Technische Universität München

Fachbereich Informationstechnik

Praktikumsbericht

im Studiengang Elektro- und Informationstechnik

Thema: Programmieraufgaben bei Siemens Healthcare
Weiterentwicklung am Simulationsprogramm ArtisSim

eingereicht von: Matthias Kreileder, Matriel-Nr. 03617386

eingereicht am: 06. Juni 2014

Betreuer: Herr Prof. Dr. J. Conradt

Inhaltsverzeichnis

1 Motivation und Ziele

Minimierung von Entwicklungsrisiken durch Simulation

In der Angiographie werden Bilder von Blutgefäßen erzeugt. Um von allen Körperpartien Aufnahmen machen zu können, muss das verwendete Robotersystem in der Lage sein sich möglichst beliebig entlang von Trajektorien bewegen zu können.

Damit die Risiken bei der Integration neuer Hardwarekomponenten minimieren werden können, hat das Mechatronik-Team der System-Engineering-Abteilung das Simulationsprogramm „ArtisSim“ entwickelt. Der Name setzt sich zusammen aus „Artis“, der gemeinsamen Produktbezeichnung der aktuellen Siemens-Angiographiegeräte, und „Sim“ für Simulator.

Der Simulator erlaubt es, alle derzeit bei Siemens verfügbaren Angiographiesysteme zu simulieren. Dafür sind die geometrischen Daten der einzelnen Objekte und ihrer Bestandteile erforderlich, die aus den CAD-Daten exportiert werden können. Mittlerweile wird der Simulator abteilungsübergreifend für die Simulation von Angiographiesystemen eingesetzt. [1]

Da die Hauptanwendungsfälle die Kollisionsberechnung und das Erstellen von Trajektorien sind, wurden im Rahmen des Praktikums die beiden folgenden Ziele vereinbart:

- Implementierung eines Rasters im Simulationsfenster zur schnellen Entfernungsabschätzung zwischen Objekten.
- Customizing des Algorithmus zur Erstellung des Kollisionsmodells für eine intuitivere Bedienung von ArtisSim.

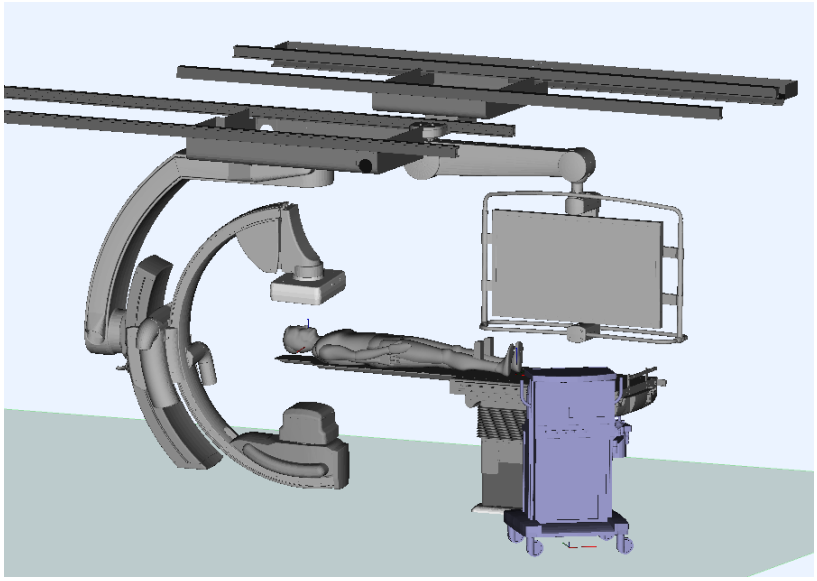


Abbildung 1.1: Simulierte Szene eines Angiographiesystems

2 Technischer Hintergrund und Ausgangslage

2.1 ArtisSim

Die Oberfläche von ArtisSim besteht aus einem Simulations- und einem Bedienfenster. Im Simulationsfenster werden aus den importierten CAD-Daten Geometrien des kompletten Angiographiesystems geladen und realitätsnah dargestellt. Es ist möglich, zusätzliche Personen und Geräte im Simulationsfenster ein- beziehungsweise auszublenden. Die Darstellung aus den Geometriedaten erfolgt unter Benutzung des Visualization Toolkits [5]. Innerhalb des Simulationsfensters beschränkt sich die Interaktion mit dem Benutzer bislang auf die Veränderung der Kameraperspektive.

2.2 Aufgabenstellung Raster

Als visuelle Hilfestellung soll im Simulationsfenster von ArtisSim ein Raster auf Fußboden, Wänden und der Decke gezeichnet werden. Das sogenannte „Grid“ soll eine Vielzahl von Linien zusammenfassen und dynamisch anpassbar sein (Variation der Abstände und Sichtbarkeit). Zunächst soll ein Konzept für die Integration in das bestehende Projekt erarbeitet werden, anschließend folgt die Umsetzung in C++.

2.3 Aufgabenstellung Customizing Kollisionsmodell

Die zweite Aufgabe beschäftigt sich mit der Erweiterung des Algorithmus zur Erstellung des Kollisionsmodells. Bisher können Simulationsobjekte über die Auswahl in einer Liste zum Kollisionsmodell hinzugefügt werden. Um die Zusammenstellung eines Kollisionsmodells benutzerfreundlicher zu gestalten, soll ein Mechanismus implementiert werden, mit dessen Hilfe man Objekte direkt im Simulationsfenster selektieren kann.

3 Konzeption und Realisierung

3.1 Konzept Raster

Vor der Entwurfsphase der Rasterimplementierung war es notwendig, den Aufbau des Teilprojekts von ArtisSim zu analysieren welches für die Erzeugung und Verwaltung der Simulationsobjekte zuständig ist.

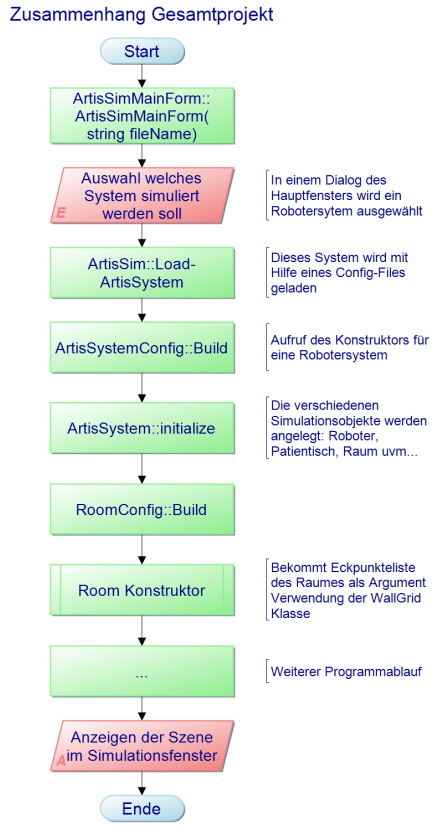


Abbildung 3.1: Zusammenhang Gesamtprojekt

In erster Linie war diese Analyse notwendig um zu untersuchen, welche der bereits existierenden Funktionalitäten wiederverwendet werden können und in welcher Ebene der Vererbungshierarchie die zu implementierende Klasse angesiedelt sein sollte.

Im Hauptmenü des Eingabefensters kann ein zu simulierendes Robotersystem ausgewählt werden. Dieses System wird anschließend mithilfe eines Config-Files geladen. In der Methode `initialize` der `ArtisSystem` Klasse werden die verschiedenen Simulationsobjekte angelegt und von Unterfunktionen danach konkret erzeugt. In der Methode `Build` der `RoomConfig` Klasse wird der `Room` Konstruktor mit den notwendigen Parametern (Eckpunkteliste des Raumes und Raumhöhe) aufgerufen, sowie die Farbgebung und die Bezeichner für den zu simulierenden Raum festgelegt.

Innerhalb des `Room` Konstruktors wird die Eckpunktliste in Unterlisten für Boden, Wände und Decken unterteilt und einer Klasse übergeben, welche daraus unter Benutzung der VTK-Bibliothek die in der Szene erscheinenden Objekte erzeugt. Des Weiteren übernimmt der `Room` Konstruktor die Verwaltung der Erbschaftshierarchie der Simulationsobjekte. Durch die Vergabe von Eltern-Kind Beziehungen werden Manipulationen an Elternobjekten an alle Kindobjekte weitergereicht. Manipulationen umfassen hierbei zum einen die Regulierung des Transparenzgrades von undurchsichtig bis komplett transparent und die Verwaltung der Kindobjekte.

Die Unteraufgabe der Rasterimplementierung ließ sich daher wie folgt aufteilen:

- Das Optionsmenü des Eingabefenster erweitern, um dem User die Möglichkeit zu geben eine Gitterfeinheit einzugeben.
- Diesen Wert nach einem Parsing-Vorgang zur Gültigkeitsüberprüfung mittels Data-Binding an eine Variable innerhalb der Simulation weiterzureichen.
- Die Klasse für das Raster mit den notwendigen Funktionen zur Erbschaftsverwaltung auszustatten.
- Eine Offset-Berechnung zu entwerfen die das Gitter für den Boden durch den Ursprung der Simulationsszene gehen lässt, da der Hauptzweck des Rasters die schnelle Abschätzung von Abständen zwischen Objekten ermöglichen soll.
- Den Algorithmus zur Gittererzeugung generisch zu entwerfen, um die gleiche Anweisungsprozedur für Boden, Decke und Wände verwenden zu können und nicht viele logisch ähnliche Funktionen zu schreiben.

3.2 Implementierung Raster

Der nachfolgende Programmablaufplan veranschaulicht die im Konstruktor implementierte Logik.

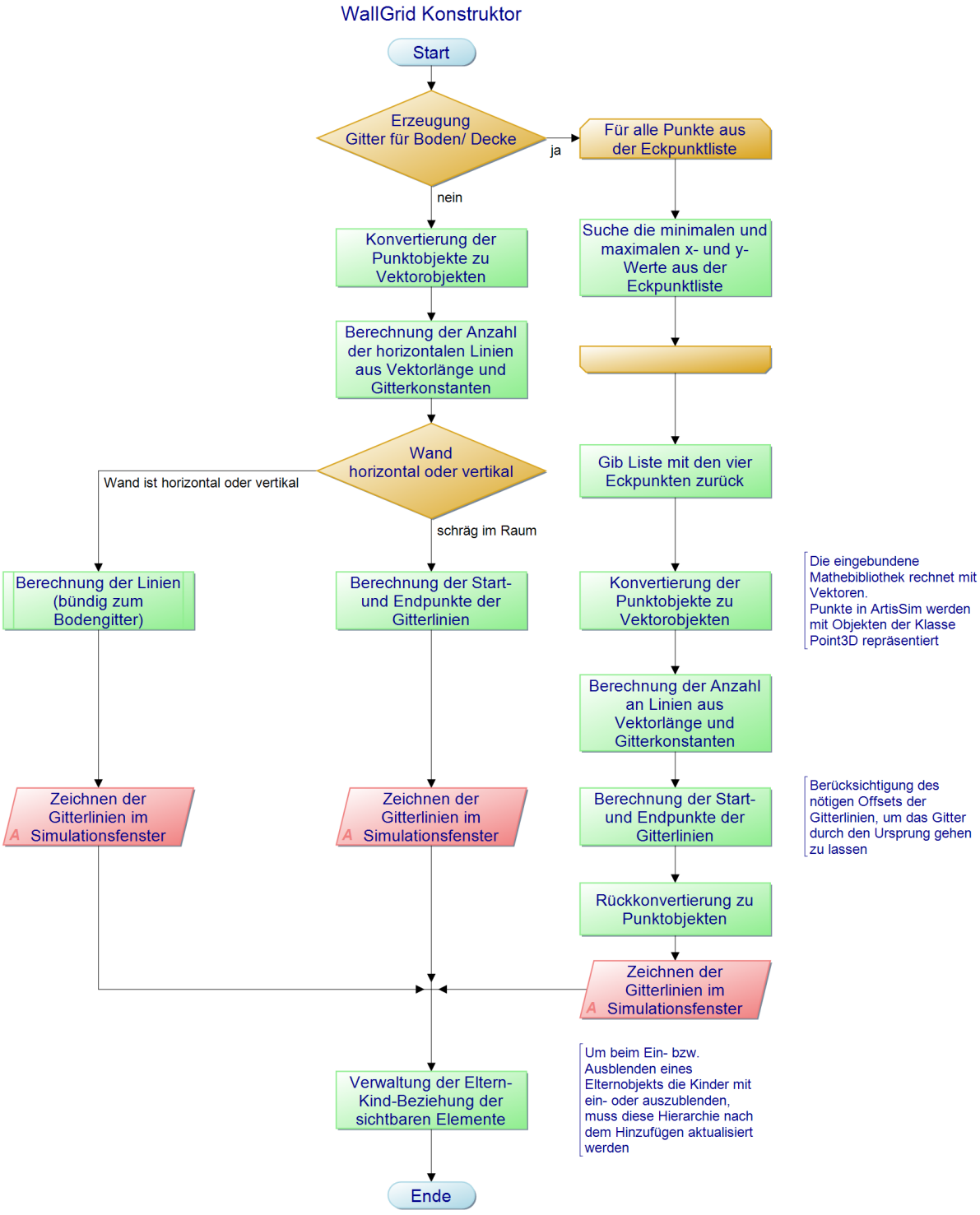


Abbildung 3.2: Zusammenhang Gesamtprojekt

3.3 Konzept Customizing Kollisionsmodell

3.3.1 Command/Observer Design Pattern

Das Konzept für die zweite Aufgabe baut auf dem Command/Observer Design Pattern aus dem VTK-Projekt auf. Hierbei kann eine Klasse die von vtkCommand erbt auf Events reagieren, indem sie eine bestimmte Callback Funktion überschreibt, die dadurch automatisch beim Eintreten des definierten Events aufgerufen wird. [3]

Für die Simulationsanwendung wurde das Drücken der Taste 'P' als Event für einen „Picking-Vorgang" im Simulationsfenster definiert und ein Handler geschrieben, der die Signalweiterleitung zum Bedienfenster anstößt. Dort wird dadurch das ausgewählte Simulationsobjekt in einer Liste markiert. Die markierten Elemente dieser Liste können dann zum Kollisionsmodell hinzugefügt werden.

Dies hat insofern Customizing Charakter, als dass das Anwählen von Roboterteilen im Simulationsfenster intuitiver ist als das Suchen der Bauteilbezeichnung in einer Liste, speziell wenn der Bezeichner für das Bauteil dem User nicht bekannt ist.

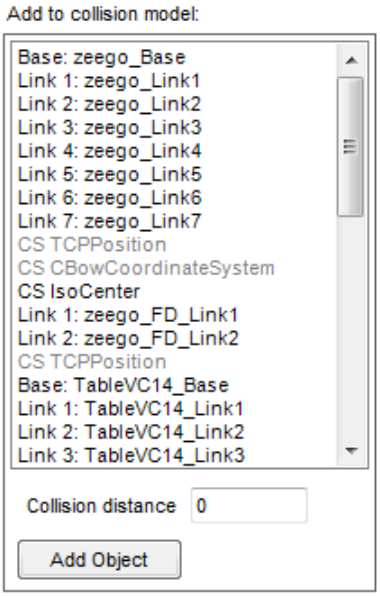


Abbildung 3.3: Liste der Roboterelemente

3.3.2 Mechanismus Signalweiterleitung

Anschließend musste ein Weg gefunden werden um das Event durch die verschiedenen Simulationsschichten bis zur GUI weiterzugeben. Wenn zwischen zwei Objekten von verschiedenen Klassen ein Event weitergeleitet werden soll, gibt es zwei Szenarien:

- Hält das Objekt der Klasse A eine Referenz auf das Objekt der Klasse B, so kann auf sehr einfache Weise das Signal weitergeleitet werden(Methode auf dem Objekt vom Typ B aufrufen).
- Hat das Objekt der Klasse A keine Referenz auf das Objekt der Klasse B, kann entweder mit einem Interface, oder dem in .NET weit verbreiteten Delegate Mechanismus gearbeitet werden.

Während dem Signalfluss zwischen Informationsquelle und Senke kommen hierbei mehrfach beide Szenarien vor. Für den Fall, dass ein Objekt keine Referenz auf sein Elternobjekt hat,

wurde sich für den Delegate Ansatz entschieden, da eine Signalweiterleitung mittels Delegates mehr Flexibilität und weniger Codeänderungen mit sich bringt. [4]

3.4 Implementierung Customizing Kollisionsmodell

3.4.1 Einbinden von nicht verwaltetem Code

Die größte Herausforderung bei der Umsetzung des geplanten Konzepts bestand darin, einen Callback von einem nicht verwalteten zu einem verwalteten Objekt zu implementieren. Der Kern des VTK Projekt ist in C++ geschrieben, ArtisSim hingegen ist ein .NET Projekt. Alle .NET-Sprachen müssen die Common Language Specification erfüllen. Dazu gehört unter anderem die automatische Verwaltung von zur Laufzeit erzeugten Objekten im Heapspeicher. Während Objekte in C++ eine feste Adresse im Speicher besitzen über die sie angesprochen werden können, werden verwaltete Objekte im Zuge der Speicheroptimierung vom Garbage Collector nicht nur automatisch freigegeben, sondern auch im Speicher verschoben um einer Fragmentierung entgegenzuwirken. [2]

Daher ist es nicht möglich für die Kommunikation zwischen verwalteten und nicht verwalteten Objekten den Delegate Mechanismus zu verwenden (da in C++ nicht bekannt) und es kann auch kein Funktionspointer wie im klassischen C++ Coding genutzt werden (verwaltete Objekte haben keine feste Adresse). Hier wurde ein Lösungsvorschlag des Microsoft Developer Networks verwendet: `http://msdn.microsoft.com/de-de/library/367eeye0.aspx`

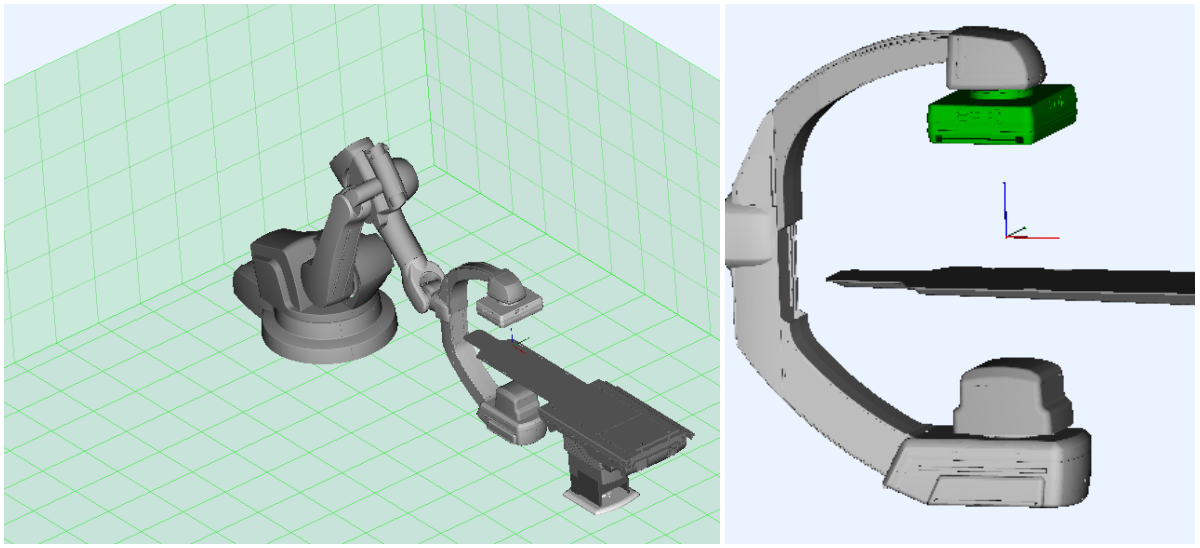
3.4.2 Threadsicheres Aufrufen von Windows Forms-Steuerelementen

Aus Optimierungsgründen existieren in ArtisSim mehrere parallele Ausführungsstränge. Das Multithreading hat Abschnittsweise die Signalweiterreichung erschwert, da der Thread des Simulationsfensters beim Ansprechen von Windows-Forms-Elementen keine Schreibrechte für Objekte besitzt die zum GUI-Ausführungsstrang gehören. Auch hierbei wurde eine Lösung des Microsoft Developer Networks genutzt: `http://msdn.microsoft.com/de-de/library/ms171728%28v=vs.110%29.aspx`

4 Ergebnisse und Evaluation

4.1 Realisierte Grundfunktionalität

Beide Teilaufgaben wurden in ihrer Grundfunktionalität umgesetzt. Das Raster ist in seiner Feinheit und Transparenz verstellbar. Auch wurde darauf geachtet das Gitter durch den Ursprung laufen zu lassen und die Gitterlinien an den Wänden bündig mit dem Boden abschließen zu lassen. Die Selektion von Simulationsobjekten wurde ebenfalls wie angefordert umgesetzt. Durch einen Klick auf ein Objekt wird es in der Simulation grün eingefärbt und im Bedienfenster in der Liste der Roboterteile markiert.



(a) Eingeblendetes Raster an Boden und Wänden

(b) Aus dem Simualtionsfenster selektiertes Roboterelement

Abbildung 4.1: Ansicht der Praktikumsergebnisse im Simulationsfenster

4.2 Ausblick

4.2.1 Gitter für nicht rechtwinklige Räume

Komplizierte Raumgeometrien werden von dem Gitteralgorithmus auf den einfachen, rechtwinkligen Fall zurückgeführt. Es wird ein Rechteck erzeugt, welches das n-Eck des Bodens einrahmt. Auf diesem Rechteck wird anschließend die gleiche Befehlsabfolge aufgerufen wie auf einem rechtwinkligen Raum. Dadurch geht das Gitter über die Grenzen des Bodens hinaus.

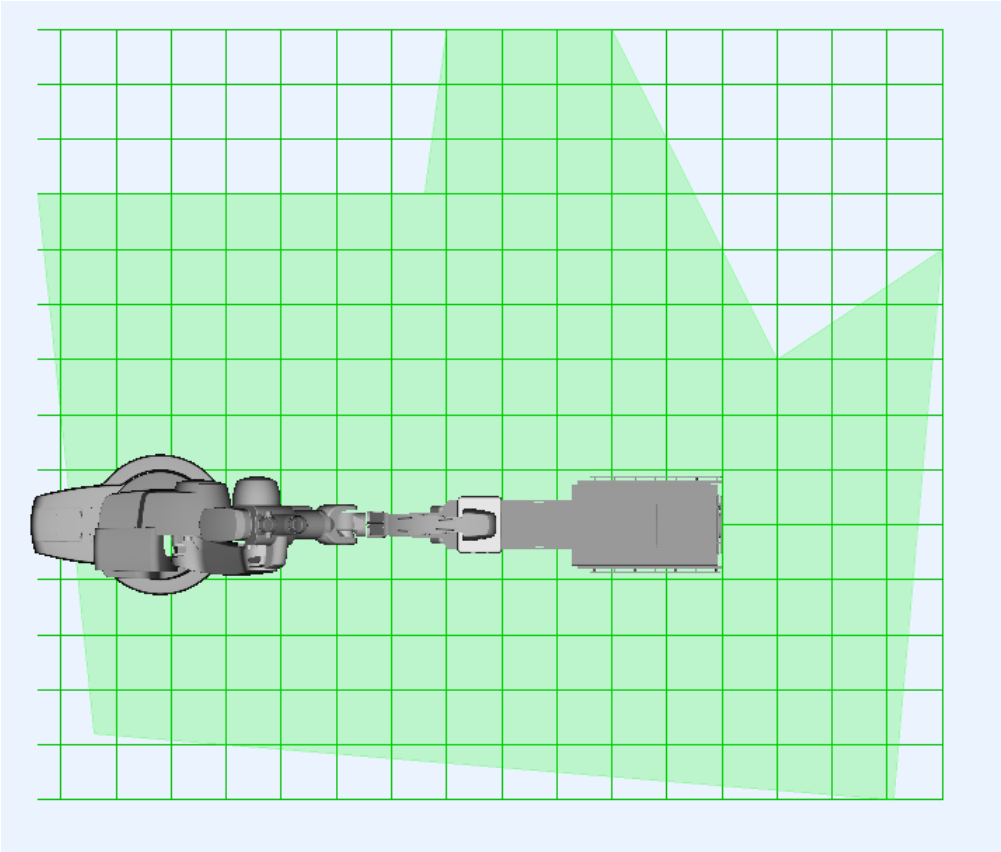


Abbildung 4.2: Der eigentliche Raum liegt nur innerhalb des grün markierten Bereichs

4.2.2 Dynamisches Einstellen der Gitterfeinheit

Des Weiteren ist das Raster nicht dynamisch anpassbar, sondern lediglich vor dem Laden der Simulation im Optionsmenü einstellbar. Einmal gesetzt ist die Gitterfeinheit allerdings fest.

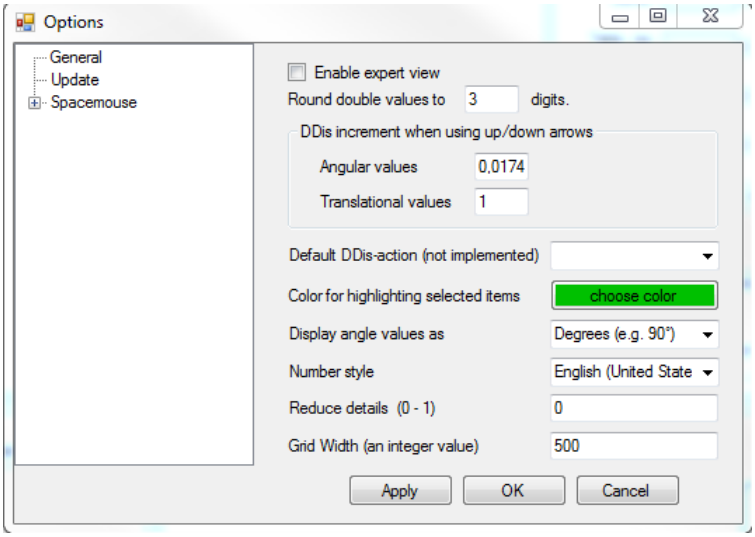


Abbildung 4.3: Das Optionsmenü

4.2.3 Abwählen von Simulationsobjekten

Es wurde versucht das Abwählen von Simulationsobjekten so zu implementieren, dass eine erneute Selektion das entsprechende Objekt wieder abwählt. Das Abwählen wird allerdings erst im Simulationsfenster sichtbar, wenn das Fenster ein Update erfährt (beispielsweise ein Mausklick auf das Fenster). Auch nach längeren Bemühungen diesen Bug zu beseitigen, lies sich dieses Problem nicht lösen.

5 Diskussion

Die Ergebnisse der Praktikumstätigkeit wurden als gut beurteilt. Die implementierten Funktionen tragen zur Beschleunigen der Arbeit mit ArtisSim bei, daher wurden sie von allen Usern sehr begrüßt.

Es wurde auf eine ausreichende Dokumentation geachtet, um eventuelle Weiterentwicklungen durch andere Mitarbeiter möglichst einfach zu gestalten.

Literaturverzeichnis

[1] Bopp, Johannes, Hochschule Mannheim, Diplomarbeit, 2013

[2] Moses, Daniel: *Programmieren unter .net C sharp Edition.* Franzis Verlag, 2002

[3] Schroeder, Will: *Visualization Toolkit.* Kitware Inc., 2006

[4] Templeman, Julian: *MicrosoftVisual C++/CLI Step by Step.* Microsoft Gmbh, 2013

[5] `http://www.vtk.org/`