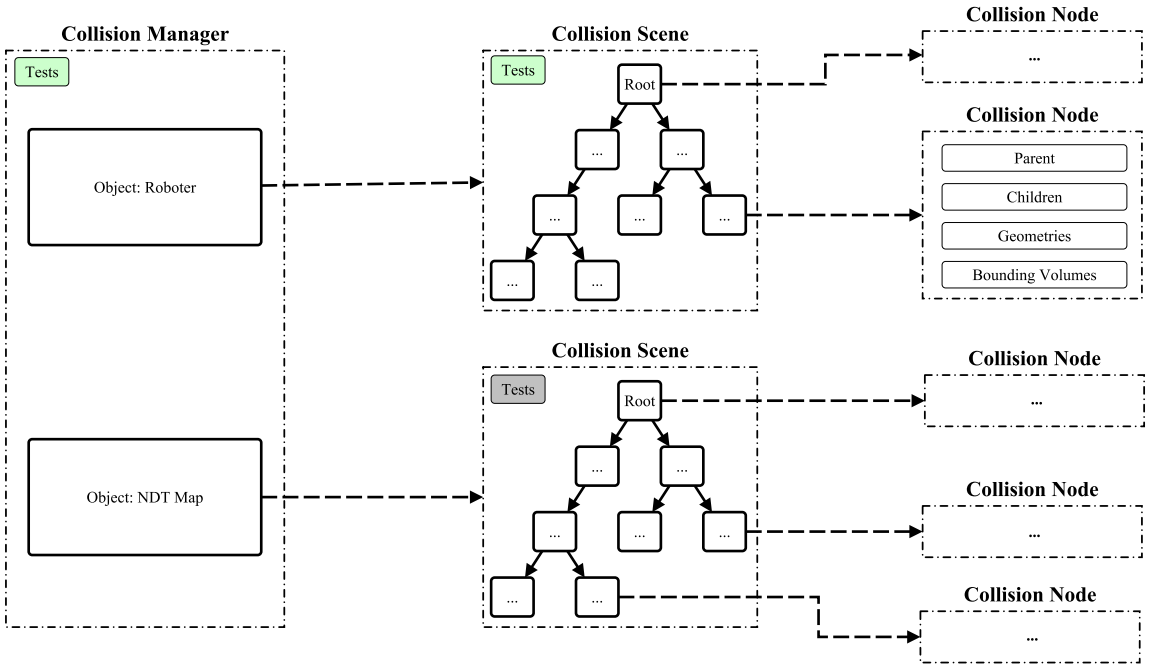
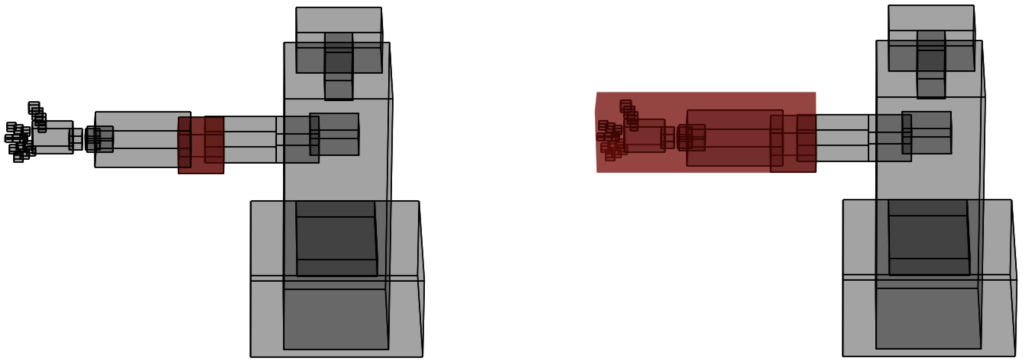


Abbildung 4.4: Software Design der erarbeiteten *Collision Detection Toolbox. Ein grünes Tests-Feld beschreibt aktivierte Tests für diesen Teil, wohingegen ein graues Feld angibt, dass die Tests deaktiviert sind.*

dem Roboterplattform der Firma PAL Robotics (TIAGo)-Modell farblich hervorgehoben. Die Abbildung 4.5a zeigt ein Node-Volumen des Robotermodells, wohingegen die Abbildung 4.5b das Branch-Volumen desselben Nodes zeigt.
Für die NDT-Karten ist das Node- gleich dem Branch-Volumen, da die Information durch die räumliche Partitionierung der unterschiedlichen Ebenen der Elemente schon gegeben ist.

4.4.2 Collision Scene

Eine *Collision Scene* repräsentiert ein Robotermodell oder eine NDT-Karte, als einen Baum aus *Collision Nodes* (Abbildung 4.4). Durch den hierarchisch Informationsfluss der Branch-Volumen der Kinder-Nodes, wie in 4.4.1 beschrieben wurde, können die Anzahl der Tests zur Selbstkollisionserkennung der Scene verringert werden. Denn schneiden sich zwei Nodes nicht, unter Berücksichtigung deren Branch-Volumen, so sind diese beiden Äste untereinander kollisionsfrei.



(a) Node-Volumen **(b)** Branch-Volumen

Abbildung 4.5: Darstellung des Node- und Branch-Volumen eines CollisionNodes *Gesamte Roboterdarstellung anhand aller CollisionNodes des Baumes: (a) zeigt das NodeVolumen eines Nodes; (b) zeigt das BranchVolumen des Nodes.*

Für die Erzeugung einer Collision Scene exisitieren zwei mögliche Varianten. Zum einen kann eine Scene auf Basis der Roboterbeschreibung (RigidModel) von Joints und Links erzeugt werden. Ausgehend von einem Root-Node wird der Baum entlang der Roboterbeschreibung aufgebaut. Die Joints bilden die Nodes und die Links befüllen diese mit den entsprechenden Geometrien, wie in Kapitel 4.4.1 schon beschrieben wurde.
Die zweite Variante erzeugt eine Collsion Scene anhand einer NDT-Karte. Die Karte beinhaltet unter anderem eine Octree-Beschreibung der Umgebung, in dem der Raum in 17 (16 bis 0) verschiedene Ebenen aufgeteilt wurde. Eine Ebene unterteilt eine Zelle dabei in acht gleiche Teilbereiche (Quadrate) und die Ebene 16 ist die erste Unterteilung des Raums der Umgebung mit mehr als 6000m Seitenlänge. Die Zellen mit der Ebene 0 hingegen besitzen eine Seitenlänge von 0,1m und stellt damit die maximale Genauigkeit in dieser Arbeit erstellten Kollisionserkennung dar. Die Genauigkeit kann durch die von der NDT Karte erzeugten Ellipsoiden aus der Normalverteilung innerhalb der Zellen noch verbessert werden, jedoch wird die auf Kollsionsfreiheit zu testende Geometrie wieder komplizierter. Daher werden die Ellipsoiden in dieser Arbeit nicht verwendet.
Der Aufbau der Scene anhand der Octree-Beschreibung erfolgt mit Hilfe einer Umkreissuche belegter Zellen um den Roboter. Für eine Kollisionserkennung ist es nur wichtig die jeweiligen Hindernisse im Nahbereich zu finden und zu prüfen. Außerdem muss eine sinnvolle Zellengröße für der höchsten Ebene definiert werden. Die Zellengröße von

Ebene 16 mit mehr als 6000m Seitenlänge ist für den Nahbereich nicht empfehlenswert. Um sicherzustellen, dass die Nodes einer Collision Scene nicht untereinander kollidieren, werden anhand der Kollisionstests aus Kapitel 4.3 *Selbstkollisionstests* durchgeführt. Bei Hindernissen, die in der Umgebung liegen, ist es nicht zielführend diese auf Kollision untereinander zu prüfen. Daher sind die Tests für eine Collision Scene auf NDT Karten Basis immer deaktiviert (vgl. Abbildung 4.4).

Da die CollisonScene nur bei der Erzeugung durch eine Roboterbeschreibung auf Selbstkollision getestet werden muss, wurde die Bedingung an den Algorithmus gestellt, dass keine Nodes mit ihren direkten Kindern auf Kollision getestet werden sollen. Aufgrund des Robotermodells besitzen die direkten Kinder einen Berührungspunkt zum Elter-Node, da sie an diesen befestigt sind.
Der Algorithmus prüft somit das Branch-Volumen der Kindeskinder mit dem Node-Volumen des aktuellen Nodes. Kommt es hier zu einer Kollision, so gibt es entweder irgendein Kind in der aktuellen oder unteren Kinderebene, welches mit dem Node kollidiert oder das Branch-Volumen schneidet im freien Bereich wegen der schlechteren Hüllkörpereffizienz das Node-Volumen. Eine weiterer Überprüfung ist notwendig. Als nächstes wird das Node-Volumen mit dem aktuellen Kindes-Kind-Branch-Volumen getestet. Bei Kollision schneiden sich die beiden Node und der Test ist mit dem Ergebnis einer Kollision zu Ende. Gibt es keine Kollision muss geprüft werden ob die vorherige Kollision durch ein Kinder der unteren Ebenen ausgelöst wurde. Der Ast des aktuellen Kindes-Kinds weiter nach unten abgetestet. Zuerst wird wieder das Node-Volumen mit dem Branch-Volumen getestet. Kommt es zu einer Kollision, werden die beiden Node-Volumen getestet. Dies erfolgt solange bis eine Kollision zwischen zwei Node-Volumen auftritt oder kollisionsfrei am Ende des Astes angekommen ist. Hierbei ist zu beachten, dass ein Ast vollständig kollisionsfrei ist, wenn alle Kindes-Kinder einer Ebene kollisionsfrei sind. Der Algorithmus 4.3 zeigt den Ablauf des Selbstkollisionstests in einer Pseudocodedarstellung.

Algorithmus 4.3: Vorgehen des Tests zur Selbstkollision

Input : Node
Output : CollisionResult (True/False)

```
1  START:
2  if Node != LeafNode then
3      for each Node.Child do
4          CALL:
5          for each Node.Child.Child do
               // Test des Node Volumen gegen das Branch Volumen des
                  Kindeskind
6              CollisionResult =Test( Node.NodeVolume,
                Node.Children.Child.BranchVolume )
7              if CollisionResult == True then
                   // Test der Node Volumen der beiden Nodes
8                  CollisionResult = Test( Node.NodeVolume,
                    Node.Children.Child.NodeVolume )
9                  if CollisionResult == True then
                       // Kollision der beiden Nodes miteinander
10                     return True
11                 else
                       // Abstieg im Baum zur nächsten Kinderebene
12                     Node.Child = Node.Child.Child
13                     goto CALL
14         if CollisionResult == True then
               // Kollision mit einem unteren Node
15             return True
16         else
               // Node ist Kollisionfrei, erneuter Durchlauf mit Kind
17             Node = Node.Child
18             goto START
19 return CollisionResult
```

Um die Vorgehensweise des Selbstkollisionsalgorithmus, besonders im Bezug auf die jeweiligen Hüllkörper, zu vertiefen, wird auf die Abbildung 4.6 verwiesen. Anhand eines einfachen Baums wird hier die Überprüfung eines Nodes auf Kollision mit anderen Nodes des Baumes dargestellt. Der aktuelle Node wird dabei mit einem Pfeil im Baum markiert. Das Objekt mit denen der Hüllkörper des Nodes kollidiert wird in der Abbildung 4.6 mit einem roten Kreuz gekennzeichnet. Bei Kollisionen mit einen anderen Node-Volumen wird der Baum nicht weiter durchlaufen, da eine Kollision vorliegt. Ist der Node kollisionsfrei im Baum wird der Test mit einem anderen Node fortgesetzt (Pfeil auf Node).

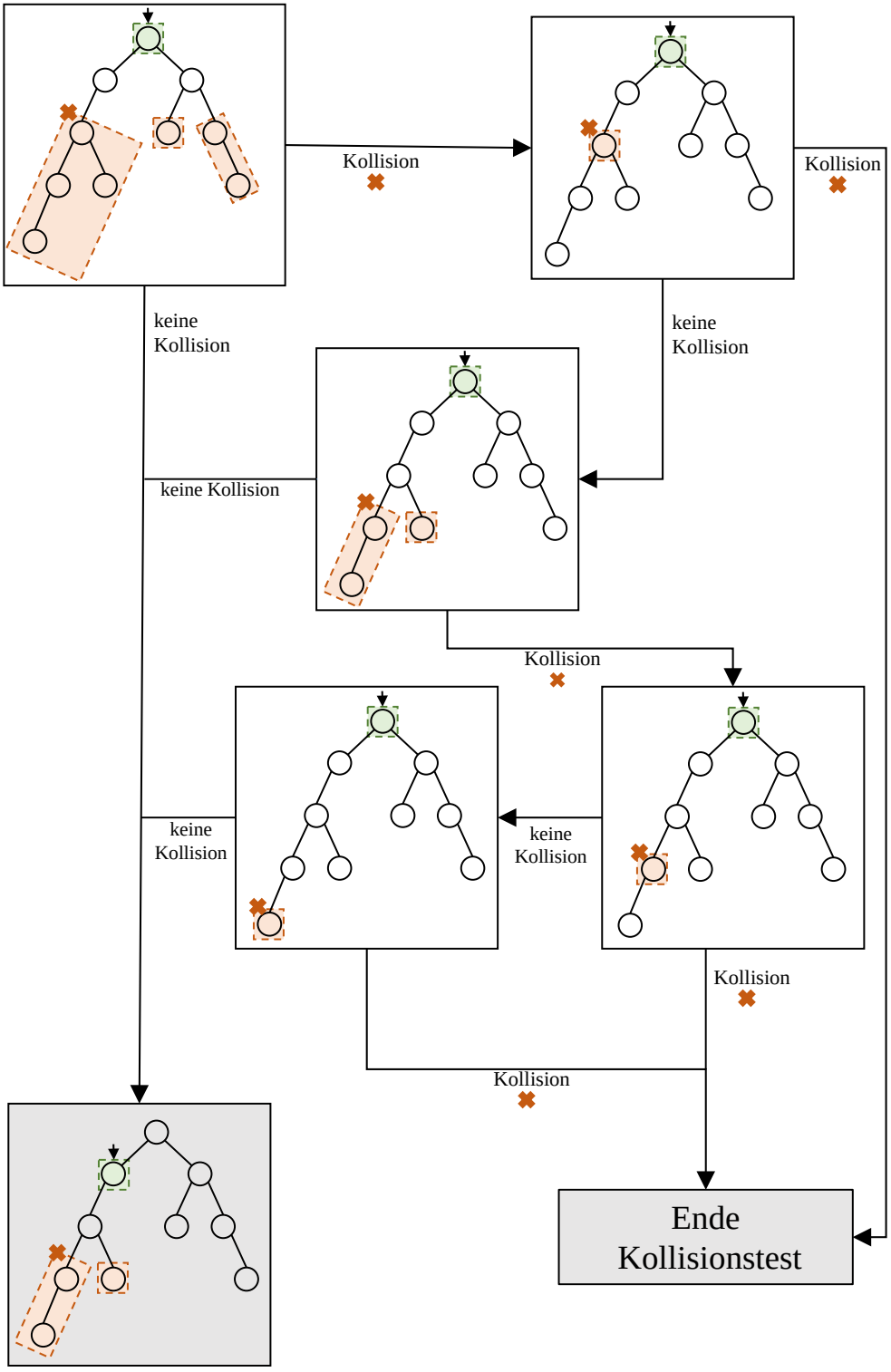


Abbildung 4.6: Übersicht des Testalgorithmus zur Selbstkollision

4.4.3 Collision Manager

Der *Collision Manager* bildet die Schnittstelle zum Anwender und vereinigt die Kollisionstests der Objekte miteinander. Mithilfe des Managers können verschiedene Objekte hinzugefügt werden, die dann in einer Collision Scene abgespeichert werden. Außerdem können nicht mehr benötigt Objekte wieder entfernt werden. Nachempfunden ist dieser Ansatz aus dem Software Design der Bibliotheken, die für diese Arbeit verwendet werden sollten.
Die Aufgabe der Scene ist es, sich selbst auf Selbstkollision zu prüfen, wenn dies gewollt ist. Im Manager werden die Kollisionstests zwischen den verschiedenen Objekten durchgeführt. Dabei werden zuerst die Branch-Volumen der Root-Nodes miteinander verglichen. Existiert hier keine Kollision ist der Test vorbei. Wurde zwischen den beiden Root-Nodes eine Kollision erkannt, werden die Hüllkörper der Kinder-Nodes gegenseitig geprüft. Auf der Ebene der Kinder müssen alle Kinder-Nodes untereinander kollisionsfrei sein. Erst wenn dies der Fall ist, dann kollidieren die beiden Objekte nicht miteinander. Kollidieren mindestens zwei Kinder-Nodes miteinander, wird der Test auf einer weiteren Ebene fortgeführt. Der Test läuft solange, bis zwischen den Objekten eine Kollisionsfreiheit oder eine Kollision besteht.

Die NDT-Karten besitzen eine andere Struktur des Baumes, da hier kein Unterschied zwischen Node-Volumen und Branch-Volumen existiert. Deshalb wurde der Suchalgorithmus angepasst. In Abbildung 4.7 wird der Ablauf dieses Vorgehens dargestellt. Bei der Prüfung einer Scene aus einer Roboterbeschreibung mit einer Scene aus einer NDT-Karte, wird zuerst der komplette NDT-Kartenbaum abgearbeitet und mit dem RootNode der Scene des Robotermodells verglichen. Im Anschluss daran werden die jeweiligen Kinder-Nodes der Karte und der Roboter-Root-Node überprüft. Wird erneut eine Kollision mit einem der Kinder-Nodes erkannt, werden eine Ebene tiefer die Nodes geprüft. Dies geschieht so lange, bis eine Kollision in der untersten Ebene des Kartenbaums mit dem Root-Node des Roboters detektiert wird oder vorher eine Ebene kollisionsfrei ist. Erst dann wird der Baum des Roboters durchlaufen und mit den kollidierenden Nodes der untersten Ebene des Kartenbaumes verglichen. Dieses Vorgehen zeigt im Folgenden der Algorithmus 4.4 anhand eines Pseudocodes.

Algorithmus 4.4: Vorgehen des Tests zur Objektkollision

Input : Node1 (Robot)
Node2 (NDT)
Output : CollisionResult (True/False)

```
1  START:
   // Test der Branch-Volumen der aktuellen Nodes
2  CollisionResult = Test( Node1.BranchVolume, Node2.BranchVolume )
3  if CollisionResult == True then
4  |  for Each Node2.Child do
   |  |  // Eine Ebene tiefer in NDT Scene
5  |  |
6  |  |  Node2 = Node2.Child
7  |  |  goto START                          // Rekursiver Funktionsaufruf
8  |  |
9  |  if CollisionResult == True then
10 |  |  CollisionResult = Test(Node1.NodeVolume, Node2.NodeVolume)
11 return CollisionResult
```

Wie bei Abbildung 4.6 werden auch hier die Nodes, die getestet werden, mit einem Pfeil markiert. Zudem wird eine Kollision erneut mit einem roten Kreuz gekennzeichnet. Aus Übersichtlichkeitsgründen wurde der Ablauf verkürzt dargestellt. Die nicht dargestellten Tests wurden mittels dreier Punkte symbolisiert.

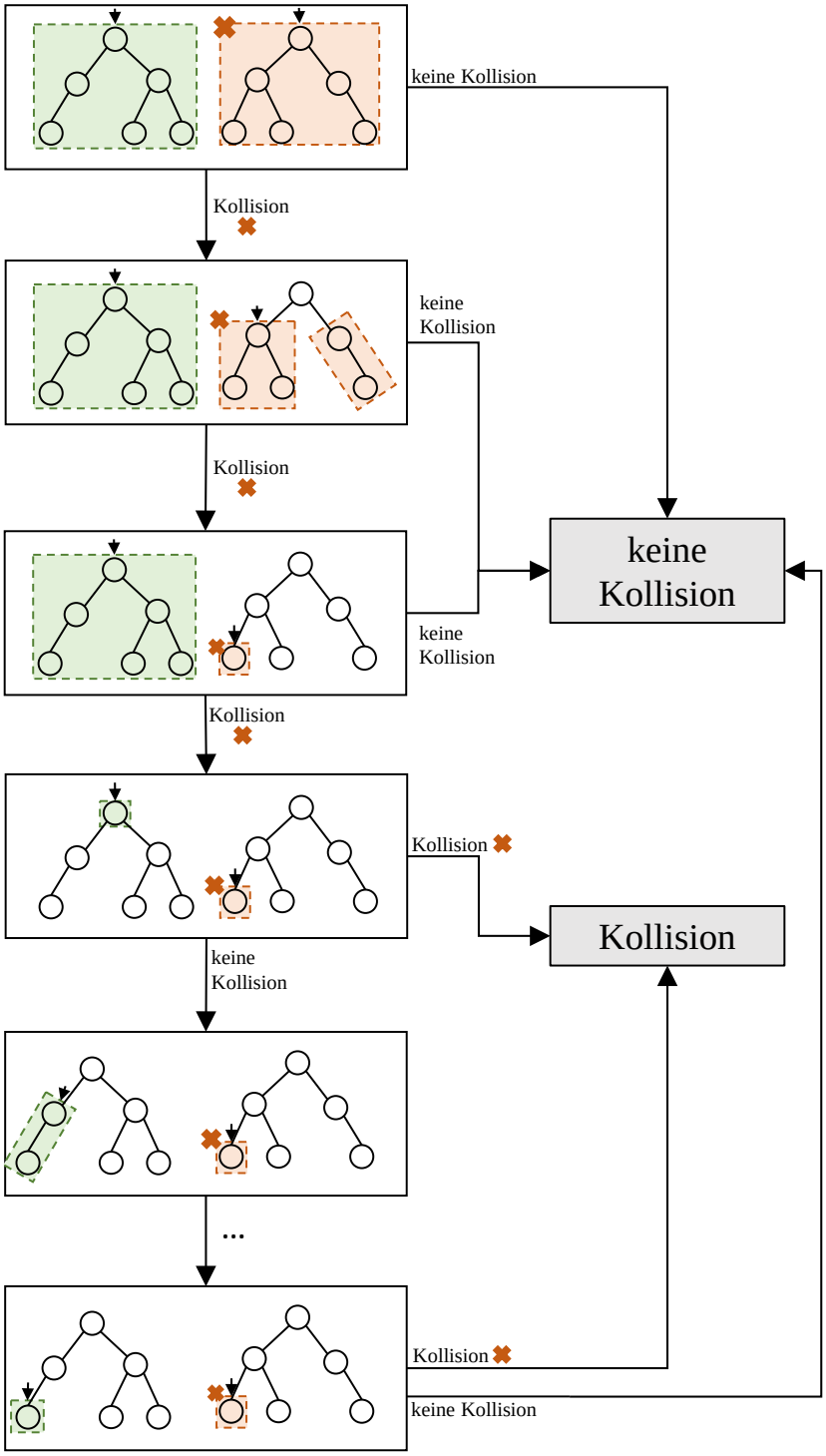
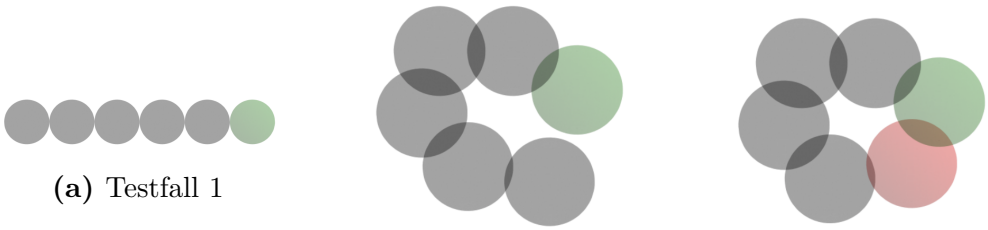


Abbildung 4.7: Übersicht des Testalgorithmus zur Kollisionserkennung zwischen zwei Objekten

4.5 Bewertung des eigenen Ansatzes

Das Gesamtsystem wurde mit einer virtuellen Maschine (Ubuntu 16.04 LTS) auf einem MacBookPro betrieben. Die folgen Ressourcen wurden der virtuellen Maschine zur Verfügung gestellt.

- 8 GB DRAM 667 MHz
- Intel Core i7-8559U CPU (2.70 GHz x 4)
- Intel Iris Plus Graphics 655



(a) Testfall 1 **(b)** Testfall 2 **(c)** Testfall 3

Abbildung 4.8: Testfälle 1 bis 3 zum Test der Selbstkollision.
Das grün markierte Objekt stellt den Root-Node dar. (a) Lange Kette aus Kugeln; (b) eingerollte Kette ohne Kollision; (c) eingerollte Kette mit Kollision (rotes Objekt)

Zur Bewertung des eigenen Ansatzes wurden zunächst drei einfache Testfälle (vgl. Abb. 4.8) erstellt. Getestet wurden jeweils Kugeln gegeneinander. Diese drei Fälle unterscheiden sich folgendermaßen:

- Testfall 1: Dieser Testfall beinhaltet die Beschreibung einer geradlinigen Kette aus Kugeln. Der Testfall wurde so aufgebaut, dass keine Kollision möglich ist und sich die Branch-Volumen nicht mit den vorhergehenden Nodes schneiden (vgl. Abb.4.8a).
- Testfall 2: Hier findet sich eine zusammengerollte Kette, die ebenfalls aus Kugeln besteht. Diese Kugeln kollidieren nicht miteinander. Die Branch-Volumen führen allerdings zu Kollisionen mit den vorherigen Nodes (vgl. Abb. 4.8b).
- Testfall 3: Auch hier findet sich eine zusammengerollte Kette aus Kugeln. Allerdings kollidieren diese miteinander. Somit kollidieren auch die Branch-Volumen mit dem vorherigen Nodes (vgl. Abb. 4.8c).

Bei der Auswertung dieser einfachen Kollisionstests werden die Anzahl an benötigte Kollisionsberechnungen, das entsprechende Ergebnis und der Durchschnitt der benötigten Berechnungszeiten betrachtet. Die einzelnen Ergebnisse sind in der Tabelle 4.1 zu finden.

Testfall	**Berechnungen**	**Ergebnis**	**Zeit [ms]**	**Zeit [ms] ***
1	4	keine Kollision	61	6
2	15	keine Kollision	65	6
3	8	Kollision	56	5

Tabelle 4.1: Auswertung der Kollisionstests mit Kugeln als Hüllkörper (* Berechnungzeit ohne Hüllkörpererzeugung)

Bei der Anzahl der Kollisionsberechnungen sind aufgrund der vorhandenen sechs Nodes maximal 10 Testungen notwendig, um alle Nodes einzeln untereinander zu berechnen. Dies liegt daran, dass bei der Berechnung jeweils der Node, der direkt am Testnode anliegt, übersprungen wird, da hier generell eine Kollision aufgrund des Kontaktes vorhanden ist bzw. angenommen wird. Bei Testfall 1 werden insgesamt vier Tests benötigt. Dies entspricht den erwarteten vier Branch-Volumen-Tests, da der Algorithmus so aufgebaut ist, dass eine Ebene kollisionsfrei im Baum ist, sobald die Branch-Volumen der Kindeskinder-Nodes sich nicht mit dem Node-Volumen schneiden. Bei Testfall 2 dagegen wurden insgesamt 15 Testungen durchgeführt. Dies übertrifft die maximal notwendige Anzahl an Berechnungsschritten.
Testfall 2 beschreibt daher das ungünstigste Szenario für die Baumstruktur. Hier werden zusätzlich zu den einzelnen Node-Volumen-Tests auch die gesamtem Branch-Volumen-Tests hinzugefügt. Das passiert, da alle Hüllkörper für diese Tests Kugeln sind, sowohl die Nodes als auch die Branch-Volumen. Diese hohe Anzahl an Tests ist vermutlich auf die ungünstige Hüllkörpereffizienz der Kugel zurückzuführen. Bei Testfall 3 dagegen sind insgesamt 8 Tests berechnet worden. Hierbei wurde der komplette Baum durchlaufen, um die Kollision zwischen dem ersten und letzten Node festzustellen.

Die einzelnen Ergebnisse, die bei den Testfällen aufgetreten sind, entsprechen den geplanten Szenarien und somit den Erwartungen. Generell sind hohe Berechnungszeiten, um einen Baum zu durchlaufen, von der Erzeugung der Hüllkörper abhängig. Die

Berechnungszeiten mit Hüllkörpererzeugung aller drei Testfälle sind ungefähr gleich, nur der Testfall 2 mit der höchsten Anzahl an Kollisionsberechnungen ist minimal höher. Die Berechnungszeiten ohne Hüllkörpererzeugung sind um ca. das Zehnfache kleiner als mit. Dies liegt, wie bereits beschrieben, an dem benötigen Zeitaufwand zur Erzeugung der Hüllkörper.

Grundlage der drei nachfolgenden Tests sind die Testungen Kugel gegen Box. Die einzelnen Szenarien der Testfälle sind entsprechend der Testfälle 1, 2 und 3 aufgebaut. Die Ergebnisse sind in der Tabelle 4.2 enthalten.

Testfall	**Berechnungen**	**Ergebnis**	**Zeit [ms]**	**Zeit [ms] ***
Testfall 4	4	keine Kollision	53	5
Testfall 5	12	keine Kollision	56	5
Testfall 6	8	Kollision	54	6

Tabelle 4.2: Auswertung der Kollisionstests mit Kugeln als Hüllkörper der Node-Volumen und Boxen für die Branch-Volumen (* Berechnungzeit ohne Hüllkörpererzeugung)

Die einzelnen Ergebnisse der Testfälle 4 bis 6 entsprechend im Wesentlichen denen der ersten drei Testfälle. Einzig bei Testfall 5, der dem Szenario des Testfalls 2 entspricht, konnte die Anzahl an Kollisionsberechnungen auf ingesamt 12 verringert werden. Dies ist auf die bessere Hüllkörpereffizienz der Box im Vergleich zur Kugel zurückzuführen. Wie auch bei den Testfällen 1 bis 3 ist die Berechnungszeit ohne Hüllkörpererzeugung ca. um das Zehnfache geringer als mit Erzeugung der Hüllkörper.

Die unterschiedlichen Berechnungszeiten, sowohl bei den Kugel-Kugel-Tests, als auch bei den Tests Box-Box, können mit der Hüllkörpererzeugung erklärt werden. Sind einmal die gesamten Hüllkörper des Baumes erstellt, so werden die Berechnungszeiten deutlich geringer. Ändern sich zur weiteren Verwendung nur noch einzelne Teile des Baumes, müssen nur die jeweiligen Hüllkörper erzeugt werden. Die Berechnungsdauer in späteren Abfragen kann somit gering gehalten werden, da die Hüllkörper bereits vorhanden sind und nicht mehr berechnet werden müssen.

Kapitel 5

Fazit und Ausblick

Es folgt im letzten Abschnitt eine Zusammenfassung der Arbeit und ein Überblick über Stärken und Schwächen der Umsetzung. Im Anschluss werden im Ausblick Möglichkeiten der Erweiterung des erarbeiteten Ansatzes sowie mögliche Überarbeitungsaspekte aufgeführt.

5.1 Zusammenfassung und Fazit

In der vorliegenden Arbeit wurde die wesentliche Grundlagen der Kollisionsberechnung genannt. Dazu wurden wesentliche Grundlagen aufgeführt, die zur Kollisionsberechnung und zur Entwicklung des eigenen Ansatzes verwendet wurden. Die in Kapitel 2 beschriebene Abstandsberechnungen stellen die Berechnungen dar, wie die kürzesten Distanzen zwischen einzelnen Elementen bestimmt werden können. Diese bilden die Grundlage der Kollisionstests zwischen verschiedenen Geometrien und somit die Basis des eigenen Ansatzes.
In Kapitel 3 wurden die Simulationsumgebungen ROS und MIRA vorgestellt. MIRA wird im Forschungsprojekt, in deren Rahmen diese Masterarbeit eingebettet ist, als Framework verwendet. Aus diesem Grund wurde der eigene Ansatz dafür entwickelt. Auch wurden Probleme bei der Einbindung von Physik Engines und bereits vorhandenen Bibliotheken zur Kollisionsberechnung genannt.
Der eigene Ansatz wurde im vierten Kapitel vorgestellt. Dafür wurden die jeweils verwendeten Hüllkörper sowie die Tests zwischen ihnen beschrieben. Diese sind wesentlich für die Kollisionsberechnung der erarbeiteten Umsetzung. Mittels einfacher

Kollisionstests wurde der Ansatz geprüft und bewertet.

Im Verlauf der Arbeit sind bei der Entwicklung des eigenen Ansatzes der Kollisionsberechnungen unterschiedliche Hindernisse aufgetreten, die sich in der Umsetzung widerspiegeln. Der eigene Ansatz weist daher einzelne Schwächen auf. Dazu gehört beispielsweise, dass die Berechnung des Kollisionsabstandes zwischen zwei Boxen in der aktuellen Umsetzung noch nicht fehlerfrei genutzt werden kann. Auch zeigt sich, dass die Berechnung eines Hüllkörperbaumes sehr rechenintensiv ist, wie es in der Bewertung auch dargestellt wurde. Dies führt teilweise zu langen Rechenzeiten gegenüber des eigenen Kollisionstests. Im ungünstigsten Fall führt diese Art der Berechnung zu mehr Berechnungsschritten im Vergleich zur der eigentlich notwendigen Anzahl an Schritten. Des Weiteren hat sich herausgestellt, dass die Abstandsberechnung im Zusammenhang mit der verwendeten Baumstruktur eher ungünstig ist. Grund dafür ist, dass für einen minimalen Abstand zwischen den Objekten der Baum für jeden Node immer bis zur untersten Ebene durchlaufen werden müsste. Somit führen die Abbruchbedingungen bei der Kollisionssuche des aktuellen Ansatzes zu falschen minimalen Distanzen zwischen den einzelnen Objekten.
Dennoch ist festzuhalten, dass das Softwaredesign des eigenen Ansatzes, wie in entsprechenden Kapitel gezeigt, funktioniert und sicher Kollisionen erkennt. Zudem ist es mit diesem Ansatz möglich, Abstände zwischen zwei Objekten zu Berechnen und somit Kollisionen vorherzusagen. Des Weiteren konnte gezeigt werden, dass eine Kollision auch dann bereits erkannt werden konnte, bevor alle Objekte durchgerechnet werden mussten. Der in der Umsetzung entwickelte Suchalgorithmus ermöglicht es demnach, Kollisionsberechnungen zu verkürzen.
Es ist sinnvoll, den erarbeiteten Ansatz im weiteren Verlauf des Forschungsprojektes SONARO zu erweitern, um ihn gewinnbringend für die Kollisionsberechnung und die daran anschließende Kollisionsvermeidung zu einzusetzen.

5.2 Ausblick

Aufgrund der im Laufe der Arbeit erkannten Schwächen des eigenen Ansatzes, erscheint es sinnvoll einige Aspekte weiter zu verfolgen, um die Kollisionsberechnung zu optimieren. Sinnvoll wäre es beispielsweise die Berechnungen zunächst als reine Kollisionstests

und nicht wie in der Arbeit erfolgt, als Abstandstests durchzuführen, wodurch die Berechnung möglicherweise beschleunigt werden könnte. Dennoch sollte ein gegebener Sicherheitsabstand immer in die Berechnungen einfließen, um Kollisionen zu vermeiden und frühzeitig darauf zu reagieren.
Der Roboter, für den die Kollisionsberechnung erarbeitet wurden, besitzt einen mobilen Greifer. Dieser ist dazu gedacht, dass der Roboter Objekte ergreifen, festhalten und übergeben kann. Hierfür muss sichergestellt werden, dass zum Zeitpunkt der Interaktion diese Objekte nicht als mögliche Kollisionsobjekte betrachtet werden. Der erarbeitete Ansatz sollte daher für diesen Fall noch erweitert werden, da zum aktuellen Zeitpunkt solch eine Betrachtung noch nicht eingearbeitet ist. Zusätzlich könnten auch weitere Bibliotheken eine gewinnbringende Erweiterung bieten und durch die Implementierung in den bestehenden Ansatz diesen noch optimieren. Eine weitere Möglichkeit, den Ansatz zu optimieren wäre, dass die Umgebungskarte in die eigentliche Szene mittels eines MapViewers eingearbeitet wird. Somit müssen die Objekte der Map nicht in die Collision-Baumstruktur gebracht werden sondern es kann die eigene Struktur verwendet werden. Um den Ansatz noch zu verbessern, wäre eine zusätzliche Möglichkeit, weitere Hüllkörper einzusetzen, deren Effizienz noch höher ist, als bei den bereits verwendeten. Dies könnte auch dem Problem entgegenwirken, dass in einzelnen Fällen zu viele Berechnungsschritte durchgeführt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[MIR, o.D.a] (o.D.a). *Comparison of MIRA with ROS*. In: https://www.mira-project.org/MIRA-doc/ComparisonWithROSPage.html (aufgerufen am 10.08.2019). *(Seite: 25)*

[MIR, o.D.b] (o.D.b). *MIRA. Introduction and Features*. In: http://www.mira-project.org/MIRA-doc/IntroductionPage.html (aufgerufen am 03.08.2019). *(Seite: 24)*

[Sta, o.D.] (o.D.). *The Players Project. Stage*. In: http://playerstage.sourceforge.net/index.php?src=stage (aufgerufen am 03.08.2019). *(Seite: 24)*

[ROS, o.D.] (o.D.). *ROS. About ROS*. In: https://www.ros.org/about-ros/ (aufgerufen am 03.08.2019). *(Seite: 23)*

[Aichele, 2015] Aichele, Fabian (2015). *Kollisionserkennung für echtzeitfähige Starrkörpersimulationen in der Industrie- und Servicerobotik*. Doktorarbeit, Universität Stuttgart. *(Seite: 17, 18, 19, 20)*

[van den Bergen, 2004] Bergen, Gino van den (2004). *Collision Detection in Interactive 3D Environments*. Interactive 3D Technology. Morgan Kaufmann Publishers. *(Seite: 5, 6)*

[Coumans, 2015] Coumans, Erwin (2015). *Bullet 2.83 Physics SDK Manual*. *(Seite: 25, 26, 59)*

[Ebert, 2003] Ebert, Dirk (2003). *Bildbasierte Erzeugung kollisionsfreier Transferbewegungen für Industrieroboter*. Doktorarbeit, Technische Universität Kaiserslautern. *(Seite: 3)*

[Einhorn und Gross, 2013] Einhorn, Erik und H.-M. Gross (2013). *Generic 2D/3D SLAM with NDT Maps for Lifelong Application*. In: *6th European Conference on Mobile Robots*. *(Seite: 27)*

[Ericson, 2005] Ericson, Christer (2005). *Real-Time Collision Detection*. Interactive 3D Technology. Morgan Kaufmann Publishers. *(Seite: 8, 10, 12, 13, 14, 16, 17, 18, 33, 57)*

[Gerkey et al., 2003] Gerkey, Brian, R. Vaughan und A. Howard (2003). *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.*. In: *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, S. 317–323. *(Seite: 24)*

[GitHub, 2019] GitHub, Inc. (2019). *Flexible Collision Library*. In: https://github.com/flexible-collision-library/fcl (aufgerufen am 13.08.2019). *(Seite: 26)*

[Gottschalk, 2000] Gottschalk, Stefan (2000). *Collision Queries using Oriented Bounding Boxes*. Doktorarbeit, University of North Carolina at Chapel Hill. *(Seite: 16)*

[Gross et al., 2017] Gross, Horst-Michael, A. Scheidig, K. Debes, E. Einhorn, M. Eisenbach, S. Mueller, T. Schmiedel, T. Trinh, C. Weinrich, T. Wengefeld, A. Bley und C. Martin (2017). *ROREAS: robot coach for walking and orientation training inclinical post-stroke rehabilitation - prototype implementationand evaluation in field trials*. Autonomous Robots, 41(3):679–698. *(Seite: 2)*

[Hillienhof, 2016] Hillienhof, Arne (2016). *Frührehabilitation: Reha-Roboter unterstützt Training von Schlaganfallpatienten*. Deutsches Ärzteblatt, 113(21):1053–1054. *(Seite: 2)*

[Klein et al., 2018] Klein, Barbara, B. Graf, I. Schlömer, H. Rossberg, K. Röhricht und S. Baumgarten (2018). *Robotik in der Gesundheitswirtschaft. Einsatzfelder und Potenziale*. medhochzwei. *(Seite: 1, 2)*

[Mueller et al., 2017] Mueller, Steffen, T. Q. Trinh und H.-M. Gross (2017). *Local real-time motion planning using evolutionary optimization.* In: *Towards Autonomous Robotic Systems*, S. 211–221. *(Seite: 28)*

[Onnasch et al., 2016] Onnasch, Linda, X. Meier und T. Jürgensohn (2016). *Mensch-Roboter-Interaktion- Eine Taxonomie für alle Anwendungsfälle.* baua:focus. *(Seite: 2)*

[Petersen, 2007] Petersen, Karen (2007). *Effiziente Kollisionserkennung und echtzeitfähige Simulation der Kinematik, Dynamik und Sensorik autonomer Fahrzeuge.* Diplomarbeit, Technische Universität Darmstadt. *(Seite: 16, 17, 18, 20, 33)*

[Verth und Bishop, 2008] Verth, James M. Van und L. M. Bishop (2008). *Essential Mathematics For Games And Interactive Applications - A Programmer's Guide.* Morgan Kaufmann Publishers, Second Edition Aufl. *(Seite: 6, 7, 57)*