Hochschule Zittau/Görlitz

Praktikumsbeleg

Untersuchung und Implementierung von Methoden der CAD-gestützten Roboterprogrammierung

Dongliang Cao

Bearbeitungszeitraum	3 Monate
Matrikelnummer	217043
Betreuer der Ausbildungsfirma	Dr.-Ing. habil. Fan Dai
Gutachter der Hochschule	Prof. Dr. Stefan Bischoff

15. Mai 2019

Danksagung

Ich möchte mich beim Prof. Stefan Bischoff bedanken, dass er mein Betreuer an der Hochschule ist, und mich bei dem Praktikum unterstützt.

Ich möchte auch Dr. Dai Fan, meinem Betreuer bei ABB Forschungszentrum danken. In meinem Praktikum hat er mir bei theoretischer Fachkenntnisse und auch bei praktischer Anwendungsbereich viele Unterstützung gegeben. Er hat mir alles klar erklärt mit viel Geduld und Verständnis.

Selbständigkeitserklärung

Ich versichere hiermit, dass ich meine Praxisarbeit mit dem Thema „Untersuchung und Implementierung von Methoden der CAD-gestützten Roboterprogrammierung“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Mannheim, 15. Mai 2019
Ort, Datum

Dongliang Cao
Unterschrift

Inhaltsverzeichnis

1 Einführung

1.1 Problemstellung

Die zunehmende Komplexität von Produkten und Maschinen sowie kurze Produktionszyklen bei kleinen Losgrößen stellen die Industriebranche vor große Herausforderungen. Sowohl die Programmierung von Industrierobotern im Online-Modus mit Handbediengerät als auch im Offline-Modus mit virtueller Simulation erfordert den Ingenieuren spezielle Kenntnisse in der Robotik und in fertigungsabhängigen Robotersteuerungssystemen und ist bei komplizierten Aufgaben sehr zeitaufwändig. Insbesondere klein- und mittelständische Unternehmen konfrontieren mit zusätzlichen Hindernissen wie höhe Investitionen für die Ausbildung der nicht qualifizierten Mitarbeitern und die wegen der Inbetriebnahme von Roboterzeller verlängerten Produktionszyklen.

1.2 Motivation

Um die Ingenieuren von den wiederholenden, eintönigen und zeitaufwändigen Programmierungsverfahren zu befreien und deshalb sich mehr auf das Projekt selbst und dessen notwendigen Funktionen zu konzentrieren, benötigt es neue Methode für die Erzeugung des Roboterprogrammes, die effizienter, intuitiver und benutzerfreundlicher ist.

Grundsätzlich gesagt, im Vergleich mit Online Programmierung kann OLP dank der Simulationsumgebung mehr Zeit und Bemühung für Benutzer sparen. Der Benutzter braucht nicht vor Ort den echten Roboter zu steuern, um die gewünschte Position zu erreichen und selbst zu programmieren, sondern sitzt sich vor den Bildschirm im Büro und gebt die notwendigen Positionen, Konfigurationen und Befehle in Simulationsumgebung. Nach der Simulation, die die Erreichbarkeit der gewünschten Positionen und die Kollisionsmöglichkeit überprüft, generiert das Roboterprogramm automatisch. Aber der Mangel für heutige Offline-Software ist auch eindeutig, wenn der Benutzer eine lange Reihe von Bewegungsanweisungen definieren möchten, braucht er entweder viele Zeit, um jede Position zu definieren oder weißt er nicht genaue die Positionen.

Die meiste Information über die wichtigsten Positionen für die Bewegungsanweisung beinhaltet normalerweise das Objekt, das direkt von dem Roboter bedient wird. Und die Information über das Objekt kann durch die Analyse der dahinter versteckten geometrische Daten nachziehen. Leider ziehen jetzt wenige Ansätze für die Kombination von CAD-Daten und Offline Programmierung.

Aus obigen Gründen möchte ich in meinem Praktikum ein Add-In-Programm

in Robotstudio entwickeln und ermöglicht die automatisch Erzeugung der Position für den Einschubprozess von zwei Objekten.

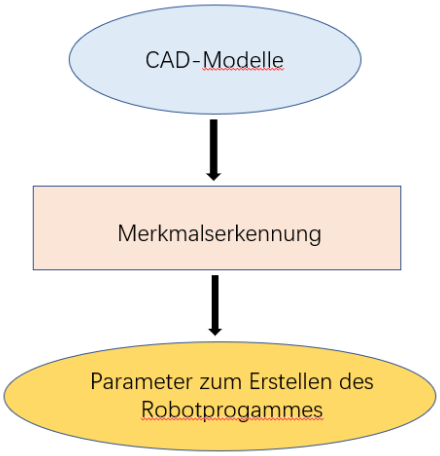

Abbildung 1: Bearbeitungsverfahren für CAD-Daten

1.3 Zielstellung

Extraktion und Analyse der Information von CAD-Daten Abruf und Analyse der geometrischen Informationen und Topologieinformationen des Objekts aus der CAD-Datei

Filtern der Information von CAD-Daten Ausschluss der irrelevanten Informationen und Herausfiltern der Informationen, die Beziehungen zwischen Objekten enthalten

Feststellung der Einschubrichtung und des Einschubabstandes Kalkulation und Bestimmung der potentiellen Einschubrichtungen mithilfe von gefilterten geometrischen Informationen und Topologieinformationen

Visualisierung der Ergebnisse Visualisierung der Ergebnisse in Benutzeroberfläche

2 Theoretische Grundlangen

2.1 Offline Programmierung

2.1.1 Definition

Die Offline-Programmierung (OLP) ist eine Roboterprogrammierungsmethode, bei der das Roboterprogramm unabhängig von der eigentlichen Roboterzelle erstellt wird. Das Roboterprogramm wird dann zur Ausführung auf den realen Industrieroboter hochgeladen. Bei der Offline-Programmierung wird die Roboterzelle durch ein grafisches 3D-Modell in einem Simulator dargestellt. Heutzutage helfen OLP Roboterintegratoren dabei, die optimalen Programm für den Roboter zu erstellen, um eine bestimmte Aufgabe auszuführen. Bei der Simulation des Roboterprogramms können Roboterbewegungen, Erreichbarkeitsanalysen, Kollisions- und Beinahe-Erkennung sowie Zykluszeitberichte berücksichtigt werden.[1]

Vorteile

1) Die Offline-Programmierung bricht die Produktion nicht ab, weil das Programm für den Roboter außerhalb des Produktionsprozesses auf einem externen PC geschrieben wird.

2) Integratoren und Endbenutzer können beim Entwurf einer Arbeitszelle Zeit und Geld sparen im Vergleich mit Online-Programmierung.

3) Die Fähigkeit zu analysieren, wie sich eine Arbeitszelle verhält, bevor Zeit und Geld in Geräte investiert werden, sorgt für eine reibungslose Umsetzung vom Konzept zur Realität.

Vorgehensweise mit Offline-Programmierung

1) **Erstellung der Arbeitszelle**
   Eine Arbeitszelle bzw. Roboterzelle bezieht sich auf eine Kombination von einem oder mehreren Robotern und das damit verbundene Werkzeuge, andere Werkstücke und Vorrichtungen. Wenn man eine Arbeitszelle erstellt, soll man zuerst alle Komponente in der Arbeitszelle importieren und danach platzieren. Außerdem muss der Roboter mit einer virtuellen Steuerung verbinden, um zu programmieren.Ein Werkzeug ist ein spezielles Objekt (z. B. eine Lichtbogenschweißzange oder ein Greifer), das an einem Werkstück arbeitet. Für korrekte Bewegungen in Roboterprogrammen müssen die Parameter des Werkzeugs in den Werkzeugdaten angegeben werden. Der wesentliche Teil der Werkzeugdaten ist der Werkzeugarbeitspunkt (TCP).

2) **Programmierung von Robotern**
Bevor der Benutzer ein Programm für seinen Roboter erstellen, solltet er die vorher genannte Arbeitszelle einrichten, in der sein Roboter arbeiten soll, einschließlich der Roboter, Werkzeug und Vorrichtungen.

das Verfahren für die Programmierung von Robotern kann sich hauptsächlich in 5 Schritte unterteilen.

- Erstellen von Positionen und Bahnen
- Prüfung der Positionsorientierung und Erreichbarkeit
- Synchronisieren des Programms mit der virtuellen Steuerung
- Ausführen von textbasierter Bearbeitung
- Kollisionserkennung

3) **Simulieren von Programmen**
Mit Simulationen werden vollständige Roboterprogramme auf einer virtuellen Steuerung ausgeführt.Durch Simulation kann die Zykluszeit berechnet,die Kollision erkannt, die E/A-Signale simuliert und auch die Ereignisse (Aktion mit einem Trigger verbunden) behandelt werden, um festzustellen, ob das Robotersystem die Erwartung von Endbenutzer erfüllt.

4) **Aufladung des Programmes in realer Steuerung**
Wenn die Simulation erfolgreich durchgeführt wird, kann der Benutzer das automatisch generiert Programm von Computer in realen Roboter aufladen.

2.1.2 Robotstudio

Robotstudio ist ein typisches Anwendungsbeispiel für die Offline-Programmierung (OLP), das von ABB entwickelt und unterstützt wird. RobotStudio ermöglicht den Benutzer das Arbeiten mit einer Offline-Steuerung. Dabei handelt es sich um eine virtuelle IRC5-Steuerung, die lokal auf dem PC ausgeführt wird. RobotStudio basiert auf dem so genannten Virtual Controller, einer exakten Kopie der Originalsoftware, die den Roboter in Produktionsprozessen steuert. So sind realistische Simulationen möglich, denn zum Einsatz kommen die Daten und Konfigurationen, die auch in der realen Produktion zum Einsatz kommen.[3] In meiner Arbeit, wird Robotstudio als eine Benutzeroberfläche zur Interaktion und Visualisierung von generierten Ergebnissen bedient. Außerdem wird Robotstudio Software Development Kit (SDK) als Programmbibliothek unter .NET Framework für die Entwicklung des Add-In-Programmes verwendet.

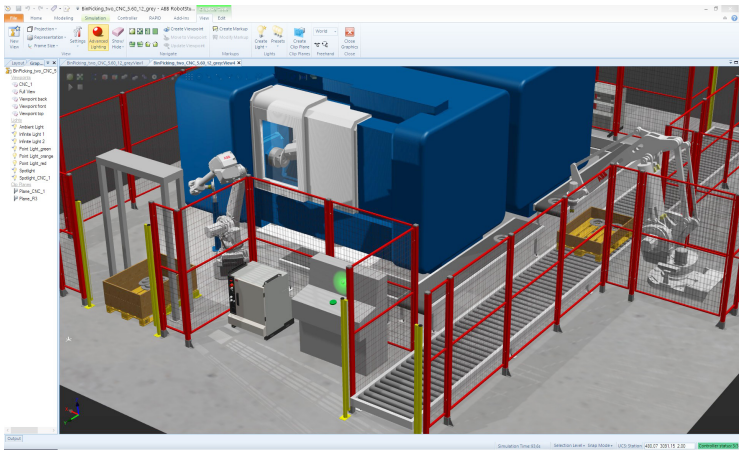

Abbildung 2: Roboterzelle in Robotstudio

2.2 Geometrie und Topologie

Geometrie und Topologie sind zwei wichtigste Begriffe für die mathematische Darstellung der räumlichen Information von Objekte in der realen Welt. Die Geometrie bezieht sich in diesem Artikel besonders auf die euklidische Geometrie, die sich mit Punkten, Geraden, Ebenen, Abständen, Winkeln usw. beschäftigt, sowie diejenigen Begriffsbildungen und Methoden, die im Zuge einer systematischen und mathematischen Behandlung dieses Themas entwickelt wurden.[2]Neben der mathematischen Beschreibung der räumlichen Lage und Form von einzelnen Komponenten, beschreibt die Topologie die Lagebeziehung zwischen Geoobjekten, wie z.B. die Knoten, Kante und Masche. In einfachen Systemen entsprechen den Punkten die Knoten, den Linien die Kanten und den Flächen die Maschen.[4]

Im meisten CAD-Datei wird boundary representation (b-rep), auf deutsch Begrenzungsflächenmodell, als Modellierungsform eines Flächen- oder Volumenmodells verwendet. Ein typisches Beispiel: Eine Fläche ist ein begrenzter Teil einer Oberfläche. Eine Kante ist ein begrenzter Teil einer Kurve und ein Scheitelpunkt liegt an einem Punkt. In der Welt des Datenaustauschs definiert the Standard for the Exchange of Product Model data (STEP) auch einige Datenmodelle für b-rep. Die allgemeinen generischen topologischen und geometrischen Modelle sind in der geometrischen und topologischen Darstellung nach ISO 10303-42 definiert.[5]

2.2.1 Geometrie

Die Geometrie, die für die Modellierung von Geoobjekte in CAD-System, handelt sich um euklidische Geometrie. Die Geometrie kann sich in vier Kategorien einteilen.

1) **Punkten**, die in einem dreidimensionalen Raum existieren.
Der Punkt ist das grundlegendste geometrische Konzept in einem Raum. Ein Punkt im dreidimensionalen Raum wird normalerweise in einem kartesischen Koordinatensystem als eine Position dargestellt und die Position enthält drei Werte, die sich auf x, y, z-Koordinaten beziehen.

Hinweis:
Der Punkt in einem dreidimensionalen Raum kann nicht nur eine Position, sondern auch einen dreidimensionalen Vektor repräsentieren. Ein Vektor ist als eine räumliche Verschiebung von einer Position zu einer anderen Position zu definieren.

2) **Kurven**, die in einem dreidimensionalen Raum existieren.
In der Mathematik ist eine Kurve ein eindimensionales Objekt. Eindimensional bedeutet dabei informell, dass man sich auf der Kurve nur in einer Richtung (bzw. der Gegenrichtung) bewegen kann. Ob die Kurve in der zweidimensionalen Ebene liegt („ebene Kurve“) oder in einem höherdimensionalen Raum.[6]

In CAD-System kann sich die Kurve wesentlich aus zwei Gruppe einteilen: analytische Kurve und interpolierte Kurve. Alle analytischen Kurven werden von einem Parameter, der normalerweise als t genannt wird, parametrisch dargestellt. Die drei wichtigsten Typen in analytische Kurven sind Geraden, Ellipsen (einschließlich Kreise) und Helices.

- *Geraden*
  Eine Gerade wird durch einen Punkt $\vec{p_0}$ und eine Richtung $\vec{r}$ dargestellt. Eine Gerade kann eine unendliche gerade Linie, eine teilweise begrenzte gerade Linie (ein Strahl) oder eine begrenzte gerade Linie (ein Liniensegment) darstellen

  Im dreidimensionalen Raum, die Position eines Punktes auf einer Gerade kann durch die parametrisierte Gleichung berechnet werden.

$$p(\vec{t}) = \vec{p_0} + t\vec{r}$$

- *Ellipsen*
  Eine Ellipse wird durch einen Mittelpunkt $\vec{p_0}$, einen Einheitsvektor $\vec{n}$, der senkrecht zur Ebene der Ellipse ist, einen Vektor $\vec{M}$, der die Hauptachse der Ellipse (einschließlich der Größe der Hauptachse) repräsentiert, und das Radiusverhältnis α (das Verhältnis der Nebenachsenlänge zur Hauptachsenlänge) vollständig definiert.

Im dreidimensionalen Raum, die Position eines Punktes auf einer Ellipse kann durch die parametrisierte Gleichung berechnet werden.

$$p(\vec{t}) = \vec{p_0} + \vec{M}\cos t + \vec{m}\sin t$$
$$\vec{m} = \alpha(\vec{n} \times \vec{M})$$

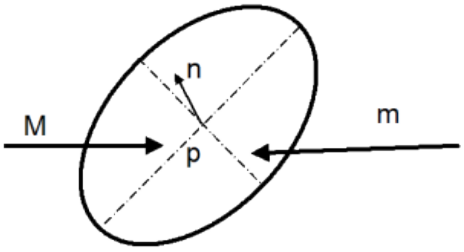

Abbildung 3: Ellipse

– *Helices*
Eine Helix ist typischerweise eine dreidimensionale Spule, die wie ein Schraubengewinde auf der Oberfläche eines Zylinders liegt. Eine Helix wird definiert durch einen Wurzelpunkt, einen Einheitsvektor, der die Achse der Helix definiert, einen Vektor vom Wurzelpunkt zu einem Punkt auf der Kurve, die Steigung, die Drehrichtung und den Parameterbereich.

Im dreidimensionalen Raum, die Position eines Punktes auf einer Helix kann durch die parametrisierte Gleichung berechnet werden.

$$p(\vec{t}) = a\cos t\vec{x_0} + a\sin t\vec{y_0} + bt\vec{z_0}$$

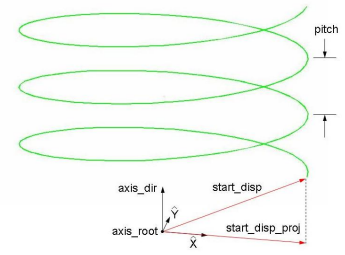

Abbildung 4: Helix

3) **Oberflächen**, die in einem dreidimensionalen Raum existieren.

Der Begriff Oberfläche hat mehrere Bedeutungen. In CAD-System wird dieser Begriff am häufigsten in seinem geometrischen oder mathematischen Sinne verwendet, um ein zweidimensionales Objekt in einem dreidimensionalen Raum mit einer einzigen geometrischen Definition zu beschreiben.

Änlich wie Kurven, in CAD-System kann sich die Oberfläche wesentlich aus zwei Gruppe einteilen: analytische Oberfläche und interpolierte Oberfläche. Alle analytischen Oberflächen werden von zwei Parametern, die normalerweise als u, v genannt werden, parametrisch dargestellt. Die vier wichtigsten Typen in analytische Oberflächen sind Ebenen, Kegel, Kugel und Torus.

- *Ebenen*
  In der Geometrie repräsentiert eine Ebene eine unendliche ebene Fläche oder einen begrenzten Bereich auf einer solchen Fläche. Der Parameter einer Ebene wird durch einen Punkt und einen Normalenvektor definiert.Die Parametrisierung einer Ebene wird durch zwei zusätzliche Parameter definiert: einen Vektor senkrecht zur Normalen, der die U- Parameterrichtung und Skalierung sowie ein Flag, das angibt, ob die Parametrisierung der Ebene für Rechts- oder Linkshänder erfolgt.

  Normalerweise, eine Ebene wird in Bezug auf ein rechtshändiges Koordinatensystem definiert $(\vec{U}, \vec{V}, \vec{N})$. Die Richtung von $\vec{U}$ ist von der entsprechenden Vector bestimmt und die Richtung von $\vec{V}$ ist von die Richtung von $(\vec{N} \times \vec{U})$ bestimmt.

$$p(\vec{u}, v) = \vec{R_0} + u\vec{U} + v\vec{V}$$

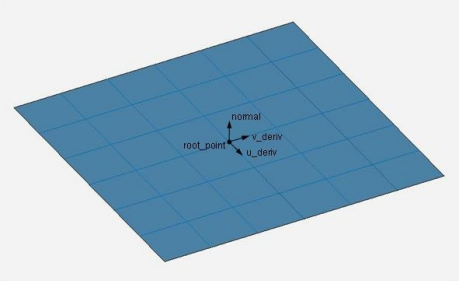

Abbildung 5: Ebene

- *Kegel*
  In der Geometrie, repräsentiert ein Kegel entweder einen Kegel oder einen Zylinder. Die Geometrie eines Kegels wird durch eine

Basisellipse und den Sinus und Cosinus des Haupthalbwinkels des Kegels definiert. Die Normale der Basisellipse repräsentiert die Achse des Kegels.

$$x = au\cos v$$
$$y = au\sin v$$
$$z = u$$

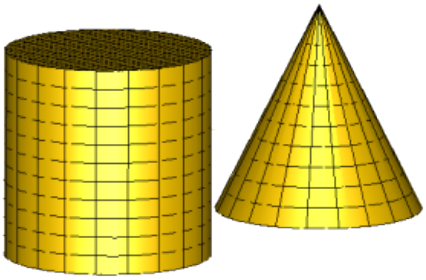

Abbildung 6: Cone

2.2.2 Topologie

Das Grundkonzept von boundary representation (b-rep) ist die Topologie, die beschreibt, wie Elemente begrenzt und verbunden werden. Topologie beschreibt die Beziehungen zwischen unterschiedlichen Geoobjekten.

Zum Beispiel, die Topologie kann sagen, dass eine Kante E_1 durch die Eckpunkte V_1 und V_2 begrenzt ist. Falls wir auch wissen, dass eine andere Kante E_2, durch die Eckpunkte V_2 und V_3 begrenzt ist, dann können wir davon ableiten, dass die Kanten E_1 und E_2 benachbart sind, weil V_2 beide Kanten begrenzt.

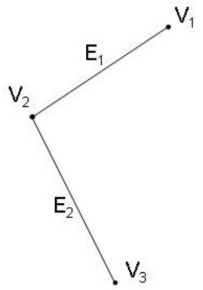

Abbildung 7: Ein Beispiel für die Bedeutung von Topologie

Eine typische Hierarchie für die topologische Elementen mit Volumen besteht einerseits aus Body, Lump, Shells, Subshells, Faces, Loops, Coedges, Edges, Vertices, andererseits für die Elemente ohne Volumen existiert keine Faces, sondern Wires.

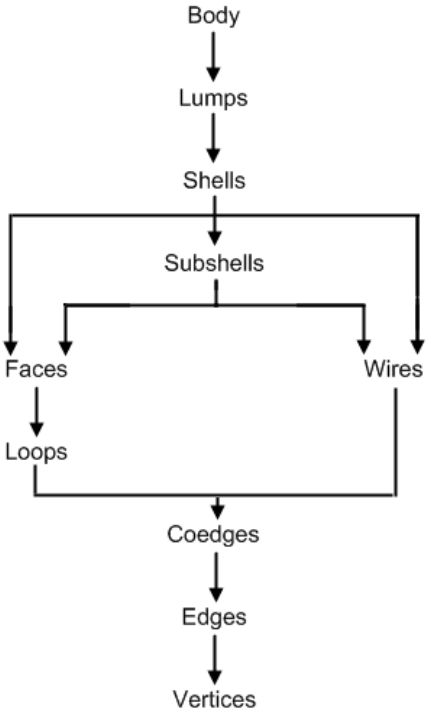

Abbildung 8: Hierarchische Beziehungen zwischen den topologischen Elementen

2.3 CAD Modellierung

Verschiedene CAD-Software unterstützen unterschiedliche CAD-Dateien. In SolidWorks haben sie beispielsweise proprietäre Formate wie ein Teil, der als ".prt"definiert ist, und eine Baugruppe, die als ".asm"gespeichert ist. Die verschiedenen proprietären CAD-Dateiformate werden von unterschiedlichen CAD-Konstruktionssoftware wie Pro Engineer, SolidWorks und AutoCAD verwendet. Wegen der Vielfalt von CAD-Dateiformat, ein neutrales CAD-Dateiformat, mit dem die Daten zwischen verschiedenen CAD-Programmen ausgetauscht werden können, zu finden, ist für die Einarbeitung der CAD-Datei sehr wichtig.

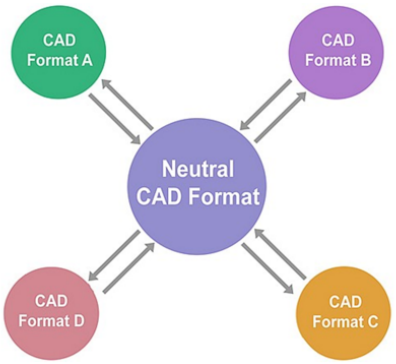

Abbildung 9: neutrales CAD-Dateiformat hilft Datenaustausch

Derzeit eines der beliebtesten CAD-Dateiformat ist STEP.Die Vorteile von STEP spiegelt sich hauptsächlich in den folgenden vier Punkten wieder.

1) Ermöglicht das Anzeigen und Ändern von Geometrie mithilfe eines beliebigen CAD-Tools, das STEP-Geometrie interpretieren kann, und unterbricht die Abhängigkeit zwischen CAD-Systemen und Produktdefinition

2) Enthält das Assembly-Schema, das alle Connector-Entitäten enthält

3) Nützlich zum Gruppieren von mechanischen Elementen in Bestimmte Ansicht

4) Einfache Ableitung der impliziten Informationsinhalte.

3 Aufgabenstellung

3.1 Addin-Entwicklung in RobotStudio

Die Entwicklung für die Benutzeroberfläche zur Interaktion und Visualisierung wird mit dem RobotStudio SDK ausgeführt. Hierbei handelt es sich um ein Framework, mit dem die ABB RobotStudio Funktionen über die verfügbaren APIs bearbeitet werden. Die Programmiersprache C# wird zum Entwickeln der Front-End-Operationen verwendet. Das SDK bietet Visual Studio-Projektvorlagen und APIs zum Erweitern von RobotStudio und zum Erstellen von SmartComponents mit Code-Behind.Download

kurze Vorstellung für RobotStudio SDK und arbeitsrelevante Komponenten
RobotStudio SDK wird unter das .NET Framework entwickelt und von C# Programmiersprache geschrieben. Grundsätzlich kann RobotStudio SDK nach unterschiedlichen Namespaces in verschiedenen Teilen geteilt werden. Am häufigsten benutzte Namespaces sind:

- **ABB.Robotics.Math:** Der Namespace wie den Namen enthält alle Struktur und Klassen für mathematische Operationen in RobotStudio. Einige typische und häufig verwandte Beispiele sind Matrix4 zur Transformation (einschließlich Rotation und Translation), Vector3 zur Beschreibung einer Position, einer Vektor oder Eulersche Winkel und BoundingBox zur Beschreibung der minimale oder kleinste Begrenzungs- oder Umschließungsrahmen für eine Geometrie.

- **ABB.Robotics.RobotStudio:** Der Namespace enthält Klassen, die die grundlegenden Bausteine der RobotStudio Pojektmodellen darstellen, z.B. das Speichern und Laden von Projekten, benutzerdefinierte Attribute, das Rückgängigmachen und verschiedene Maßeinheiten. Viele der Klassen in diesem Namespace dienen als Basisklassen für die spezifischen Klassen im ABB.Robotics.RobotStudio.Stations-Namespace. Ein Beispiel davon ist ProjectObject, das als abstrakte Basisklasse für alle Objekte, die Teil eines Projekts sein können, bedient.

- **ABB.Robotics.RobotStudio.Environment:** Der Namespace enthält Klassen für die Beschäftigung mit der RobotStudio-Benutzeroberfläche, z. B. die Buttons, Windows und Kontextmenüs.

- **ABB.Robotics.RobotStudio.Stations:** Der Namespace enthält Klassen, die das RobotStudio-Objektmodell, das Projektmodell, die Simulationslogik und den Informationsstrom darstellen. Die relevantesten Objektmodelle für meine Arbeit handelt es sich um die geometrische Topologie Klassen.

Die Hierarchie für die Topologie kann man durch folgende Abbildung klar verstehen. die Klassen, die von der Basisklasse „GraphicComponent“ abgeleitet sind, kann direkt in GUI hinzugefügt und gezeigt werden. Die anderen müssen eine Unterkomponente davon sein.

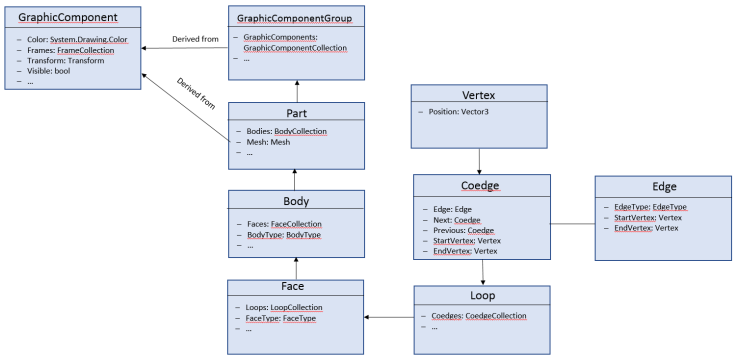

Abbildung 10: geometrische Topologie in RobotStudio

- **ABB.Robotics.RobotStudio.Stations.Forms:** Der Namespace enthält Klassen für die Interaktion mit der 3D-Ansicht sowie eine Reihe von RobotStudio-spezifischen Kontrolle für Formulare und ToolWindows, z. B. Kontrolle für die Eingabe von Koordinaten und die Darstellung von Maßeinheiten.

Die Vorgehensweise für die Entwicklung von UI erläutert wie folgendes:

1) **Herunterladen und Installation von RobotStudio SDK**
   RobotStudio SDK kann man in die Website ABB Developer Center herunterladen. Bevor man RobotStudio SDK installiert, muss man beachten, dass zuvor Visual Studio 2015/2017 schon am Computer installiert ist.

2) **Erstellung eines neuen Add-in Projektes in Visual Studio**
   Öffnen Sie zuerst Microsoft Visual Studio und wechseln Sie zu einem neuen Projekt. Aufgrund der Installation von RobotStudio SDK werden in der folgenden Abbildung die folgenden Vorlagen angezeigt. Wählen Sie aus der Liste RobotStudio 6.0 Empty Add-In aus und wählen Sie einen Namen für das Projekt.
   Aus dieser Vorlage wird die folgende Lösung mit der * .cs-Datei und den grundlegenden RobotStudio SDK * .dlls generiert, die als Referenz zu dieser Lösung verwendet wurden. Jetzt wird mit der * .sln die Umgebung für die Entwicklung des RobotStudio-Add-Ins festgelegt.

   Führen Sie den Build-Befehl in Visual Studio aus, um den Code zu kompilieren und eine .rsaddin-Datei für das Add-In zu generieren. Diese .rsaddin-Datei

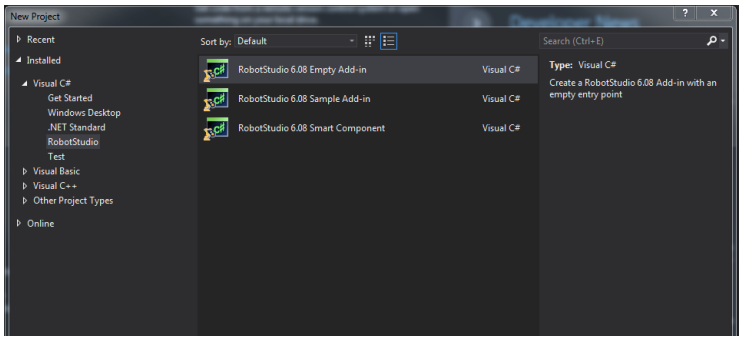

Abbildung 11: neues Projekt für RobotStudio Add-in auswählen

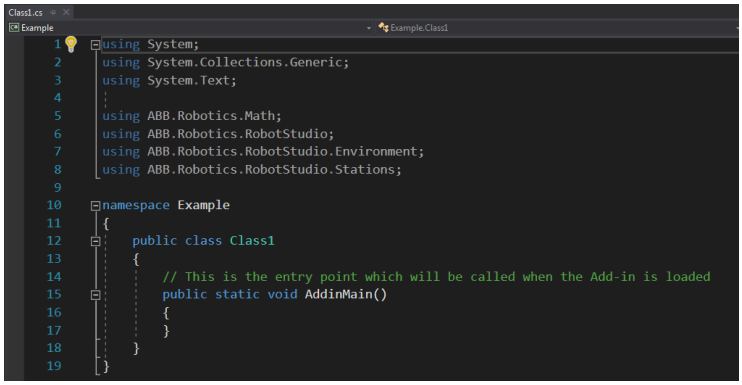

Abbildung 12: leere Lösung für RobotStudio Add-in

ist dafür verantwortlich, dass RobotStudio die Add-In-Assembly (die .dll-Datei) lädt.

3) **UI-Entwicklung**
Im RobotStudio-Arbeitsbereich gibt es verschiedene Arten von Bereichen, z.B. Ribbonbereich, Eigenschaftsbereich und Arbeitsbereich.

Um sich besser an die Gewohnheit des Benutzers anzupassen, verwende ich den Ribbonbereich und füge ich ein neues RibbonButton hinzu. Die Reihenfolge für die Hinzufügung des Ribbons ist grundsätzlich wie folgendes:

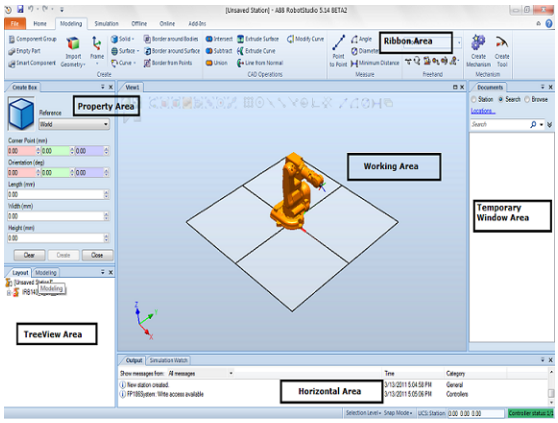

Abbildung 13: Arbeitsbereich in RobotStudio

```
//create a new tab
RibbonTab ribbonTab = new RibbonTab("myTab", "myTab");
UIEnvironment.RibbonTabs.Add(ribbonTab);
//create a new ribbon group
RibbonGroup ribbonGroup = new RibbonGroup("myGroup", "myGroup");
ribbonTab.Groups.Add(ribbonGroup);
//create a new button
CommandBarButton button = new CommandBarButton("myButton", "myButton");
ribbonGroup.Controls.Add(button);
```

Abbildung 14: Der Kode für die Erstellung von einem Button

Erstens, ein neues RibbonTab erstellen und in UIEnvironment hinzufügen.
Zweitens, ein neues RibbonGroup erstellen und in das vorher erstellte RibbonTab hinzufügen.
Drittens, ein neues RibbonButton erstellen und in das vorher erstellte RibbonGroup hinzufügen.

Um die Organisation der Kodestruktur zu verbessern und später besser zu verwalten, habe ich für jeden individuellen Button eine neue Klasse erstellt, sodass die Funktionen und die Daten in dieselbe Klasse eingekapselt sind.

ToolWindow ist eine andere wichtige Klasse für die graphische Benutzeroberfläche, die multifunktional als ein einziger Button ist. Ein ToolWindow ist ein Container für andere graphische Kontrolle, somit ist ein ToolWindow vielseitig und leicht erweiterbar. Eine typische Anwendung für die Nutzung von ToolWindow ist wie folgende Abbildung gezeigt.

```
class ButtonAddInsert
{
    private ViewInsertDefinition view_InsertDefinition = new ViewInsertDefinition();
    public ButtonAddInsert(RibbonGroup ribbonGroup, RibbonTab ribbonTab)
    {
        CommandBarButton btn_new_action = new CommandBarButton("New_Insert", "Insert");
        btn_new_action.HelpText = "Create a new insert action into the assembly tree";
        btn_new_action.Image = Image.FromFile(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "Ri
        btn_new_action.DefaultEnabled = true;

        ribbonGroup.Controls.Add(btn_new_action);

        //Include Seperator between buttons
        CommandBarSeparator seperator = new CommandBarSeparator();
        ribbonGroup.Controls.Add(seperator);

        ribbonGroup.SetControlLayout(btn_new_action, RibbonControlLayout.Large);

        // Add an event handler.
        //btn_new_action.UpdateCommandUI += new UpdateCommandUIEventHandler(Button_UpdateCommandUI_New_Action);
        // Add an event handler for pressing the button.
        btn_new_action.ExecuteCommand += new ExecuteCommandEventHandler(Button_ExecuteCommand_New_Action);
    }
```

Abbildung 15: Ein Beispiel für eine Buttonklasse

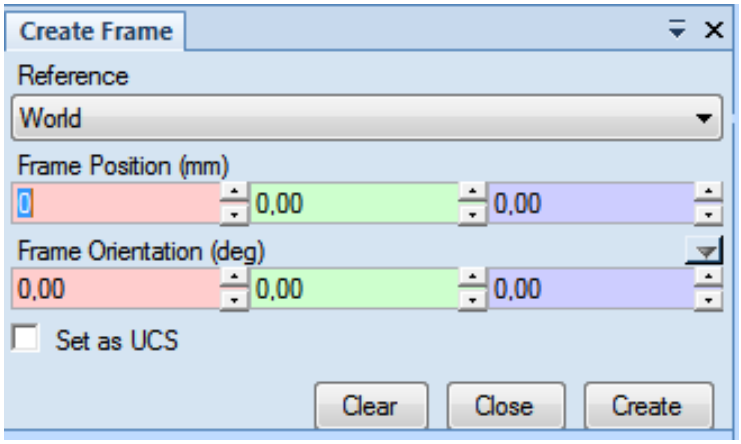

Abbildung 16: Ein Beispiel für die Anwendung von ToolWindow

Das Beispiel dient zur Erzeugung eines neuen Frames, der als eine relevante Position für anderen Komponenten, wie Roboter, Werkzeug und Bewegungsanweisung usw. bedient. In diesem ToolWindow kann man unterschiedliche graphische Kontrolle sehen, wie ComboBox, Label, NumericTextBox, Button, die als ein Einheit zusammenarbeiten, um die Position und die Orientierung in Bezug auf unterschiedlichen Referenzframen zu bestimmen und endlich zu erzeugen.

In meinem kleinen Add-In-Programm, wählt der Benutzer einfach zwei Körper aus eine Geometrie Komponente und danach bestätigt, deshalb sieht das ToolWindow wie folgende Abbildung aus.

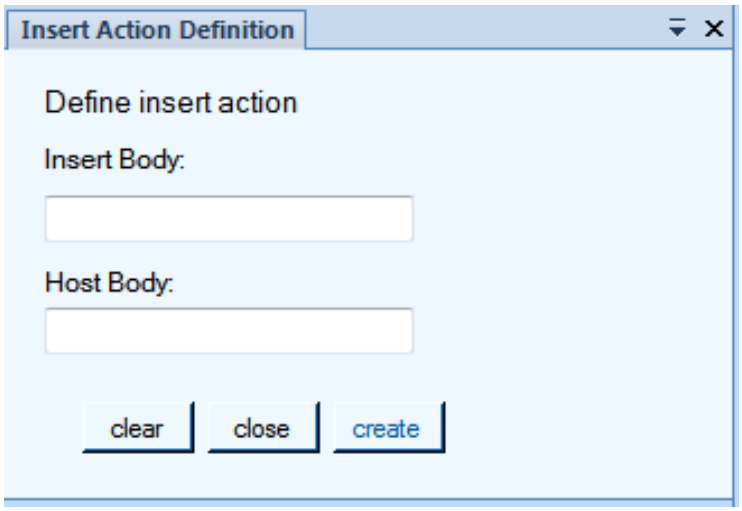

Abbildung 17: Tool Window für die Auswahl von zewi Körpern

Für die Auswahl von den gewünschten Körpern kann der Benutzer entweder in Bereich von Baumansicht oder in Arbeitsbereich durch Mausklick ermöglichen. Um diese Funktion zu verwirklichen, muss man ein neues Ereignis „ObjetAdded“ unter dem Objekt „ProjectSelection“, das als eine Eigenschaft unter der Klasse „Selection“ ist, registrieren. Nach dem Kicken von „Create Button“ werde die Analyse von zwei ausgewählten Körpern beginnen und endlich die kalkulierte Einschubrichtung und -abstand zwischen zwei Körpern in Arbeitsbereich zeigen.

```
//add select body event
Selection.SelectedObjects.ObjectAdded += SelectedObjects_ObjectAdded;
```

Abbildung 18: Registrierung eines neuen Ereignisses

```
private void SelectedObjects_ObjectAdded(object sender, SelectionEventArgs e)
{

    if (!this.toolWindow.Visible || !this.toolWindow.DefaultTabVisibility) return;
    if(e.SelectionObject is Body b)
    {
        if (this.selectInsert)
        {
            this.InsertBodyTB.Tag = b;
            this.InsertBodyTB.Text = b.Parent.DisplayName + "-" + b.DisplayName;
            this.selectInsert = false;
        }
        else
        {
            this.HostBodyTB.Tag = b;
            this.HostBodyTB.Text = b.Parent.DisplayName + "-" + b.DisplayName;
            this.selectInsert = true;
        }
    }

}
```

Abbildung 19: Inhalt des Ereignises

3.2 CAD-Modell Analyse Mithilfe von ACIS Modeler

3.2.1 Kurze Vorstellung von ACIS Modeler

Der 3D ACIS Modeler (ACIS) ist eine objektorientierte dreidimensionale (3D) Geometrie- / Volumenmodellierungs-Engine von Spatial Corp. Sie dient als Grundlage für die Geometrie in praktisch jeder 3D-Modellierungsanwendung für Endbenutzer. ACIS wurde in C ++ geschrieben und bietet ein offenes Architektur-Framework für die Modellierung von Drahtmodellen, Oberflächen und Volumenkörpern aus einer gemeinsamen, einheitlichen Datenstruktur.

3.2.2 Vorstellung der angewandten Funktionen

Für meine Analyse werde ich hauptsächlich die Modellierungskomponente und die Komponente für die Analyse von Beziehung der Objekte benutzt.

- **Modellierungskomponent:** Die Modellierungskomponente lässt sich wesentlich als zwei Gruppe einteilen.

  Einerseits, die erste Gruppe beinhaltet alle Klassen und Strukturen, die für die Beschreibung und Modellierung der Geometrie und Topologie verantwortlich sind. Für die Topologie, die hierarchische Struktur von ACIS ist ähnlich wie die in obergenannte Struktur von RobotStudio nur ohne die oberen Klassen wie Part und GraphicComponentGroup, die als eine Visualisierungseinheit bedienen. Aber für die Geometrie, bietet ACIS andere geometrische Klassen an. Die Trennung von Geometrie und Topologie ermöglicht effizientere Algorithmen und bessere Modifikation von Geometrie. Jedes geometrische Objekt versteckt in ein topologisches Objekt, d.h. kann der Programme durch einen Pointer im topologischen Objekt das dahinter versteckende geometrische Objekt ergreifen. Zum Beispiel, das FACE ist ein topologisches Objekt in ACIS, das eine begrenzte Oberfläche repräsentiert, und dahinter verborgenes geometrisches Objekt SURFACE wird durch einen Pointer in FACE zugewiesen.

  Andererseits, die zweite Gruppe beinhaltet viele nützliche application programm interface (API), die hilfreich für die Traverse durch ganze topologische Hierarchie sind. Die APIs vereinfacht die Arbeit von Programmen zu einem großen Teil. Zum Beispiel, wenn der Programme eine Operation für alle Ecken unter dem gleichen Körper machen möchte, muss er zuerst alle Ecken unter dem Körper bekommen, was sehr mühsam ist, weil der Programme zuerst alle Oberflächen, danach alle Schleife, endlich alle Kanten durchqueren muss. Glücklicherweise, bieten viele einfach APIs für solche Operationen auf unterschiedlichen topologischen Niveaus.

- **Komponente für die Analyse von Beziehung der Objekte** ACIS bietet verschiedene Mechanismen zum Analysieren der Objektbeziehung zwischen zwei Objekten. Es bietet allgemeine Funktionen für den Abstand von Objekte auf unterschiedlichen Niveaus, wie Körper und Körper, Körper und Oberfläche usw. Außerdem bietet es alternative Funktionen, die eine facettierte Annäherung an die Objekte verwenden, um den Mindestabstand zwischen ihnen zu bestimmen. Wenn weitere Informationen zu potenziellen Kollisionsbereichen benötigt werden, stellt ACIS eine Kollisionsfunktion bereit, um zu bestimmen, wie zwei Objekte interagieren, sofern sie interagieren. Darüber hinaus bietet ACIS eine Reihe von Funktionen, die die Einschlussbeziehung zwischen zwei Objekten oder zwischen einem Punkt und einem Objekt bestimmen.

3.2.3 Die Verbindung zwischen RobotStudio und ACIS

Wie oben vorgestellt wird, RobotStudio SDK ist in C# geschrieben und gleichzeitig ACIS ist in C++ geschrieben. Deshalb müssen wir eine Lösung finden, um in einer gemeinsamen .NET-Lösung zwischen diesen beiden Sprachen zu kommunizieren.

Einer der beste Lösung für die Kommunikation zwischen beiden Programmiersprachen ist durch die Kompatibilität von .NET bei Anwendung von C++/CLI. Die common language infrastructure (CLI) wird als eine von Microsoft entwickelte Variante der Programmiersprache C++ bezeichnet, die den Zugriff auf die virtuelle Laufzeitumgebung der .NET-Plattform mit Hilfe von speziell darauf zugeschnittenen Spracherweiterungen ermöglicht. C++/CLI erfüllt die ebenfalls von Microsoft entwickelte Spezifikation zur Sprach- und Plattform-neutralen Entwicklung und Ausführung von .NET-Anwendungen. Programme, die in C++/CLI geschrieben sind, können vom Compiler in common intermediate language (CIL) übersetzt und auf der virtuellen Maschine der .NET-Plattform betrieben werden.[7]

Um sich die Referenzklassen in C#, die automatisch von garbage collection (GC) verwaltet werden, und die Klassen in C++ voneinander zu unterscheiden, benutzt CLI eine andere Syntax für die Deklaration und Definition von den Referenzklassen in C#. Ein typischer Unterschied ist wie folgende Abbildung gezeigt.

```
array<int>^ numbers = gcnew array<int>(100);

String^ name;
```

Abbildung 20: Deklaration von Array und String in CLI

3.3 Einschubprozesserkennung

Wenn man über die Montage zwischen zwei Objekten nachdenkt, fällt ihm als Erstes die einfachste und gebräuchlichste Art von Montage ein: Einschub. Deshalb fokussiere ich in meine Praktikumsarbeit für den Einschub in einen Montageprozess.

3.3.1 Definition des Einschubprozesses

Die erste Frage für die Analyse des Einschubprozesses ist wie kann man den Einschubprozess vollständig definieren, sodass der Prozess bei Generierung des Roboterprogrammes direkt benutzt werden kann. Zur Verallgemeinerung und Vereinfachung des Prozesses kann man annehmen, dass der Einschubprozess ist nur von der Start-, Endposition und der Richtung abhängig. Außerdem der Einschub ist auf eine lineare Translationsbewegung beschränkt und deshalb kann die Start- und Endposition durch einen Abstand ersetzt werden.

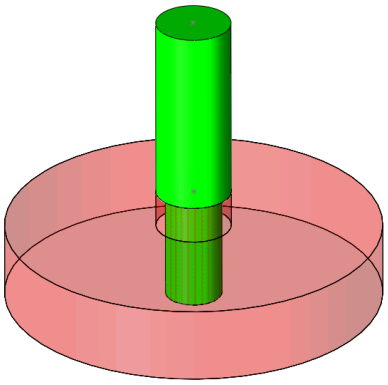

Abbildung 21: Ein Beispiel von Einschubprozess

3.3.2 Definition des Modells in CAD

Die zweite Frage ist wie kann man das CAD-Modell, das zwei Objekte beinhaltet, definieren. Wenn man den Einschubprozess einordnen möchte, ist die Anzahl der beteiligten Teile ein wichtiges Kriterium. Nach diesem Kriterium kann man den Prozess in zwei Gruppen teilen, nämlich eins zu eins und eins zu mehr.

Wenn wir eins-zu-mehr Baugruppe dekodieren, können wir verstehen, dass ein bewegliches Teil eine individuelle Montagebeziehung zwischen anderen Teilen in der Baugruppe hatte. Die Lösung für die mehrteilige Montage kann erreicht werden, indem ein bewegliches Teil entnommen und die Beziehung für

die festen Teile nacheinander ermittelt wird. Dann können wir die einzelnen Ergebnisse zwischen einem beweglichen und mehreren festen Teilen akkumulieren und den Montageprozess nacheinander durchführen. Aber dabei muss man beachten, dass die Reihenfolge eine wichtige Rolle spielt.

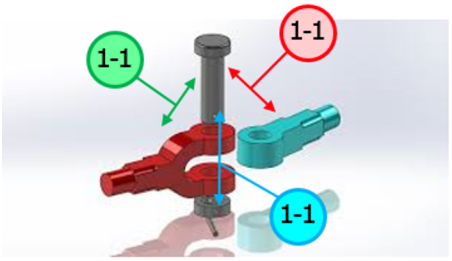

Abbildung 22: Ein Beispiel für eins-zu-mehr Baugruppe

Aus obengenannten Gründen, können wir das Modell so definieren:

- Die Anzahl der Montageteile beträgt zwei
- Ein Montageteil gilt als fest und ein anderes als beweglich
- Anfangszustand von Montageteile ist montiert

3.3.3 Die Lösung für Einschubrichtungserkennung

Die Grundidee

- Die Auswahl von relevantesten Flächen in Festkörper und EinschubKörper separat
- Extrahierung der möglichen Einschubrichtungen aus den ausgewählten Flächen
- Festlegung der potentialsten Einschubrichtung, indem der bewegliche Körper entlang der Richtung bewegen und eine Kollision zwischen beiden Körpern berechnen
- Berechnung der Einschubdistanz aus die festgelegten Einschubrichtung

Die Auswahl von relevanten Flächen spielt eine wichtige Rolle für die Analyse von Einschubprozess zwischen zwei Objekten, weil die relevantesten Oberflächen von festem Teil und beweglichem Teil unbedingt die Einschubrichtung enthalten. Also kommt es zur nächsten Frage, wie kann man die relevantesten Oberflächen definieren und herausfinden, sodass keine wichtigen Informationen für die Einschubrichtung übersehen werden, aber auch keine überflüssigen Informationen enthalten sind.

- **Entity Clash Calculations**
  Basierenden auf dieser Idee, habe ich zuerst die APIs für „Entity Clash Calculations“ benutzt, und die zwei Körpern als Eingänge für die Funktion eingegeben. Danach werden die zusammengestoßene Flächenpaare und deren Kollisionstypen ausgegangen. Die Vorteile für diese Methode sind eindeutig. Auf diese Weise erhalte ich nicht nur die Kollisionsflächenpaare, sondern auch den Kollisionstyp, die sehr nützlich ist, um nicht verwandte Flächenpaare auszuschließen. Weil ich mich nur für die Flächenpaare interessiere, deren Normalvektoren einander entgegengesetzt sind.

  Trotz der Bequemlichkeit habe ich bei Anwendung des Algorithmus in verschiedenen CAD-Dateien festgestellt, dass aufgrund der Toleranz- oder Abstandunterschied in Festkörper und beweglichen Körper zwischen unterschiedlichen CAD-Dateien viele verwandte Flächen, die die Information über die Einschubrichtung enthalten, nicht erkannt werden können.
  Einer der wichtigsten Gründe in der Praxis für den Fehlschlag des „Entity Clash Calculations“ Algorithmus ist aufgrund der Präzision verschiedener CAD-Modelle der Mindestabstand oder der Grenzwert für die Bestimmung, ob zwei Körper zusammengestoßen sind, sehr unterschiedlich. Beispielsweise kann bei einer großen mechanischen Baugruppe der Abstand zwischen Objekten einige Millimeter betragen, bei einem Präzisionsinstrument kann der Abstand zwischen Objekten jedoch nur einige Mikrometer betragen.

- **Entity Distance Calculation**
  Um obengenanntes Problem zu bewältigen, muss man den Abstand berücksichtigen. Deshalb habe ich stattdessen die APIs für „Entity Distance Calculation“ benutzt. Die verwandten Flächen nach diesem Kriterium sind die Flächen, die sich innerhalb der vordefinierten Entfernung von den anderen Körpern befinden.

  Mit diesen Funktionen kann man nach der Genauigkeit von unterschiedlichen CAD-Modellen den Grenzwert selbst festlegen und die entsprechenden Flächen herausfinden. Aber gleichzeitig sollen die Nachteile von diesem Algorithmus auch nicht übersehen werden. Einerseits, haben die Informationen zur Flächenpaaren und Kollisionstypen auf diese Weise verloren. Andererseits, gehören dazu viele nicht verwandte Flächen, von denen sich nur eine Kante oder ein Punkt im Abstand befindet und keine direkte Information für den Einschub beinhalten.

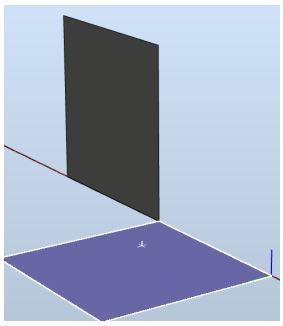

Abbildung 23: Nur ein Punkt von Flächen innerhalb des Abstandes

- **Die Kombination von zwei Lösungen**
  Die beiden Algorithmen haben ihre eigenen Vorteile und Nachteile, deshalb habe ich doch eine Kombination von diesen zwei Lösungen versucht.
  - **Erster Schritt:** Mit der Entfernungsberechnung werden alle Flächen von Festkörper und beweglichen Körper separat innerhalb des vordefinierten Abstandes ausgewählt
  - **Zweiter Schritt:** Lassen sich die Flächen verbleiben, die in einem anderen Körper eine entsprechende Fläche finden können, die sich innerhalb des Abstandes befindet, und deren Normalvektoren in entgegengesetzte Richtungen weisen.

  Durch die Kombination von zwei Lösungen reduziert die Anzahl der ausgewählten Flächen dramatisch und die nicht relevante Flächen werden ausgeschlossen.

Extrahierung der möglichen Einschubrichtungen Um die potenziellen Einschubrichtungen von den ausgewählten Flächen ableiten zu können, muss man zuerst festlegen, welche geometrische Informationen von einer Fläche der möglichen Einschubrichtung entsprechen. Nach der Analyse von unterschiedlichen Situationen kann ich feststellen, dass die Einschubrichtung immer identisch mit folgenden Richtungen ist:

- Die Richtung einer Kante auf einer Seitenfläche
- Die Richtung der Normalvektor auf Unterseite, falls es existiert
- Das Kreuzprodukt zweier Normalenvektoren, die nicht gegeneinader sind, auf zwei verschiedenen Seitenflächen.

Die nächste Aufgabe ist die Einordnung der ausgewählten Flächen. Am Anfang habe ich die Flächen in zwei Gruppen eingeteilt und das Kriterium ist ob eine einseitige Verlängerung des Normalvektors von der Flächemitte den Körper

berührt. Aus der Gruppe, die den Körper nicht berührt, können wir die Richtung nach außen durch Summation ihrer Normalenvektoren bestimmen. Danach habe ich alle Analyse in den anderen Gruppen durchgeführt, weil die Seitenflächen, deren Kante der Einschubrichtung entsprechen, befinden sich am meisten in der zweiten Gruppe.

Im ersten Schritt, habe ich die dahinter versteckte Geometrie berücksichtigt. Wenn die zugrunde liegende Geometrie einer Fläche ein Kegel ist, können wir annehmen, dass die Achse des Kegels eine der möglichen Einfügerichtungen ist. Wenn die ausgewählten Flächen mindestens zwei Seitenflächen enthalten, die sich nicht gegenüberliegen, können wir davon ausgehen, dass das Kreuzprodukt ihrer Normalenvektoren eine der möglichen Einschubrichtungen ist. Außerdem wenn die einseitige Verlängerung einer Kante in der Seitenfläche den Körper nicht berührt, können wir auch annehmen, dass die Richtung der Kante eine der möglichen Einschubrichtungen ist.

Festlegung der potentialsten Einschubrichtung Nach der Suche nach der möglichen Einschubrichtungen muss ich auch entweder die Anzahl der Einschubrichtungen einschränken oder die möglichste Einschubrichtung herausfinden. In diesem Bereich habe ich „Entity Entity Clash“ benutzt. Die Grundidee besteht darin, wenn der EinschubKörper entlang der Einschubrichtung herausgezogen wird, muss es in jede Position keine Kollision gibt.

Das Problem ist wie kann man den Abstand für den Kollisionstest festlegen. Egal der Abstand zu groß oder zu klein ist, wird der Test fehlgeschlagen. Die endliche Lösung läutert wie folgendes:

- Aus den endgültig ausgewählten Flächen im Fest- und Einschubkörper kann ich zwei Begrenzungsrahmen bekommen und lassen sich die zwei Begrenzungsrahmen schneiden, um einen kleineren Begrenzungsrahmen zu erhalten.
- Festlegung der Bewegungsdistanz, indem die Entfernung entlang der Richutung innerhalb des Begrenzungsrahmens berechnen.

Wenn entlang der Richtung keine Kollision gibt, kann die Richtung als einer der möglichsten Einschubrichtungen betrachtet wird.

Berechnung der Einschubdistanz Die Einschubdistanz muss gewährleisten, dass bei Anfangsposition von Einschubprozess keine Kollision zwischen beiden Körpern existiert. Von diesem Gedanken kann ich ausgehen, dass die Minimumeinschubdistanz zwischen Festkörper und Einschubkörper ist die Subtraktion von Maximaltiefe in Einschubkörper und Minimaltiefe in Festkörper entlang Einschubrichtung.

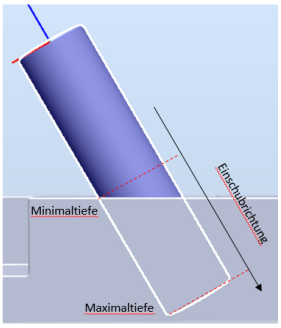

Abbildung 24: Ein Beispiel für die Berechung der Einschubdistanz

Der Algorithmus für die Berechnung ist in diesem Fall ziemlich eindeutig und einfach.

- Finden alle Eckpunkte in ausgewählten Flächen in beiden Körpern
- Projektieren alle Eckpunkte in die Einschubrichtung und Finden die Maximaltiefe für Einschubkörper und Minimaltiefe für Festkörper
- $Einschubdistanz = Maximaltiefe - Minimaltiefe$

4 Ergebnis und Disskusion

5 Zusammenfassung

6 Ausblick

Literatur

[1] wikipedia:Off-line programming (robotics)

[2] wikipedia:geometrie

[3] ABB:Offline-Programmierung leicht gemacht!

[4] wikipedia:Geodaten

[5] wikipedia: boundary representation

[6] wikipedia: Kurve (Mathematik)

[7] wikipedia: C++/CLI

Abbildungsverzeichnis

Abkürzungsverzeichnis

OLP Offline-Programmierung 4

TCP Werkzeugarbeitspunkt 4

SDK Software Development Kit 5

b-rep boundary representation 6

STEP the Standard for the Exchange of Product Model data 6

ACIS 3D ACIS Modeler 19

API application programm interface 19

CLI common language infrastructure 20

CIL common intermediate language 20

GC garbage collection 20