



Hochschule
Zittau/Görlitz
UNIVERSITY OF APPLIED SCIENCES

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	2
1.1	Problemstellung	2
1.2	Motivation	2
1.3	Zielstellung	3
2	Theoretische Grundlagen	4
2.1	Offline Programmierung	4
2.1.1	Definition	4
2.1.2	Robotstudio	5
2.2	Geometrie und Topologie	6
2.2.1	Geometrie	6
2.2.2	Topologie	10
2.3	CAD Modellierung	12
3	Aufgabenstellung	12
3.1	Addin-Entwicklung in Robotstudio	12
3.2	CAD-Modell Analyse Mithilfe von ACIS Modeler	16
3.3	Einschubrichtungserkennung	16
4	Methoden	16
5	Ergebnis und Diskussion	16
6	Zusammenfassung	16
7	Ausblick	16

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 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 hohe 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 Operationen zu konzentrieren, benötigt 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 Benutzer 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 gibt 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 nachzuziehen. 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 automatische Erzeugung der Position für den Einschub von 2 Objekte.

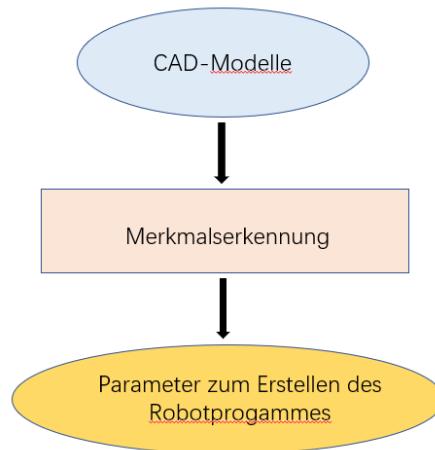


Abbildung 1: Bearbeitungsverfahren für CAD-Daten

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 Grundlagen

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 der 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.

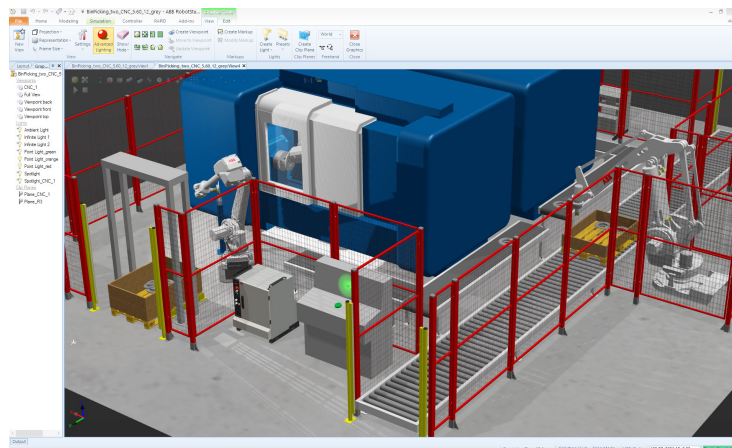


Abbildung 2: Roboterzelle in Robotstudio

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.

$$p(\vec{t}) = \vec{p}_0 + \vec{M} \cos t + \vec{m} \sin t$$

$$\vec{m} = \alpha(\vec{n} \times \vec{M})$$

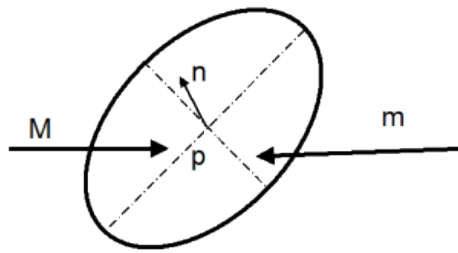


Abbildung 3: Ellipse

– 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.

$$p(\vec{t}) = a \cos t \vec{x}_0 + a \sin t \vec{y}_0 + bt \vec{z}_0$$

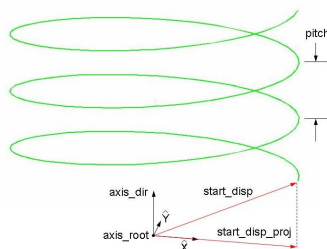


Abbildung 4: Helix

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.

Ähnlich 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.

$$p(\vec{u}, \vec{v}) = \vec{R}_0 + u\vec{U} + v\vec{V}$$

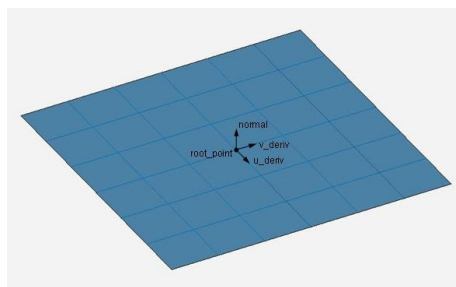


Abbildung 5: Ebene

– 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 Haupthalb winkels des Kegels definiert. Die Normale der Basisellipse repräsentiert die Achse des Kegels.

$$x = au \cos v$$

$$y = au \sin v$$

$$z = u$$

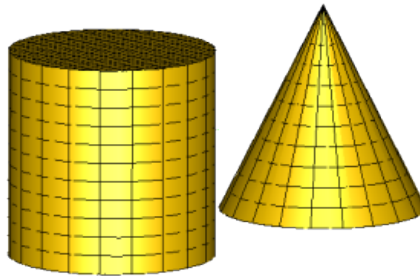


Abbildung 6: Cone

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 Geobjekten.

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.

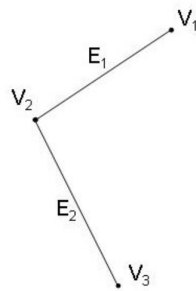


Abbildung 7: Ein Beispiel für die Bedeutung von Topologie

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.

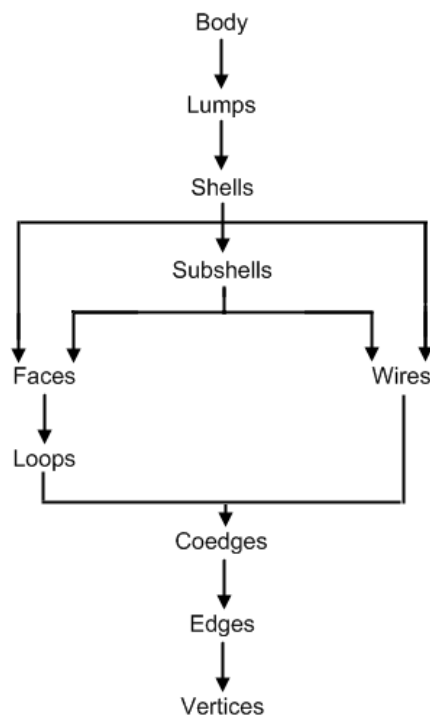


Abbildung 8: Hierarchische Beziehungen zwischen den topologischen Elementen

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.

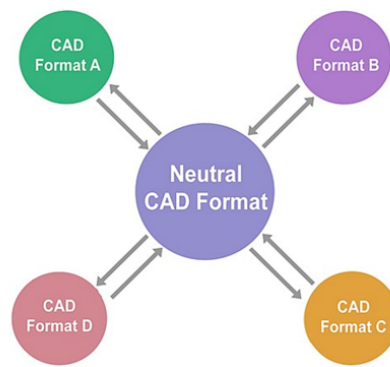


Abbildung 9: neutrales CAD-Dateiformat hilft Datenaustausch

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

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.

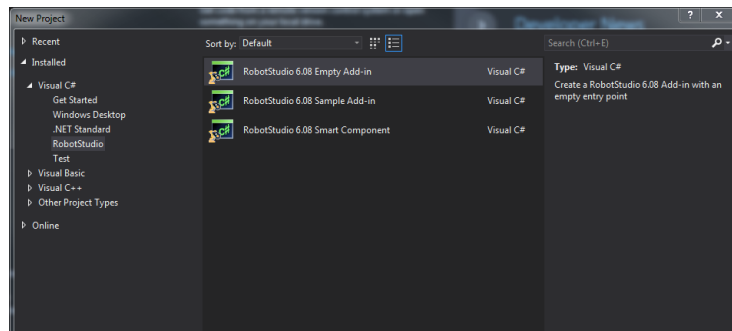


Abbildung 10: neues Projekt für RobotStudio Add-in auswählen

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


```

Class1.cs
Example
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 using ABB.Robotics.Math;
6 using ABB.Robotics.RobotStudio;
7 using ABB.Robotics.RobotStudio.Environment;
8 using ABB.Robotics.RobotStudio.Stations;
9
10 namespace Example
11 {
12     public class Class1
13     {
14         // This is the entry point which will be called when the Add-in is loaded
15         public static void AddinMain()
16         {
17         }
18     }
19 }

```

Abbildung 11: leere Lösung für RobotStudio Add-in

3) UI-Entwicklung

Im RobotStudio-Arbeitsbereich gibt es verschiedene Arten von Bereichen, z.B. Ribbonbereich, Eigenschaftsbereich und Arbeitsbereich.

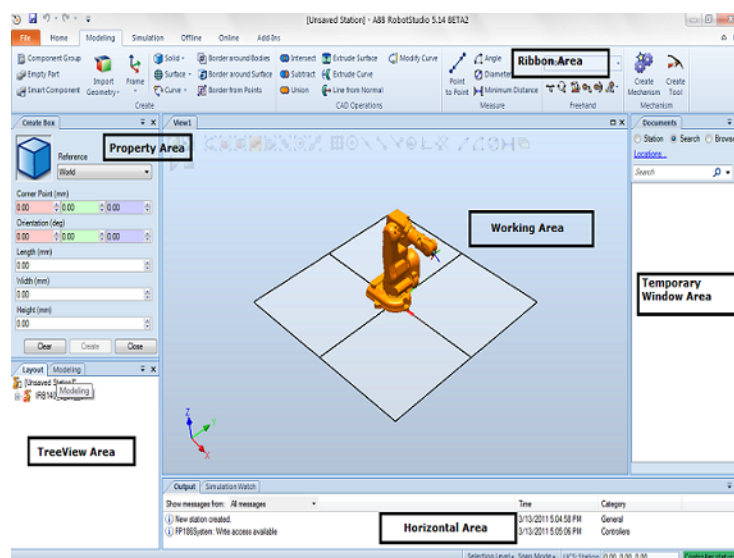


Abbildung 12: Arbeitsbereich in RobotStudio

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:

```
//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 13: Der Kode für die Erstellung von einem Button

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

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.

```
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), "RibbonIcons", "New_Insert.png"));
        btn_new_action.DefaultEnabled = true;
        ribbonGroup.Controls.Add(btn_new_action);

        //Include Separator between buttons
        CommandBarSeparator separator = new CommandBarSeparator();
        ribbonGroup.Controls.Add(separator);

        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 14: Ein Beispiel für eine Buttonklasse

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.

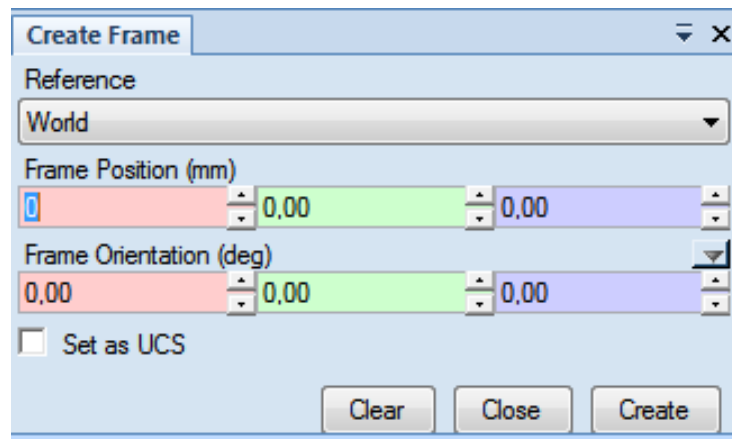


Abbildung 15: Ein Beispiel für die Anwendung von ToolWindow

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 Referenzrahmen zu bestimmen und endlich zu erzeugen.

3.2 CAD-Modell Analyse Mithilfe von ACIS Modeler

3.3 Einschubrichtungserkennung

4 Methoden

5 Ergebnis und Diskussion

6 Zusammenfassung

7 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)

Abbildungsverzeichnis

1	Bearbeitungsverfahren für CAD-Daten	3
2	Roboterzelle in Robotstudio	6
3	Ellipse	8
4	Helix	8
5	Ebene	9
6	Cone	10
7	Ein Beispiel für die Bedeutung von Topologie	11
8	Hierarchische Beziehungen zwischen den topologischen Elementen	11
9	neutrales CAD-Dateiformat hilft Datenaustausch	12
10	neues Projekt für RobotStudio Add-in auswählen	13
11	leere Lösung für RobotStudio Add-in	14
12	Arbeitsbereich in RobotStudio	14
13	Der Kode für die Erstellung von einem Button	15
14	Ein Beispiel für eine Buttonklasse	15
15	Ein Beispiel für die Anwendung von ToolWindow	16

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