Masterarbeit

SOFTWARETOOL ZUR SCHNELLEREN ADAPTIERUNG EINER 2D ROBOTERBAHNSTEUERUNG

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Ing. Markus Krempl, BSc
1610322023

betreut und begutachtet von
Dipl.-Ing. (FH) Gernot Hofer

Graz, Jänner 2018

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

Hiermit möchte ich mich zuerst bei meiner Frau, meinem Kind und meiner Familie für die Unterstützung während der Erstellung dieser Masterarbeit bedanken. Ein Dank gilt Herrn Dipl.-Ing. (FH) Gernot Hofer der meine Arbeit betreut hat.

Ein Dank gilt auch meinen Arbeitskollegen, die durch Korrekturlesen und der Hilfestellung bei der Programmierung zur Vollendung dieser Arbeit beigetragen haben. Weiters möchte ich mich bei der Firma M&R Automation für die Zurverfügungstellung des SCARA Testaufbaus bedanken.

KURZFASSUNG

In der Automobilindustrie gibt es unzählige Anwendungen, bei denen Industrieroboter Aufgaben eines Fertigungsprozesses übernehmen. Für die Erstellung und Anpassung von Roboterprogrammen sind Techniker erforderlich, die Erfahrung im Umgang mit Robotersteuerungen von unterschiedlichen Lieferanten haben. Zusätzlich sind spezielle Softwarepakete für die Roboterprogrammentwicklung ebenfalls erforderlich.

Ziel ist es, ein Software-Tool zu erstellen, mit dem ohne spezieller Grundkenntnisse in der Roboterprogrammierung Anpassungen an Roboterprogrammen getätigt werden können. Ein zusätzliches Ziel ist es, die Änderungszeit dieser Programme zu verringern.

Das Softwaretool enthält intuitive Funktionen zur Anpassung von 2D-Roboterbahnen. Die erste Funktion besteht darin, ein Roboterprogramm in das Softwaretool zu importieren, das aus Koordinaten aus einer Bahn besteht. Darauf basierend werden die Koordinaten verbunden, um den Weg der Roboterbewegung anzuzeigen. Außerdem sollen die Koordinaten veränderbar sein. Die Funktion wird per Drag & Drop realisiert. Diese Änderungen können wiederum im Programm des Roboters abgespeichert werden.

Mit diesem Softwaretool ist man in der Lage, Roboterprogramme ohne besondere Erfahrung in der Programmierung und Bedienung zu ändern. Darüber hinaus verkürzt die Verwendung dieses Werkzeugs die Arbeitszeit für die Anpassung von Roboterprogrammen.

ABSTRACT

There are countless applications in the automotive industry in which industrial robots take on tasks of a manufacturing process. For the creation and adapting of robot programs, maintenance technicians are required, who have experience in handling robot controllers from different suppliers. In additional special software packages for robot program development are also required.

The aim of this thesis is to create a software tool with the requirement to easily adapt robot programs without any experience in handling them. An additional goal is to decrease the changing time of those programs.

Therefore, the software tool includes intuitive functions for adapting 2D robot pathways. The first function is to import a robot program into the software tool which includes coordinates from a path. Based on that the coordinates will be connected to show the path of the robot’s movement. Furthermore, the coordinates shall be alterable. The function is realized by drag and drop. Finally, it is possible to feed back the changes to the original robot program.

With the use of that software tool it is possible to change robot programs without any special experience in programming and robot teaching. Furthermore, the use of this tool decreases working time for adapting robot programs.

INHALTSVERZEICHNIS

1 EINLEITUNG

Industrieroboter sind in der heutigen Zeit nicht mehr wegzudenken. So werden unzählige Roboter in Industrieanlagen integriert, um eine Steigerung der Anlagenverfügbarkeit und Ausbringung zu gewährleisten. Durch den Einsatz dieser Roboter kann auch eine gleichbleibende Qualität der einzelnen Aufgaben erzielt werden. Roboter können ohne Pausen ihre Tätigkeiten verrichten. Der Einsatz von technischen Hilfsmittel bei der Arbeit löst jedoch nicht den Menschen ab, es kommt dadurch zu einer Verschiebung der Arbeit. Zu einer hochautomatisierten Anlage, mit integrierten Industrierobotern werden zusätzliche Arbeitskräfte in der Instandhaltung benötigt um die Wartungsarbeiten an den Robotern durchzuführen.

Um eine Industrieanlage wirtschaftlich zu betreiben, wird sie von vorhinein so ausgelegt, dass unterschiedliche Produkte gefertigt werden können. So kann diese Anlage über mehrere Jahre genützt werden. Läuft ein Produkt aus, kann diese Anlage auf ein anderes Produkt angepasst werden. Ändert sich das Produkt nur geringfügig, sind neben Wartungsarbeiten auch Änderungen in der Robotersteuerung notwendig. Um diese Anpassungen durchzuführen wird Personal benötigt, welches die Erfahrung im Umgang mit Robotersteuerungen unterschiedlicher Hersteller mitbringt. Ebenso werden spezielle Softwarepakete für die Roboterprogrammentwicklung benötigt. Ist dieses Fachpersonal jedoch nicht vorhanden, wird diese Arbeit extern zugekauft.

Im nächsten Kapitel wird auf die Ausgangssituation eingegangen.

2 AUSGANGSSITUATION

Geringfüge Änderungen in Roboterprogrammen sind Anpassungen eines Bahnsegmentes, die durch Änderung eines einzelnen Punktes realisiert werden. Ein Bahnsegment benötigt eine variable Anzahl an Punkten, welche die Robotersteuerung zeilenweise abarbeitet. Je höher die Anzahl der Punkte, desto unübersichtlicher wird das Roboterprogramm. Durch eine hohe Anzahl an Punkten wird die Änderung eines Programmteils erschwert. Für geschultes Personal sind solche Änderungen in kürzester Zeit realisierbar.

2.1 Einsatzgebiet

Bahnsteuerungen werden unter anderem beim Auftragen eines Klebers auf ein Werkstück eingesetzt. Verwendet wird diese Technik bei Werkstücken, die bedingt durch ihren Fertigungsprozess nicht als Ganzes gefertigt werden können. Speziell bei einem Achsgetriebe eines Fahrzeuges wird diese Fertigungsweise angewendet. Beim Montageprozess in einer Fertigungsanlage werden diese Bauteile zusammengefügt und danach miteinander verschraubt. Bevor jedoch dieser Schritt erfolgen kann, wird auf der sogenannten Dichtfläche ein Dichtmittel aufgetragen, welches zur Abdichtung dieser Berührungsflächen dient. In der nachfolgenden Abbildung ist eine Explosionsansicht eines Achsgetriebes mit gekennzeichneter roter Dichtfläche schematisch dargestellt.

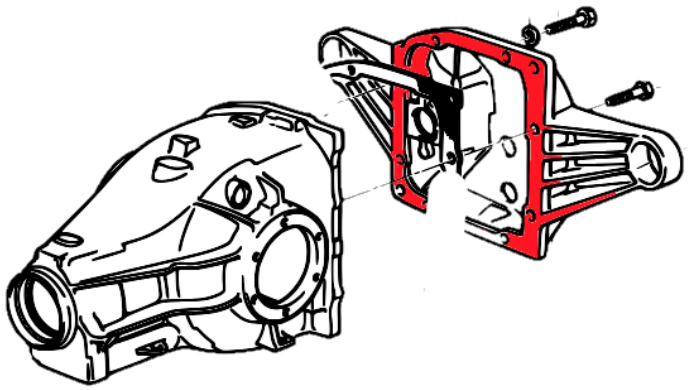

Abb. 1: Explosionsansicht Achsgetriebe, Quelle: Auto-Leebmann GmbH - BMW / MINI Vertragshändler (2017), Online-Quelle [12.11.2017], leicht modifiziert

Auf die dargestellte Dichtfläche wird das Dichtmittel in Form einer „Schnur“ aufgetragen. Durch den Einsatz eines Industrieroboters werden diese Dispensiervorgänge mit einer hohen Qualität durchgeführt. Das benötigte Dichtmittel wird dabei mit Druckluft durch ein Dosierventil gedrückt. Dabei gibt es zwei Arten wie dieser Prozess abläuft. Einerseits lässt sich das Dosierventil direkt am Roboter montieren, oder das Ventil bleibt in einer Halterung montiert und der Roboter bewegt das Bauteil an dem Ventil vorbei.

Änderungen an der Bahn werden mit der Teach-in Programmierung durchgeführt. Bei dieser Methode werden die zu ändernden Positionen mit dem Teach-Panel angefahren, neu eingelernt und danach abgespeichert.[1]

Die Aufgabenstellung für diese Arbeit ergibt sich nun aus folgender Ausgangssituation:

- Verschiedene Aufgabenstellungen die eine Änderung eines Roboterprogrammes erfordern
- Einsatz von unterschiedlichen Industrieroboterherstellern
- Benötigtes Fachpersonal für Handhabung von Industrierobotern
- Verschiedene Robotersoftwarepakete
- Änderung eines Roboterprogrammes mit Hilfe der Teach-in Programmierung

[1] Vgl. Hesse (2000), S. 143.

2.2 Bestehenden Softwarelösungen

Jeder Roboterhersteller bietet eigene Softwarelösungen zur Realisierung von Roboteransteuerungen an. Mit diversen Softwarepaketen, welche in den meisten Fällen herstellerspezifisch sind, kann eine offline Roboterprogrammierung, sowie auch 3D-Simulationen erstellt werden. Softwarepakete namhafter Roboterhersteller sind lizenzpflichtig. Meist wird eine 30 tägige Testversion angeboten, um die Funktion und Bedienbarkeit der Software testen zu können. Zusätzlich werden Schulungen für das Anwenderpersonal benötigt, welche ebenfalls nicht kostenfrei sind.

Die Erstellung von Roboterprogrammen mit Offline Programmierung und Simulationen sind für spezielle Anlagen nicht mehr wegzudenken. Durch eine Robotersimulation können frühzeitig Fehler am Layout der Anlage gefunden und behoben werden. Ist die Reichweite des Roboters in der Simulation nicht gegeben, kann in Zusammenarbeit mit dem Konstrukteur ein neues Zellenlayout entworfen werden. Ein Zellenlayout beinhaltet dabei Bearbeitungsstationen und Industrieroboter, welcher die Station mit Bauteilen versorgt. Auf die Anforderung abgestimmt, können spezielle Zusatzpakete für Schweiß-Applikationen oder Lackieranwendungen als Erweiterung der Software erworben werden.

Firmen die im Anlagenbau tätig sind und Roboterprogrammierung anbieten, benutzen verschiedene Software-Tools der Roboterhersteller. Ohne eine spezielle Schulung oder Wissen über Roboterprogrammierung sind Änderungen am Roboterprogramm jedoch nicht durchführbar. Um Änderungen am Programm durchführen zu können, muss die jeweilige Roboterprogrammiersprache erlernt werden. Doch nicht für jede Änderung am Programm lohnt es sich eine Robotersoftware anzuschaffen.

Um einen Überblick von bestehenden Softwarelösungen zu bekommen, wurde im Vorfeld dieser Arbeit eine Tabelle erstellt, die den Kostenaufwand, den Schulungsaufwand und die Benutzerfreundlichkeit dieser Programme widerspiegeln soll. Für das Software-Tool wurden ebenfalls das gewünschte Ergebnis eingetragen. Diese angeführten Bewertungskriterien ergeben sich aus den Erfahrungswerten des Fachpersonals der Firma M&R Automation. Die Benutzerfreundlichkeit wurde mit Hilfe eines Umfragebogens eruiert, welche im Anhang ersichtlich ist.

Benutzerfreundlichkeit: Für die Beurteilung der Benutzerfreundlichkeit wurden folgende Tatsachen ermittelt. Sie sollen zeigen wie sich die jeweilige Software bedienen lässt.

Die Robotersoftware

- ist intuitiv bedienbar,
- bietet eine ausreichende Dokumentation,
- ist einfach verständlich,
- lässt eine effektive Arbeitsweise zu und
- lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen.

Bei der Beurteilung dieser Tatsachen stehen folgende Optionen zur Auswahl: trifft nicht zu, trifft eher nicht zu, trifft eher zu, trifft zu und keine Erfahrung.

Die Benutzerfreundlichkeit wird in Schulnoten von eins bis vier vergeben. Für unzureichende Erfahrung, abhängig der Software werden die Buchstaben (ka) für keine Angabe eingetragen.

Die Gegenüberstellung der verschiedenen bewerteten Software-Pakete und das Ergebnis der Benutzerfreundlichkeitsbefragung sind in folgender Tabelle dargestellt.

	ABB Robot Studio	Kuka WorkVisual	Fanuc ROBOGUIDE	Hirata BASIC Development Environment
Kostenfrei	Begrenzt	Ja	Nein	Ja
Fachpersonal	Wird benötigt	Wird benötigt	Wird benötigt	Wird benötigt
Herstellerunabhängig	Nein	Nein	Nein	Nein
Programmänderungen	ohne erweiterte Kenntnisse nicht durchführbar	ohne erweiterte Kenntnisse nicht durchführbar	ohne erweiterte Kenntnisse nicht durchführbar	ohne erweiterte Kenntnisse nicht durchführbar
Schulungsaufwand	90%	90%	90%	90%
Benutzerfreundlichkeit	1 (11 von 16 Personen)	3 (10 von 16 Personen)	2 (5 von 16 Personen)	3 (1 von 16 Personen)

	Siemens RobotExpert	Stäubli VAL 3	Software Tool
Kostenfrei	Nein	Nein	Ja
Fachpersonal	Wird benötigt	Wird benötigt	Wird nicht benötigt
Herstellerunabhängig	Ja	Nein	Ja
Programmänderungen	ohne erweiterte Kenntnisse nicht durchführbar	ohne erweiterte Kenntnisse nicht durchführbar	einfach durchführbar
Schulungsaufwand	90%	90%	20%
Benutzerfreundlichkeit	ka	1 (3 von 16 Personen)	noch keine Erfahrung

Tabelle 1: Gegenüberstellung Software-Pakete, Quelle: Eigene Darstellung

Zusammenfassung: Diese Gegenüberstellung zeigt, dass es sehr unterschiedliche Ansätze gibt um eine Robotersoftware anzubieten. Zwei Roboterhersteller setzen auf eine kostenfreie Lösung, den Roboter über eine Software zu programmieren. Das ABB Robot Studio ist generell auch kostenfrei, jedoch kann bei dieser Version keine Offline Programmierung durchgeführt werden. Hierzu werden kostenpflichtige Zusatzpakete benötigt.

Um diese Programme zu bedienen und somit Programmänderungen durchzuführen, wird ebenso Personal mit dementsprechendem Fachwissen benötigt. Diese Tastsache bestätigt sich auch in dem angegebenen Schulungsaufwand. Von den hier aufgelisteten Software-Paketen ist nur die Software der Firma Siemens universell für eine Roboterprogrammierung einsetzbar.

Bei der Umfrage in Zusammenhang mit der Benutzerfreundlichkeit wurden 16 Personen befragt, welche sich hauptsächlich mit Anwendungen in der Robotik beschäftigen. Es stellte sich heraus, dass unter den Testpersonen die beiden Software-Pakete der Firma ABB und Kuka am häufigsten im Einsatz sind. Die Note der Benutzerfreundlichkeit wurde anhand der eingetragenen Erfahrungen vergeben.

Im folgenden Unterkapitel wird auf die Zielsetzung dieser Arbeit eingegangen. Ausgehend von den Nachteilen bestehender Softwarelösungen soll ein neues Software Tool erstellt werden, dass ohne spezielle Kenntnisse in der Roboterhandhabung Programmänderungen möglich macht.

2.3 Zielsetzung

Durch diese Arbeit soll ein Werkzeug entstehen, das auch Laien in der Roboterprogrammierung ein erleichtertes Handling im Umgang mit verschiedenen Roboterprogrammen ermöglicht. Es soll der Visualisierung und Adaptierung von Roboterbahnsegmenten dienen.

Im Gegensatz zu herkömmlichen Robotersoftwaretools, konzentriert sich dieses Tool vorrangig auf eine Adaptierung einzelner Programmelemente eines Roboterprogrammes. Dabei sollen nur die Programmteile behandelt werden, die für Optimierungen angepasst werden müssen. Dadurch können Kosten für Software und Schulungen eingespart werden. Durch den Wegfall der Teach-In Programmänderung mit Hilfe des Teach-Panels kann es ermöglicht werden, dass auch ungeschultes Personal einfach Änderungen durchführt. Durch die Änderung des Programmes mit Hilfe des Software-Tools sollen ebenfalls Fehlerquellen minimiert werden.

Das Software-Tool soll dabei nicht in den Programmaufbau eingreifen, sondern lediglich den Inhalt der projektierten Befehle ändern. Konkret soll eine Visualisierung und Anpassung von Koordinaten einer Roboterbahn ermöglicht werden. Dabei sollen Programmzeilen jeder Roboterprogrammiersprache, die in das Tool implementiert werden, eingelesen werden können. Weiters soll es die Möglichkeit geben, Robotercodedateien als Text, oder direkt vom Robotersystem zum Beispiel über das Dateiübertragungsprotokoll (File Transfer Protocol, FTP) einzulesen.

Um eine grafische Darstellung der eingelesenen Punkte zu ermöglichen, soll der Robotercode zeilenweise eingelesen werden. Die dadurch eingelesenen Punkte sollen mit Hilfe von Linien und Kreissegmenten miteinander verbunden dargestellt werden. Somit entsteht eine grafische Darstellung der Roboterbahn. Die einzelnen Punkte sollen dabei so dargestellt werden, dass sie sich von den Verbindungslinien abheben und leicht ersichtlich sind. Diese Verbindungslinien sollen in der Visualisierung nur als einfache Verbindung der Punkte gezeichnet werden. Die Visualisierung etwaiger Punktverschleifungen werden für diese Anforderung der Änderung nicht benötigt.

Dargestellte Punkte einer Roboterbahn sollen dabei exakt das Bahnsegment einer Roboteranwendung abbilden. Durch eine weitere Funktion im Software-Tool soll es möglich sein, diese eingelesenen Punkte so darzustellen als würde die Bahn direkt am Werkstück verlaufen.

Um die Beziehung vom visualisierten Bahnsegment zum Werkstück herzustellen, soll ein Bild der Bahn, oder des zu bearbeiteten Bauteils als Hintergrund in die Visualisierung eingefügt werden. Dabei müssen jedoch der Koordinatenursprung und der Bezug zwischen einem Bildpunkt und der Maßeinheit mm definiert werden. Für den Bezug zwischen Bildpunkten und der Maßeinheit wird ein definierter Abstand im Bild benötigt werden.

Dieser soll durch Setzen zweier Punkte im Bild durchgeführt werden. Über diese zwei Punkte soll die Anzahl der Pixel im Bild gezählt und mit einer Eingabe des tatsächlichen Abstandes dieser Punkte ein Skalierungsfaktor errechnet werden. Ein Abstand von z.B. 100 mm sind dann 1000 Bildpunkte. Mit dieser Information soll die Beziehung der visualisierten Bahn und dem Werkstück definiert werden. So wie in der Abb. 2 dargestellt, könnte die Skalierung des Bildes mit Hilfe der abgebildeten Punkte A und B erfolgen. Der Abstand dieser Punkte zueinander muss dabei bekannt sein. Die nachfolgenden Abbildlungen Abb. 2

und Abb. 3 zeigen die Dichtflächen nicht exakt von oben und dienen lediglich als Beispiel für die behandelten Funktionen.

Abb. 2: Mögliche Skalierungsfunktion, Quelle: Streetlights Vertriebs GmbH (2010), Online-Quelle [30.November.2017] (leicht modifiziert).

Nach der Definition der Punkte A und B, und der erfolgten Eingabe des tatsächlichen Abstandes, soll der Skalierungsfaktor automatisch errechnet werden. Wenn danach eine Roboterbahn in das Tool geladen wird, ist diese Bahn jedoch zum Bild verschoben. Der Zustand nach dem Einlesen der Roboterbahn und des bereits skalierten Bildes soll, wie in der nächsten Abbildung dargestellt, im Software-Tool ersichtlich sein.

Abb. 3: Zustand nach dem Einlesen des Bildes, Quelle: Streetlights Vertriebs GmbH (2010), Online-Quelle [30.November.2017] (leicht modifiziert).

Um diese Verschiebung auszugleichen, muss das Bild zur dargestellten Bahn verschoben werden können. Die Definition, wo sich der Koordinatenursprung des Roboterprogrammes befindet, soll dadurch auch nicht verändert werden. Für das Software-Tool ist es dennoch wichtig den Bezug zur Realität herzustellen. Nur so lässt sich eine Änderung der Bahn mit Hilfe eines Bildes des originalen Werkstückes realisieren. Somit muss für eine Änderung eines Punktes, nicht mehr die konkrete Zeile im Roboterprogramm gefunden werden, sondern kann über die visualisierte Bahn erfolgen. Diese Verschiebung des Bildes ist in der folgenden Abbildung dargestellt.

Abb. 4: Zustand nach der Verschiebung des Bildes, Quelle: Streetlights Vertriebs GmbH (2010), Online-Quelle [30.November.2017] (leicht modifiziert).

Änderung und Speichern von Koordinaten.

Um eine Änderung der Bahn durchzuführen, soll es zwei Möglichkeiten geben. Einerseits soll durch Anklicken der Endpunkte eine Möglichkeit geschaffen werden die X- und Y-Koordinaten zu ändern. Wird die Änderung bestätigt, soll sich die Visualisierung der Punkte aktualisieren. Als zweite Möglichkeit soll durch Ziehen des Endpunktes mit der Maus die gewünschte Änderung erfolgen. Beim Ziehen des Endpunktes mit der Maus soll die Visualisierung ebenso kontinuierlich aufgebaut werden.

Das Software-Tool soll somit eine intuitive Änderung der Bahn unterstützen. Durch das Einlesen der Bahn und die Skalierung des Hintergrundbildes können Fehler in der Bahn leicht gefunden werden. Meist sind auch nur Optimierungen an der Bahn erforderlich. In der Abb. 5 ist ein Anwendungsfall für eine Optimierung einer Bahn dargestellt. Der Roboter dispensiert dabei ein Dichtmittel auf eine Getriebedichtfläche. Im markierten Bereich „F“ ist die Dichtmittelschnur zu nah an der Dichtflächenkante platziert. Mit dem Software-Tool soll der Punkt mit Hilfe der Maus auf eine neue Position verschoben werden können.

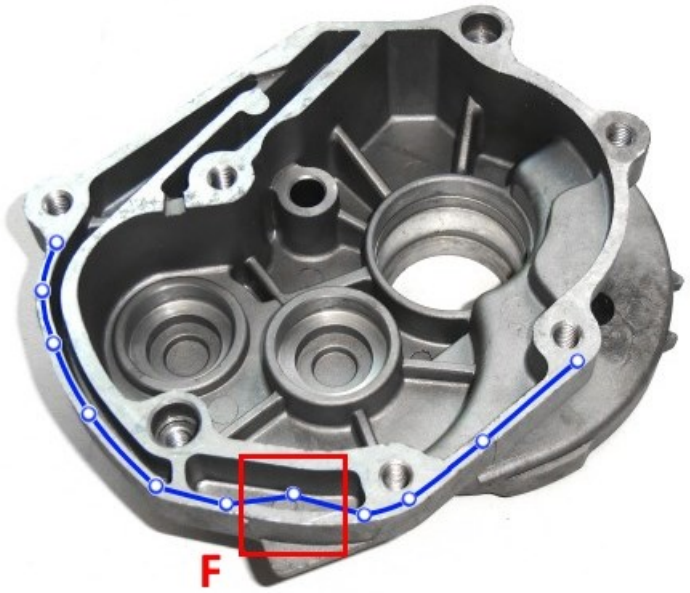

Abb. 5: Simulierter Anwendungsfall, Quelle: Streetlights Vertriebs GmbH (2010), Online-Quelle [30.November.2017] (leicht modifiziert).

Durch die Änderung des Punktes soll nur die X und Y Komponente des Fahrbefehls angepasst werden. Beim Speichern der Änderung sollen die neuen Koordinaten im jeweiligen Fahrbefehl gespeichert werden. Das erzeugte Roboterfile soll über eine definierte Kommunikationsmethode direkt in die Robotersteuerung übernommen werden.

Nach Abschluss des praktischen Teils soll dieses Software-Tool an einem Roboter getestet und evaluiert werden.

2.4 Abgrenzung

Wie schon im Titel dieser Arbeit verankert, bezieht sich das Software Tool auf 2D Roboterbahnsteuerungen. Für die Umsetzung und die grafische Anzeige von Bahnsegmenten, ist es notwendig, die benötigten Informationen der jeweiligen Roboterprogrammiersprache auszuarbeiten. Um den Unterschied verschiedener Programmiersprachen aufzuzeigen, werden in dieser Arbeit die Sprache Rapid der Firma ABB und der Maschinencode G-Code näher betrachtet. Für die Entwicklung des Software Tools werden im ersten Schritt drei Befehle des G-Codes implementiert. Mit diesen Befehlen lässt sich bereits eine einfache Roboterbahn abbilden.

Ausgehend von der Zielsetzung und der Abgrenzung, die eine genaue Vorstellung des Software-Tools darstellt, wird in den nachfolgenden Kapiteln auf die einzelnen Thematiken eingegangen um diese Zielsetzung zu erreichen.

3 ROBOTIK

Da in dieser Arbeit Begriffe der Robotik verwendet werden, wird in Grundzügen auf die Eckpunkte von Industrierobotern und Werkzeugmaschinen eingegangen.

3.1 6 Achsroboter

In den nachfolgenden Abschnitten werden die Grundlagen für einen 6 Achsroboter beschrieben. Sein Einsatzgebiet reicht von einfachen „Pick and Place“ Anwendungen bis hin zu komplexen Bahnsteuerungen zum dreidimensionalen Aufbringen von Dichtstoffen. Des Weiteren kann ein entsprechender Roboter mit einem Werkzeugwechsler mehrere Aufgaben erledigen, wie zum Beispiel das Einlegen eines Blechteiles in eine Vorrichtung. Nach erfolgtem Werkzeugwechsel vom Greifer auf eine Schweißzange kann der Roboter dann an den vorgegebenen Stellen Punktschweißungen durchführen. Durch so einen Werkzeugwechsel ist es möglich, Roboter in zusammenhängenden Anlagen, sowohl platztechnisch als auch ablauftechnisch, ideal auszunutzen.

3.1.1 Aufbau Kinematik

Die Anzahl und Anordnung der an den Bewegungen beteiligten Achsen definieren den kinematischen Aufbau eines Roboters. Die Achsen dienen zum Bewegen des Werkzeuges bzw. des Werkstücks im Raum, welches am Roboter montiert ist. Bei jeder Achse spricht man von einer unabhängig voneinander gesteuerten und geführten Einheit. Am Roboter montierte Werkzeuge, wie zum Beispiel ein Greifer oder eine Schweißzange, werden nicht als Achse bezeichnet, sondern nur als Hilfseinrichtung. Montierte Werkzeuge am Roboterflansch werden ebenfalls als Endeffektoren bezeichnet.[2]

Bei Industrierobotern gibt es zwei Arten von Achsen:[3]

- Rotatorische Achse
    - Fluchtende Achsen bzw. vertikale Achsen
    - Nicht fluchtende Achsen bzw. horizontale Achsen (in einem Drehgelenk liegende Achsen)
- Translatorische Achse
    - Nicht fluchtende Verschiebeachsen
    - Fluchtende Teleskopachsen
    - Linearschlitten als Verfahr-Achsen

Translatorische Achsen werden für ein X, Y, und Z Handling eingesetzt. Bei dieser Aufbauart der Achsen sind ebenso 6 Achssysteme und mehr möglich.

In der folgenden Abbildung ist ein Beispiel für rotatorische Achsen ersichtlich.

[2] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 36.

[3] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 36.

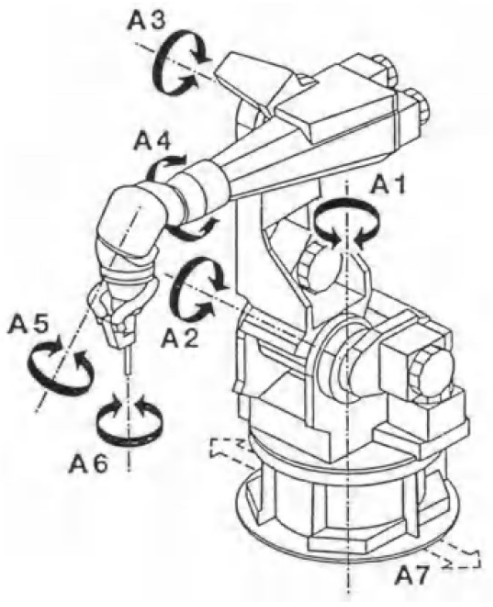

Bild 3-1
Beispiele für rotatorische Achsen (KUKA)

Achsen A1 bis A6	rotatorisch
Achsen A1, A4, A6	vertikal bzw. fluchtend
Achsen A2, A3, A5	horizontal bzw. nicht fluchtend

Abb. 6: Darstellung eines 6 Achsroboters, Quelle: Bartenschlager/Hebel/Schmidt (1998), S. 36.

Da jeder Roboterhersteller etwas anders konstruiert, können die Roboter an sich schwer miteinander verglichen werden. Um einen Vergleich einzelner Roboter in ihrer Kinematik zu gewährleisten, kann man auf die sogenannten kinematischen Ersatzschaltbilder zurückgreifen. Zur Veranschaulichung, wird im folgenden Bild ein 6 Achsroboter mittels eines kinematischen Ersatzschaltbildes dargestellt. In dieser Abbildung geht hervor, ob es sich um eine translatorisch bzw. rotatorisch bewegte Achse handelt, und ob diese dann fluchtend oder nicht fluchtend ausgeführt ist.[4]

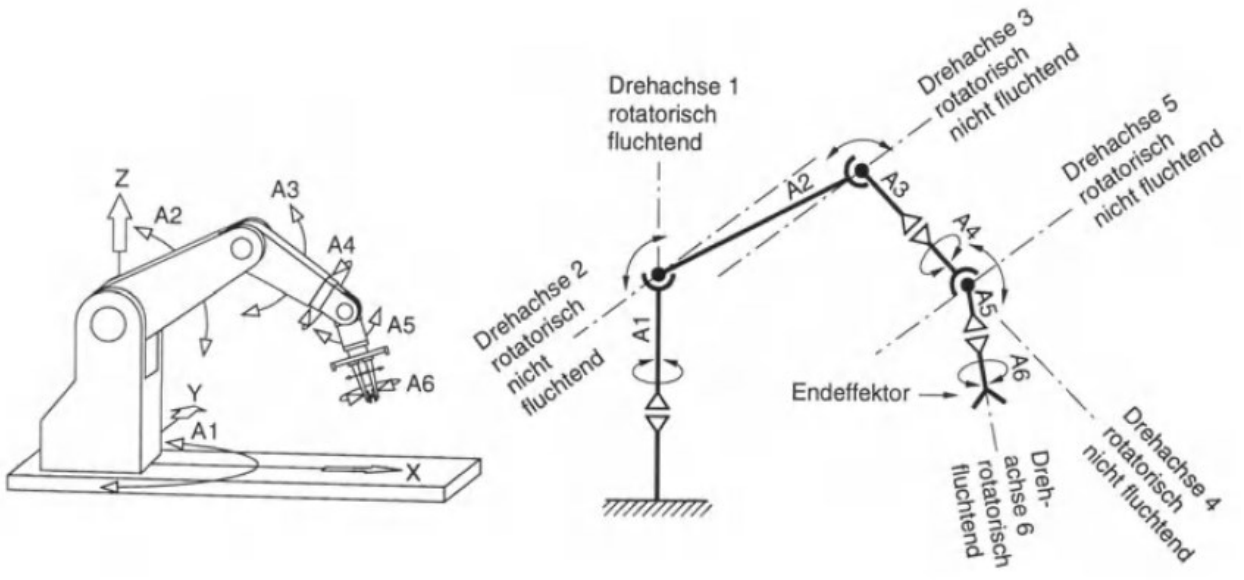

Abb. 7: 6 Achsroboter mit kinematischen Ersatzschaltbild, Quelle: Bartenschlager/Hebel/Schmidt (1998), S. 38.

[4] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 37 ff.

3.1.2 Steuerung

Um einen Bewegungsablauf zu realisieren, koordiniert die Steuerung das Zusammenwirken aller am Bewegungsprozess beteiligten Komponenten. Damit auch Fremdgeräte, wie Greifer oder Schweißzangen angesteuert werden können, verfügt die Steuerung, je nach Konfiguration, über sämtlich benötigte Schnittstellen:[5]

- TCP/IP
- Digitale Ein/ Ausgänge
- Analoge Ein/Ausgänge

In der Regel werden Industrieroboter in übergeordnete Anlagen integriert und arbeiten Befehle bzw. Anforderungen von einer Speicher Programmierbaren Steuerung (SPS) ab. Diese Kommunikation kann ebenfalls über eine der oben genannten Schnittstellen erfolgen. Natürlich ist die Steuerung auch für das Steuern und das Überwachen des Programmablaufs zuständig.[6]

3.1.3 Bediengerät

Das Bediengerät ist ein Teil der Steuerung und dient zur Kommunikation zwischen Mensch und Steuerung. Mit dem Bediengerät kann das Programm editiert bzw. können sämtliche Roboterfunktionen gesteuert werden. Diese sind jedoch hersteller- bzw. typenspezifisch und in den jeweiligen Handbüchern nachzulesen. Als Beispiel für ein Bediengerät wird ein ABB Flex-Pendant dargestellt.

Abb. 8: Roboterbediengerät, Quelle: Angelehnt an ABB (Hrsg.) (2016), S. 43.

Mit diesem Bediengerät kann der Roboter mit Hilfe des Steuerknüppels bewegt werden. Bei anderen Robotern erfolgt die Bewegungsansteuerung über Cursor Tasten. Über die Funktionstasten an der rechten Seite können die Bewegungsart bzw. Schrittweite und noch andere Einstellungen geändert werden. Kleinere Programmänderungen werden ebenfalls über das Bediengerät durchgeführt. Bei größeren Änderungen greift man auf Programmiereditoren, oder herstellerspezifische Softwarepakete zurück. Aus Sicherheitsgründen hat jedes Bediengerät einen Zustimmtaster und einen Not-Halt Taster. Um die Personensicherheit zu gewährleisten, ist eine Bewegung des Roboters mit dem Bediengerät, nur mit einem halb durch gedrückten Zustimmtaster möglich.[7]

[5] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 19 f.

[6] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 19 f.

[7] Vgl. ABB (Hrsg.) (2016), S. 127 ff.

3.1.4 Werkzeugarbeitspunkt

Um die Position des Roboters zu definieren, wird der Werkzeugarbeitspunkt verwendet. Dieser Arbeitspunkt befindet sich auf einem definierten Punkt am Werkzeug des Roboters und wird als Tool Center Point (TCP) bezeichnet. Als Werkzeug kann ein Bauteilgreifer, oder eine Düsenspritze einer Klebepistole zum Einsatz kommen. Es können auch mehrere TCPs definiert werden, wobei jedoch nur einer aktiv sein kann. Der TCP kann auf einem beliebigen Punkt eines Werkzeuges gesetzt werden. Die Position des TCPs bezieht sich immer auf ein definiertes Koordinatensystem des Roboters. Dadurch kann die Programmierung und Anpassungen der Roboterprogramme einfacher gestaltet werden. Wird kein Koordinatensystem definiert, so befinden sich alle Positionen des Roboters in dem sogenannten Basiskoordinatensystem.[8]

3.1.5 Koordinatensysteme

Basiskoordinatensystem: Im Basiskoordinatensystem befinden sich der Ursprung im Sockel des Roboters. Alle Stellungen des Roboters befinden sich in diesem Koordinatensystem welche in der Abb. 9 ersichtlich ist.[9]

Anwenderkoordinatensystem: Das Anwenderkoordinatensystem bezieht sich auf eine Arbeitsfläche des Roboters, welches an verschiedenen Position mit unterschiedlichen Orientierungen arbeitet. Für jede Arbeitsfläche kann ein eigenes Koordinatensystem angelegt werden. Befinden sich die Positionen des TCPs in diesem Koordinatensystem, ist es möglich das Anwenderkoordinatensystem zu verlagern, ohne dabei eine Neuprogrammierung der Position erforderlich zu machen. Durch eine Verlagerung des Anwenderkoordinatensystems in Bezug auf die Arbeitsfläche, verschieben sich alle programmierten Positionen um den gleichen Betrag mit. Dadurch kann von einer Neuprogrammierung der Position abgesehen werden. Der Koordinatenursprung des Anwenderkoordinatensystems wird im Weltkoordinatensystem definiert. [10]

Weltkoordinatensystem: Wird der Roboter stehend montiert, liegt das Weltkoordinatensystem, sofern nicht anders definiert, mit dem Basiskoordinatensystem übereinander. Bei einer Deckenmontage des Roboters stimmen jedoch die Richtungen der Achsen im Arbeitsraum nicht mehr zusammen. Durch die Drehung des Weltkoordinatensystems kann dieses Problem gelöst werden. Wenn mehrere Roboter in einem Arbeitsraum vorhanden sind, ist durch die Verwendung des Weltkoordinatensystems eine Interaktion mit den Roboterprogrammen möglich.[11]

[8] Vgl. ABB (Hrsg.) (2004-2016), S. 113 f.

[9] Vgl. ABB (Hrsg.) (2004-2016), S. 114.

[10] Vgl. ABB (Hrsg.) (2004-2016), S. 114 ff.

[11] Vgl. ABB (Hrsg.) (2004-2016), S. 114 f.

Der Zusammenhang der einzelnen Koordinatensysteme wird in der folgenden Abbildung dargestellt.

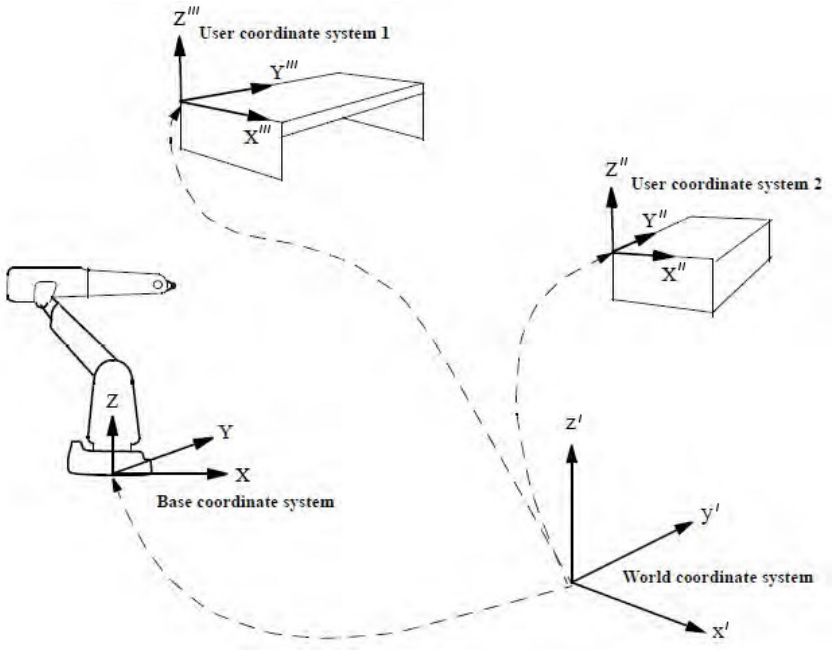

Abb. 9: Basiskoordinatensystem und Anwenderkoordinatensystem eines Roboters, Quelle: ABB (Hrsg.) (2004-2016), S. 116. (leicht modifiziert).

3.2 CNC Steuerungen

Auch Computerized Numerical Control (CNC) Maschinen finden ihre Verwendung in der Automatisierungstechnik. CNC Maschinen werden dabei mit einem Industrieroboter verbunden um den Belade und Entlade Zyklus zu beschleunigen. Im Gegensatz zu Roboter stellt man an eine CNC Maschine einen höheren Anspruch an die Genauigkeit. Obwohl der Aufbau völlig unterschiedlich ist, werden ebenfalls verschiedene Achsen mit unterschiedlichen Koordinatensystemen benötigt um auch bei CNC Maschinen Bahnsteuerungen zu realisieren.

Um nun einen Einblick in die CNC Thematik zu bekommen, werden in diesem Unterkapitel die Grundlagen der CNC Steuerungen behandelt.

3.2.1 Grundlagen

Der aus der amerikanischen Fachsprache übernommene Begriff CNC bedeutet computergestützte numerische Steuerung. Mit diesen Steuerungen werden vorwiegend Werkzeugmaschinen geregelt und gesteuert. Zentrale Aufgabe einer CNC Steuerung ist es, Bewegungsabläufe schnell und wiederholgenau zu realisieren. Dadurch ist gewährleistet, dass eine Massenproduktion mit gleichbleibender Qualität erfolgen kann. Um ein Werkstück mittels einer Maschine bearbeiten zu können, braucht diese Informationen. Bevor die Numerical Control (NC) Technik Einzug genommen hat kamen diese Informationen von den Maschinenbedienern oder durch mechanische Hilfsmittel wie zum Beispiel Kurvenscheiben oder Schablonen. Um auf ein anderes Produkt zu wechseln war es notwendig, diese mechanischen Hilfsmittel zu tauschen bzw. zu verändern. Des Weiteren waren auch die Werkzeuge auf das neue Produkt anzupassen. Diese Vorgänge hatten einen relativ langen Rüstprozess zur Folge. Durch den Einsatz von CNC Steuerungen in der „modernen“ Fertigungstechnik ist es möglich, Produktwechsel

rasch durchzuführen, da nur noch das Programm angepasst werden muss. Um Rüstzeiten zu minimieren, können die entsprechenden Werkzeuge bereits vorab in den Werkzeugwechsler eingebaut werden.[12]

3.2.2 Aufbau

Eine Achse alleine ergibt noch keine CNC Maschine. Deshalb ist es notwendig das richtige Zusammenspiel der einzelnen Antriebe sicher zu stellen. Je nach Komplexität der Anwendung kann eine CNC Maschine bis zu neun Achsen besitzen. In der folgenden Abbildung ist eine weit verbreitete fünf Achsen Maschine zu sehen.

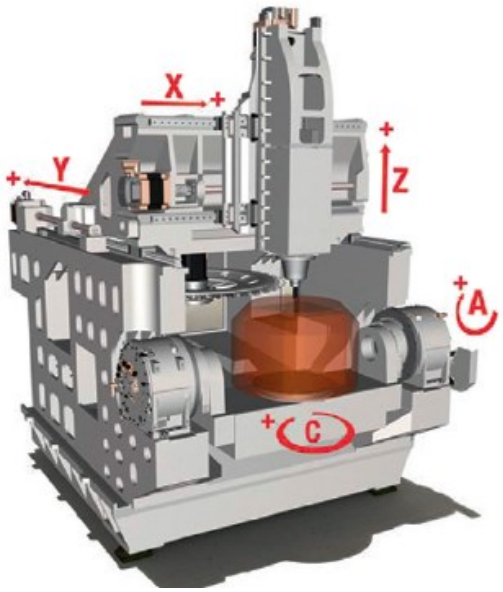

Abb. 10: Aufbau CNC Maschine, Quelle: Kief/Roschiwal/Schwarz (2017), S. 103.

So eine Maschine findet Einsatz für die Produktion von Teilen mit kleinerer oder mittlerer Stückzahl. Eine fünf Achsen CNC-Maschine ist die ideale Maschine für eine Lohnfertigung in Industriebetrieben wie zum Beispiel Automobilzulieferern oder für eine generelle Fertigung mit kleineren Stückzahlen. Je komplexer die Maschinen werden, desto höher wird der Automatisierungsgrad und umso mehr Durchsatz wird benötigt um eine kostendeckende Fertigung zu realisieren.[13]

Damit die NC-Achsen numerisch gesteuert werden können ist es notwendig, dass jede Achse ein elektronisch auswertbares Wegmesssystem und einen regelbaren Servoantrieb besitzt. Die CNC Steuerung kann somit, aufgrund der Abweichung auf den rückgelesenen Istwert auf die Sollwerte, den entsprechenden Stellwert vorgeben. Diese Stellwerte werden direkt auf den Achsantrieb geschrieben und man erhält dadurch einen geschlossenen Regelkreis. Bei Bahnsteuerungen werden laufend neue Positionsvorgaben gesendet. Dadurch erhält man kontinuierliche Bahnbewegungen. [14]

[12] Vgl. Kief/Roschiwal/Schwarz (2017), S. 39.

[13] Vgl. Kief/Roschiwal/Schwarz (2017), S. 53.

[14] Vgl. Kief/Roschiwal/Schwarz (2017), S. 84 ff.

In den folgenden Darstellungen wird das Zusammenspiel zwischen Antrieb und Wegmesssystem dargestellt.

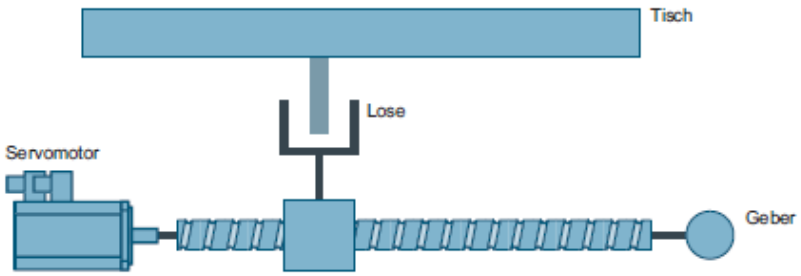

Abb. 11: Zusammenspiel zwischen Antrieb und Wegmesssystem, Quelle: Kief/Roschiwal/Schwarz (2017), S. 85.

Wie in dieser schematischen Abbildung ersichtlich, ist der Geber direkt mit der Spindel verbunden. Aufgrund der nicht formschlüssigen Kraftübertragung zwischen Werkzeugtisch und Antrieb kommt es zu einem Umkehrspiel. Dieses Umkehrspiel kann nur bedingt minimiert werden, da eine völlig spielfreie Verbindung einen zu hohen mechanischen Verschleiß mit sich bringen würde. Dieses Spiel kann jedoch ermittelt werden und als Maschinenparameter zur Kompensation eingestellt werden.[15]

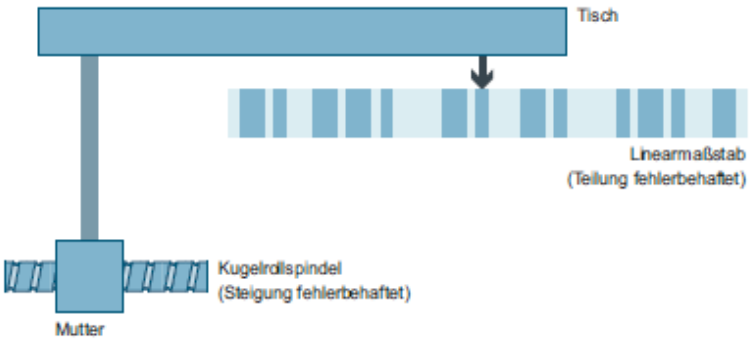

Abb. 12: Positionsmessung am Maschinenschlitten, Quelle: Kief/Roschiwal/Schwarz (2017), S. 87.

Eine wesentlich genauere Positionierung ist mit dieser Art von Messtechnik möglich, da direkt am Maschinenschlitten gemessen werden kann. Dadurch ist kein Spiel zu berücksichtigen und somit keine Kompensation nötig.[16]

3.2.3 Steuerung

CNC Maschinen werden mit Hilfe eines Mikrorechners und weiteren Steuerungsbaugruppen gesteuert. Ein weiterer Bestandteil ist die Bedientafel mit integriertem Display die eine Programmierung der Arbeitsabläufe ermöglicht. Bei der Programmierung kann auf verschiedene Steuerungsarten zurückgegriffen werden. Dabei kommen folgende Steuerungsarten wie zum Beispiel die Punktsteuerung, Streckensteuerung oder die Bahnsteuerung zum Einsatz.[17]

---

[15] Vgl. Kief/Roschiwal/Schwarz (2017), S. 84 ff.

[16] Vgl. Kief/Roschiwal/Schwarz (2017), S. 87.

[17] Vgl. Böge/Böge (Hrsg.) (2014), S. O 61.

In der folgenden Abbildung wird die Funktionsweise dieser Steuerungsarten aufgezeigt.

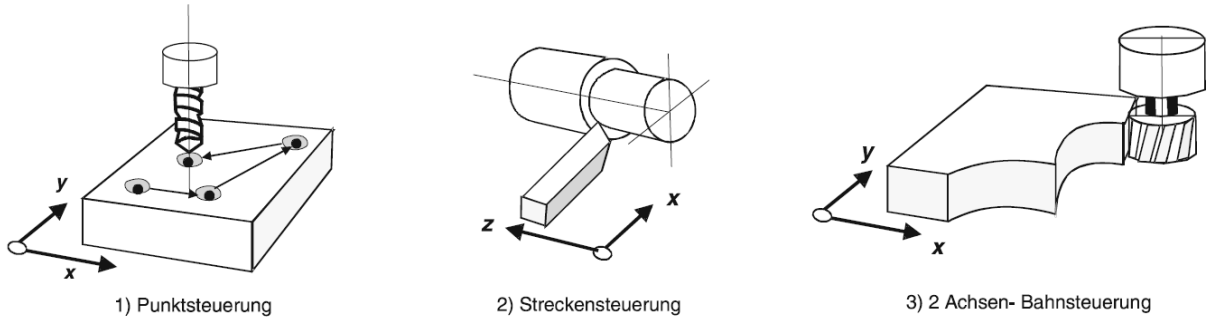

Abb. 13: Steuerungsarten verschiedener CNC Maschinen, Quelle: Böge/Böge (Hrsg.) (2014), S. O 61.(leicht modifiziert).

Mit diesen Informationen wird im nächsten Unterkapitel auf den jeweiligen Bahnsteuerungen von Industrieroboter und CNC Maschinen eingegangen.

3.3 Bahnsteuerung

Unterschiedliche Einsatzmöglichkeiten eines Industrieroboters setzen verschiedene Steuerungsprinzipien voraus. Roboter, Dreh- und Fräsmaschinen haben eine numerische Steuerung. Diese werden wie vorher erwähnt als CNC Maschinen bezeichnet. Numerische Maschinen arbeiten mit Zahlen oder Ziffern, die in die Steuerung eingetragen werden können.[18]

Gemeinsam haben diese Steuerungen, Punkte die in der Ebene, oder im Raum definiert werden. Eine Änderung dieser Punkte erfolgt über eine direkte Eingabe oder durch das Teach-in Verfahren. Somit werden Punkte generiert, und für die Verwendung im Roboterprogramm zur Verfügung gestellt. Das Programm bestimmt dabei, auf welche Art und in welcher Reihenfolge der Roboter diese Punkte anfährt. Durch die Verwendung von unterschiedlichen Befehlen wird das gewünschte Programm zeilenweise aufgebaut. CNC Maschinen unterschiedlicher Hersteller verwenden einheitliche Befehle. Industrieroboter unterscheiden sich aber meist durch ihre unterschiedlichen Programmiersprachen.[19]

Die Informationen hinter diesen Programmiersprachen sind jedoch dieselben. Ein Punkt in der Ebene besteht immer aus einer X- und Y-Koordinate und für einen Bewegungsablauf werden zwei Punkte benötigt. Es unterscheiden sich dabei die Befehle für die Bewegungen zwischen diesen Punkten.

[18] Vgl. Böge/Böge (Hrsg.) (2014), S. O 61.

[19] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 138.

Je nach Befehl wird ein unterschiedlicher Verfahrweg zwischen Startpunkt und Endpunkt ausgeführt. In den nachfolgenden Unterpunkten werden Standard Befehle einer CNC Steuerung und eines Industrieroboters erläutert und gegenübergestellt:[20]

- Bei einem Punkt zu Punkt (Point to Point, PtP) Befehl hängt die Bahn von der jeweiligen Robotersteuerung ab. Hierbei kann vom BedienerInnen nur der Startpunkt und der Endpunkt fixiert werden, zum Beispiel:
  - CNC Maschine: G00
  - Roboter Steuerung: PtP
- Soll der Verfahrweg eine Gerade bilden wird ein Linearbefehl benötigt. Dabei wird die kürzeste Verbindung zwischen Start- und Endpunkt abgefahren, zum Beispiel:
  - CNC Maschine: G01
  - Roboter Steuerung: Linear
- Weiters kann zwischen den Punkten ein Kreisbogen abgefahren werden. Die Parameter des Bogens werden dem Befehl über Zusatzinformation übergeben. Parameter sind z.B.: Kreisbogen im oder gegen den Uhrzeigersinn oder der Radius des Kreisbogens, zum Beispiel:
  - CNC Maschine: G02/G03
  - Roboter Steuerung: Circular

Bei einer Bahnsteuerung in einer Ebene werden zusätzlich noch weitere Befehle verwendet. Die Punkte der einzelnen Befehle werden hierzu in ein ebenes kartesisches Koordinatensystem gelegt.[21]

In dieser Arbeit geht es um die Änderung dieser einzelnen Koordinaten. Der Programmaufbau und die Befehle für die Bahnsteuerung bleiben dieselben. Mit diesem Konzept soll es möglich sein, ein Roboterprogramm ohne Kenntnisse der dahinterliegenden Programmiersprache zu ändern. (eine Intuitive Bedienung als Schlüsselfunktion des Software-Tools)

[20] Vgl. Bartenschlager/Hebel/Schmidt (1998), S. 138.

[21] Vgl. Hesse (2000), S. 50 f.

4 BEDIENKONZEPT

Das Bedienkonzept des Software-Tools beinhaltet den grafischen Aufbau der Bedienoberfläche, die Anordnung der Bedienelemente und die Funktionselemente und die Interaktion damit. Dabei wird auf eine mögliche Touch-Bedienung Rücksicht genommen. Bevor es zur Konzeptionierung der Bedienoberfläche kommt, werden bestehende Bedienkonzepte unterschiedlicher Roboterhersteller betrachtet. Dazu werden definierte Anforderungen an diese Konzepte gestellt. Die Vor- und Nachteile der betrachteten Bedienkonzepte sollen die Grundlage für die Bedienoberfläche und das Bedienkonzept des Software-Tools darstellen.

4.1 Bedienoberfläche bestehender Roboterpanel

Bedienoberflächen bestehender Roboterpanel sind herstellerspezifisch unterschiedlich aufgebaut. Roboterpanels werden zur Konfiguration der Robotersteuerung, für die Programmierung und für das Teach-In Verfahren verwendet. Es werden folgende Anforderungen an die Bedienoberfläche bestehender Roboterpanels gerichtet und näher betrachtet:

- Darstellung des Programmcodes
- Änderung von Positionen
- Darstellung von Bahnsegmenten
- Bezug zwischen Bauteil und Roboterbahn

In den folgenden Absätzen wird die Umsetzung von Bedienoberflächen mit den einzelnen Anforderungen von verschiedenen Roboterherstellen verglichen.

Darstellung des Programmcodes

Wie in den Abbildungen, Abb. 14, Abb. 15 und Abb. 16 gezeigt, wird herstellerunabhängig der Programmcode vollständig am Roboterpanel angezeigt. Es werden dabei alle Argumente dargestellt, die für ein Roboterprogramm benötigt werden. Für einen Laien in der Roboterprogramierung ist diese Darstellung jedoch verwirrend, denn es ist auf den ersten Blick nicht ersichtlich, welche Programmzeilen für eine Änderung einer Roboterbahn relevant sind.

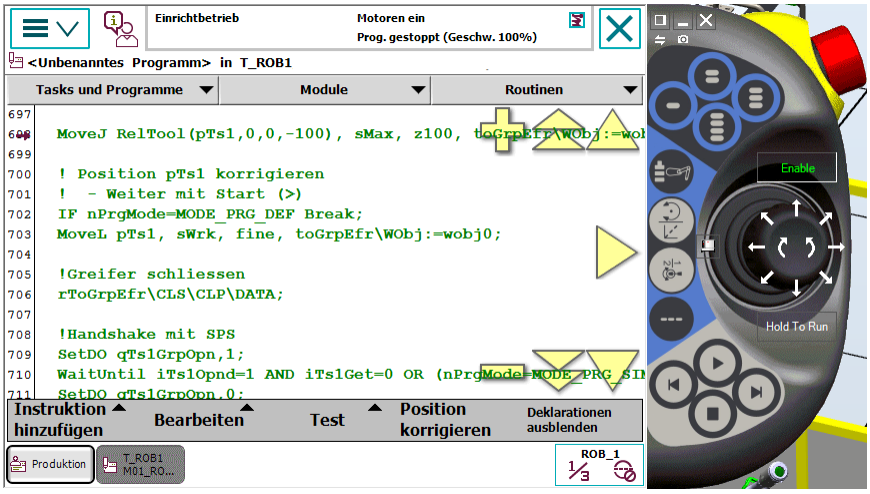

Abb. 14: Darstellung Programmcode ABB, Quelle: In Anlehnung an ABB Robot Studio.

Wie in der vorigen Abbildung des Roboterpanels der Firma ABB, ist auch beim Roboterhersteller Fanuc und Kuka, dieselbe Darstellung des Robotercodes ersichtlich. Dies zeigen die nachfolgenden zwei Abbildungen.

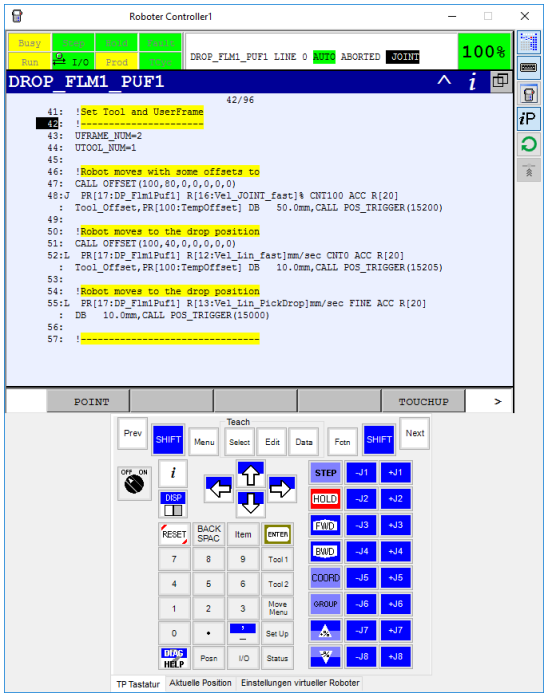

Abb. 15: Darstellung Programmcode Fanuc, Quelle: In Anlehnung an Fanuc ROBOGUIDE.

Abb. 16: Darstellung Programmcode Kuka, Quelle: In Anlehnung an Kuka WorkVisual.

Die Darstellung des Programmcodes soll im Software-Tool soweit angepasst werden, dass für die Änderung eines Punktes nur die dafür relevanten Daten ersichtlich sind. Es soll damit eine Bedienung für nicht geschultes Personal ermöglicht werden.

Programmstruktur

Für die Erstellung des Software-Tools ist es notwendig, Grundzüge aus der Roboterprogrammierung aufzuzeigen. Diese werden anhand des ABB Roboters und einer CNC Steuerung behandelt.

Betrachtung der Programmstruktur des ABB Roboters.

Instruktionen

Ein Roboterprogramm besteht aus eine Anzahl von Instruktionen, mithilfe dieser das Programm aufgebaut wird. Es gibt Instruktionen die für eine Roboterbewegung benötigt werden. Mit Argumenten können Instruktionen näher beschrieben werden.[22]

Routinen

Unterprogramme werden als Routinen bezeichnet und können in Prozeduren, Funktionen und Interrupts aufgeteilt werden. Prozeduren beinhalten keine Rückgabewerte, Funktionen geben einen Wert zurück. Mithilfe von Interrupt-Routinen kann das Programm unterbrochen werden und eine definierte Routine ausgeführt werden. Eine Interrupt-Routine wird nicht vom Programm aufgerufen, sondern von dem Betriebssystem des Roboters.[23]

Daten

Informationen welche zur Beschreibung eines Werkzeugs dienen, wie zum Beispiel das Gewicht eines Robotergreifers, werden als Werkzeugdaten bezeichnet. Dazu zählen auch numerische Daten. Werkzeugdaten und numerische Daten können als Daten im Robotersystem gespeichert werden. Daten unterscheiden sich durch Konstanten, Variablen und Persistenten. Konstanten können neue Werte nur manuell beschrieben werden und stellen damit einen statischen Wert dar. Im Unterschied zu Konstanten können Variablen auch während der Programmabarbeitung neue Werte zugewiesen werden. Um eine Variable speichernd zu beschreiben sind Persistenten zu verwenden. Beim Speichern eines Programmes wird der aktuelle Wert des Persistenten als Initialisierungswert dieses Datentyps überschrieben.[24]

Bewegungsinstruktionen

Bewegungen eines Roboters werden als Punkt zu Punkt Bewegung projektiert. Das Robotersystem berechnet dabei die Bahn zwischen diesen Punkten automatisch. Durch die Auswahl einer geeigneten Instruktion wird die Art dieser Bewegung bestimmt. Eine Auswahl dieser Positionierinstruktionen sind in der nächsten Abbildung dargestellt.[25]

[22] Vgl. ABB (Hrsg.) (2004-2016), S. 11.

[23] Vgl. ABB (Hrsg.) (2004-2016), S. 11.

[24] Vgl. ABB (Hrsg.) (2004-2016), S. 11 f.

[25] Vgl. ABB (Hrsg.) (2004-2016), S. 57.

Instruktion	Art der Bewegung
MoveC	Der TCP bewegt sich auf einer kreisförmigen Bahn.
MoveJ	Achsenbewegung.
MoveL	Der TCP bewegt sich auf einer linearen Bahn.
MoveAbsJ	Absolute Achsenbewegung.
MoveExtJ	Bewegt eine lineare oder drehende externe Achse ohne TCP.

Abb. 17: Positionierungsinstruktionen, Quelle: ABB (Hrsg.) (2004-2016), S. 57.

Diese Bewegungsinstruktionen benötigen bestimmte Bewegungseigenschaften welche als Daten definiert und als Argumente bezeichnet werden. Diese Argumente beinhalten:[26]

- Positionsdaten
- Geschwindigkeitsdaten
- Positionsgenauigkeit
- Werkzeugdaten
- Aktuelles Koordinatensystem

Übergeordnet können zusätzliche Bewegungseigenschaften definiert werden, welche für das gesamte Roboterprogramm gelten. Diese beinhalten wiederum:[27]

- Geschwindigkeitsgrenzwerte
- Beschleunigungen
- Nutzlast
- Organisation unterschiedlicher Roboterkonfigurationen

In den nachfolgenden Aufzählungen werden die zwei häufigsten Instruktionen inklusive ihrer Argumente näher betrachtet.

MoveL[28]

Die Bewegungsinstruktion lineares Bewegen (Move linear, MoveL) bewegt den Werkzeugarbeitspunkt eines Roboters linear zur Zielposition. Beispiel der Instruktion MoveL mit verwendeten Argumenten:

MoveL p1, v1000, z30, tool1;

Bei dieser Bewegung bewegt sich der Werkzeugarbeitspunkt des Werkzeuges tool1 linear zur Position p1. Die Geschwindigkeit v beträgt dabei 1000 und als Zonendaten z ist 30 definiert.

[26] Vgl. ABB (Hrsg.) (2004-2016), S. 57.

[27] Vgl. ABB (Hrsg.) (2004-2016), S. 57.

[28] Vgl. ABB (Hrsg.) (2004-2016), S. 451 f.

MoveC[29]

Die Instruktion kreisförmige Bewegung (Move circular, MoveC) wird für eine kreisförmige Bewegung des Roboters verwendet. Beispiel der Instruktion MoveC mit verwendeten Argumenten:

MoveC p1, p2, v400, z20, tool2;

Der Befehl MoveC wird meist in Verbindung mit dem Befehl MoveL verwendet. Diese beiden Instruktionen werden folgendermaßen eingesetzt:

MoveL p1, v400, z30, tool2;

MoveC p2, p3, v400, z20, tool2;

Mit dem Befehl MoveL bewegt sich der Werkzeugarbeitspunkt linear zur Position p1, welche zugleich als Startposition für die MoveC Instruktion verwendet wird. Der Kreisbogen wird dabei durch die Startposition p1, der Stützposition p2 und die Zielposition p3 definiert. Die nachfolgende Abbildung zeigt diesen Bewegungsablauf.

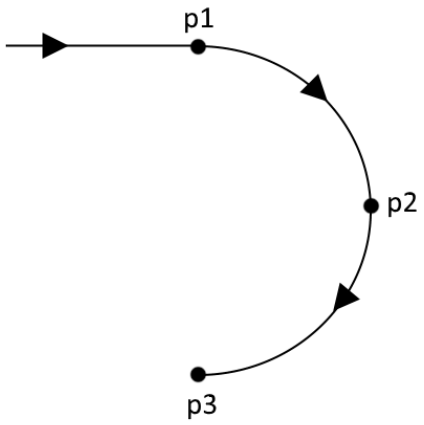

Abb. 18: Ausführung einer MoveL und MoveC Instruktion, Quelle: Eigene Darstellung.

Argumente der Bewegungsinstruktionen MoveL und MoveC[30]

MoveL ToPoint, Speed, Zone, Tool [\WObj];

MoveC CirPoint, ToPoint, Speed, Zone, Tool [\WObj];

ToPoint: Mit diesem Argument wird die Zielposition angegeben, welche als Datentyp „robtarget“ definiert ist.

CirPoint: Der CirPoint hat den Datentyp „robtarget“ und wird als Bogenpunkt bezeichnet.[31]

[29] Vgl. ABB (Hrsg.) (2004-2016), S. 392 f.

[30] Vgl. ABB (Hrsg.) (2004-2016), S. 451 ff.

[31] Vgl. ABB (Hrsg.) (2004-2016), S. 393.

Speed: Im Argument Speed mit dem Datentyp „speeddata“, werden die Geschwindigkeitsdaten für die einzelnen Instruktionen gespeichert. Die Geschwindigkeit wird dabei in mm/s angegeben.

Zone: Zone steht für die jeweiligen Zonendaten der Bewegung.

Tool: Das Argument Tool benötigt eine Variable mit Datentyp „tooldata“, welche die Information des gewünschten Werkzeuges des Roboters beinhaltet.

Zusätzlich kann dieses Argument mit einem Werkobjekt erweitert werden. Es beschreibt das Koordinatensystem auf das sich die Zielposition bezieht. Für die Verwendung wird eine Variable mit Datentyp „wobjdata“ benötigt.

Betrachtung der Programmstruktur einer CNC Steuerung.

Mit Hilfe der Programmiersprache G-Code, welche in der CNC Steuerung eingesetzt wird lässt sich ein Industrieroboter ebenfalls programmieren.

Dieses NC Programm setzt sich aus einer Reihe von Anweisungen zusammen, die von einer NC Maschine abgearbeitet werden. Bei der Betrachtung einer CNC Maschine definieren diese Anweisungen die Herstellung eines Werkstückes. Dieses wird durch Relativbewegungen zwischen Werkstück und Werkzeug hergestellt. Die dabei verwendeten Maßeinheiten werden in mm oder in zoll angegeben. Weiters beinhaltet ein NC Programm zusätzlich zu den Bahninformationen alle Schaltinformationen und Hilfsbefehle, die für eine automatische Herstellung eines Werkstückes benötigt werden.[32]

In Abb. 19 ist der Aufbau eines NC Programmes dargestellt. Die Anzahl der Sätze kann dabei beliebig gewählt werden. Diese Sätze spiegeln den gesamten Ablauf der Maschine wider und werden als eine Zeile im Programm definiert. Eine Nummerierung der einzelnen Zeilen erleichtert dabei die Fehlersuche. Ein einziger Satz beinhaltet wiederum Wörter die aus einer Adresse und einem Zahlenwert bestehen. Die Adresse definiert dabei die Zieladresse für die nachfolgenden Zahlenwerte. Ein wiederholtes Vorkommen einer Adresse in einem Satz ist jedoch nicht zulässig.[33]

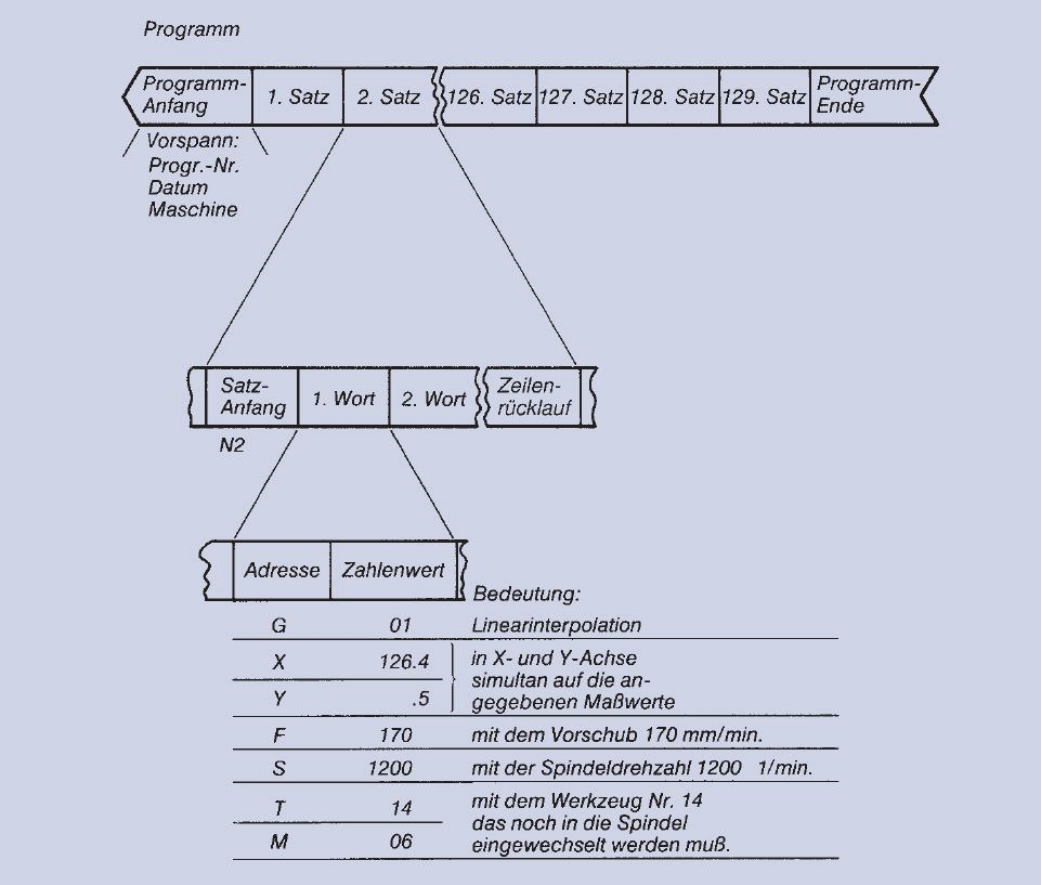

Abb. 19: Aufbau eines NC Programmes, Quelle: Kief/Roschiwal/Schwarz (2017), S. 567.

[32] Vgl. Kief/Roschiwal/Schwarz (2017), S. 566 ff.

[33] Vgl. Kief/Roschiwal/Schwarz (2017), S. 566 ff.

Bei der Eingabe von Zahlenwerten ist eine Dezimalpunkt-Schreibweise anzuwenden. Diese Zahl definiert die anzufahrende Position der Werkzeugmaschine. Die Anweisungen die in einem Satz vorkommen, werden folgendermaßen unterschieden:[34]

- **Fahranweisungen**: Eine Fahranweisung beschreibt die Art einer Bewegung, wie zum Beispiel eine Linearinterpolation, oder eine Zirkularinterpolation.
- **Geometrische Anweisungen**: Mit geometrischen Anweisungen wird zwischen Werkzeug und Werkstück eine Relativbewegung durchgeführt. Adressen können zum Beispiel X, Y, Z, … sein.
- **Schaltbefehle**: Schaltbefehle werden für die Auswahl von Werkzeugen, Schalttischeinstellungen und Kühlmittelzuführung benötigt.

Syntax und Semantik

Grammatikregeln in einem Programmaufbau werden als Syntax bezeichnet. Dabei wird nicht auf die Bedeutung der einzelnen Wörter Rücksicht genommen. Die Semantik beschreibt die Bedeutung der Wörter, welche wiederum zusammen mit der Syntax den Programmcode ergeben. Ein Satz für eine zwei Achs Bahnsteuerung wäre zum Beispiel, **N**1 **G**2 **X**5.4 **Y**3.4 **I**4.3 **J**4.3 und hat die Bedeutung:[35]

- **N1**: Mit einem N und einer bis zu vierstelligen Nummer können einzelne Sätze mit einer Satznummer versehen werden.
- **G2**: Die Fahranweisung G2 beschreibt eine Bewegung mit einer Zirkularinterpolation.
- **X, Y**: Diese geometrischen Anweisungen geben die Zielposition der Bewegung an
- **I, J**: Bei einer Kreisinterpolation werden sogenannte Hilfsparameter benötigt die mit I und J eingetragen werden.

Wegbedingungen

Für den praktischen Teil dieser Arbeit werden insbesondere Wegbedingungen benötigt. Wegbedingungen werden als G Funktionen Bezeichnet. Das G steht dabei für *gehe* und gibt der Steuerung vor wie sie die Nachfolgenden Wegbefehle abarbeiten soll. *Wohin* gibt die Weginformation vor und *wie* gibt die Wegbedingung vor.[36]

In der nachfolgenden Abbildung sind drei Wegbedingungen abgebildet, die bei der Erstellung des Software Prototyps zum Einsatz kommen.

Code	Funktion
G01	Lineare Interpolation
G02	Kreisinterpolation, im Uhrzeigersinn
G03	Kreisinterpolation, gegen Uhrzeigersinn

Abb. 20: G-Code Wegbedingungen, Quelle: Kief/Roschiwal/Schwarz (2017), S. 575. (leicht modifiziert).

[34] Vgl. Kief/Roschiwal/Schwarz (2017), S. 568.

[35] Vgl. Kief/Roschiwal/Schwarz (2017), S. 570.

[36] Vgl. Kief/Roschiwal/Schwarz (2017), S. 574.

Änderung von Positionen

Das Ändern von Positionen kann mit dem Teach-In Verfahren durchgeführt werden. Es ist jedoch auch möglich, die Positionen über die Positionsdatenansicht zu ändern. In der Abb. 21 wird das Programmdatenfenster der Firma ABB gezeigt. Es wird eine Liste der einzelnen Punkte erstellt und dem BedienerInnen angezeigt. Über das Menüfeld „Position korrigieren“ können diese durch Eingabe von Koordinaten geändert und abgespeichert werden.

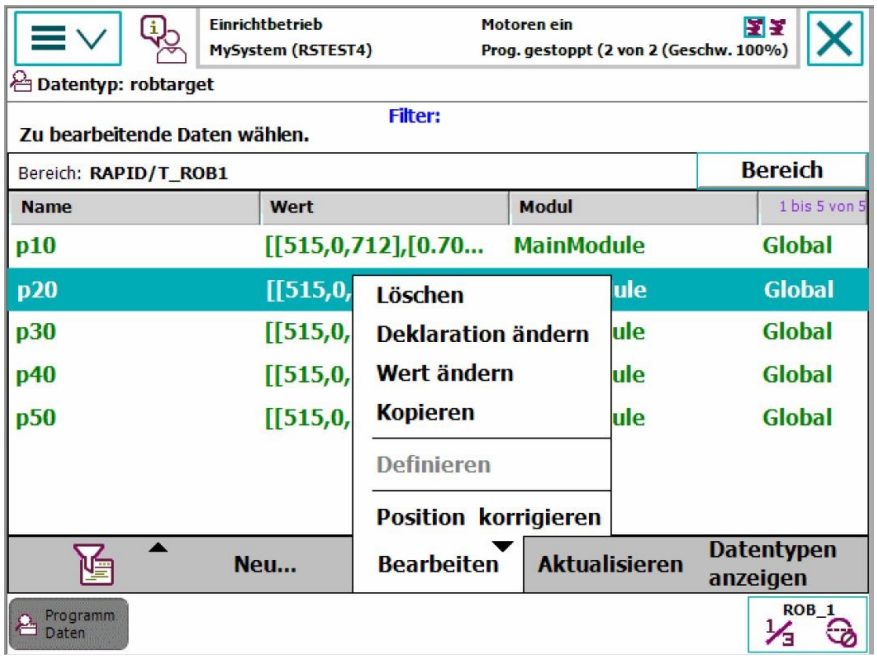

Abb. 21: Programmdatenansicht ABB Flex Pendant, Quelle: ABB (Hrsg.) (2016), S. 180.

Für die Änderungen einer Position wird zusätzlich ein eigenes Fenster aufgerufen. Wie in Abb. 22 gezeigt, beinhaltet die Position p10 die Koordinaten X, Y und Z welche in dem Fenster angepasst werden können.

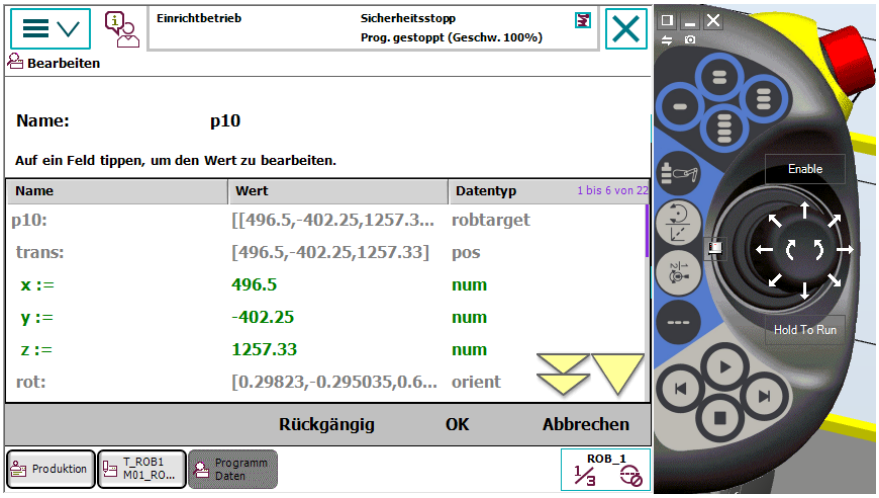

Abb. 22: Erweiterte Programmdatenansicht ABB Flex Pendant, Quelle: In Anlehnung an ABB Robot Studio.

Darstellung von Bahnsegmenten

Mit dieser Darstellung der Position erhält man jedoch keine genaue Übersicht, wo sich diese Positionen auf einer Roboterbahn befinden. Durch kostenpflichtige Zusatzpakete ist es jedoch möglich, für spezielle Aufgaben eine graphische Anzeige am Roboterpanel zu erhalten. Die Erstellung dieser Oberflächen muss jedoch ebenfalls durch ein geschultes Personal durchführt werden und sind nur für den jeweiligen Robotertyp einsetzbar.

Das Zusatzpaket RobotWare Machining FC von dem Unternehmen ABB beinhaltet die Möglichkeit über das Roboter Panel Roboterbahnen anzulegen.

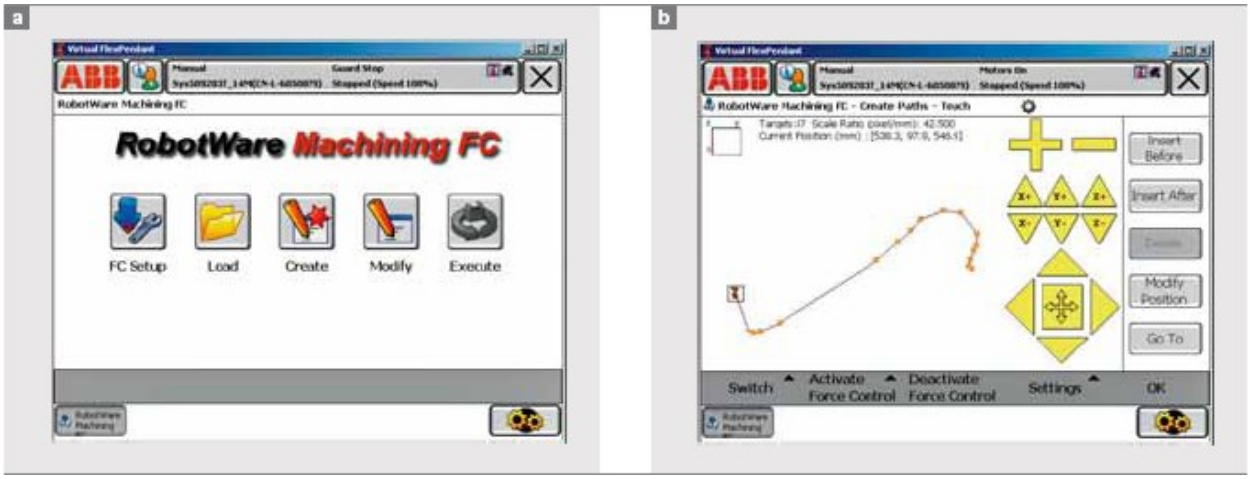

Abb. 23: Darstellung von Bahnsegmenten mit RobotWare Machining FC, Quelle: Fritsch (2017), Online-Quelle [13.Oktober.2017].

Grundsätzlich können Positionen am Roboterpanel geändert werden. Durch das Software-Tool soll die Änderung von Positionen und die Darstellung von Bahnsegmenten durch eine graphische Oberfläche ermöglicht werden.

Bezug zwischen Bauteil und Roboterbahn

Auf einer Robotersteuerung wird das Bezugssystem wie im Abschnitt 3.1.5 dargestellt, über das Anwenderkoordinatensystem definiert. Dieser Zusammenhang wird jedoch am Roboterpanel nicht grafisch dargestellt.

In den nachfolgenden Unterkapiteln wird der Aufbau der Bedienoberfläche und der Bedienelemente ausgearbeitet, in dem die Erkenntnisse aus diesem Unterkapitel einfließen.

4.2 Bedienoberfläche

Die Bedienoberfläche soll in zwei Bereiche unterteilt werden. Im linken Bereich werden die Bedienelemente und im rechten Bereich die grafische Bahndarstellung platziert. An den bisher betrachteten Bedienoberflächen ist zu bemerken, dass sich diese aus vielen verschachtelten Oberflächen zusammensetzt. Dadurch lässt sich eine gute Struktur erkennen, jedoch werden für eine Änderung von Position mehrere unterschiedliche Seiten benötigt. Im Software-Tool soll die gleichzeitige Ansicht der Roboterbahn und die zugleich durchführbare Adaptierung in einem Anzeigefenster untergebracht werden.

4.3 Bedienelemente Steuerelemente

Im Bereich der Bedienelemente sollen folgende Steuerelemente zur Verfügung stehen:

- **Einstellungen:** Unter diesem Element soll der Pfad für die Roboterdatei und der Hintergrundbilddatei eingestellt werden.
- **Konfiguration speichern unter / wählen:** Für die getätigten Änderungen am Hintergrundbildes soll eine eigene Konfigurationsdatei abgelegt werden. Somit kann bei einer erneuten Anpassung des Bahnsegmentes die Einstellungen des Hintergrundbildes aufgerufen werden.
- **Neu laden:** Mit dieser Funktion sollen die ausgewählten Dateien neu geladen werden. Damit können etwaige Änderungen rückgängig gemacht werden.
- **Speichern:** Um die adaptierten Punkte in den Robotercode abzuspeichern, soll die Funktion Speichern zur Verfügung stehen.
- **Grafik rotieren / skalieren:** Die Funktionen rotieren und skalieren sollen über Checkboxen angewählt und abgewählt werden können. Zusätzlich soll ein Eingabefeld für die Skalierung des Hintergrundbildes angezeigt werden.
- **Tabelle:** Mit einer Tabellenansicht sollen die eingelesenen Punkte angezeigt werden.
- **Anzeige der Roboterbahn:** Die dargestellten Punkte sollen sich visuell von der Bahn abheben und per Drag and Drop veränderbar sein. Ebenfalls soll die Anwahl eines Punktes eine Änderung über Verschiebbalken zulassen.
- **Verschiebalken:** Mit diesen Steuerelementen soll eine Position über die Änderung der X und Y Komponente ermöglicht werden. Abhängig vom ausgewählten Punkt, soll auch eine Adaptierung des Radius durchführbar sein.

5 ENTWICKLUNGSUMGEBUNG

Für die Erstellung des Software-Tools wurde die Software Visual Studio® 2017 gewählt. Visual Studio ist eine Entwicklungsumgebung für Windowsanwendungen, welche auf der Plattform Microsoft .NET beruht. Diese integrierte Entwicklungsumgebung bietet alle benötigten Tools für die Erstellung einer Applikation. Unter der Software Visual Studio können mehrere objektorientierte Programmiersprachen kombiniert eingesetzt werden.[37]

Objektorientierte Programmierung

Für das Verständnis der objektorientierten Programmierung sind einige Grundbegriffe notwendig. Diese Grundbegriffe werden in den folgenden Aufzählungen erörtert:[38]

- **Objekt:** Unter einem Objekt versteht man die Zusammenfassung von Daten und deren Funktionalitäten. Es werden dabei Gruppen von Eigenschaften, Ereignissen und Methoden gebildet. Bei einem Objekt werden die Eigenschaften und Methoden verändert, um auf dessen Ereignisse reagieren zu können.
- **Klasse:** Eine Klasse ist die Beschreibung eines Objektes. Es werden die Eigenschaften, Ereignisse und Methoden festgelegt.
- **Instanz:** Bei der Instanziierung wird eine Instanz einer Klasse gebildet. Die Eigenschaften der so erzeugten Objekte können dabei unterschiedlich sein. Eine wiederholte Instanziierung einer Klasse ist möglich.
- **Kapselung:** Eine Kapselung wird als eine streng definierte Schnittstelle zwischen dem inneren und äußeren Code einer Klasse bezeichnet. Dadurch entsteht eine saubere Trennung der Schnittstelle.
- **Wiederverwendbarkeit:** Klassen können an verschiedenen Positionen im Programmcode verwendet werden. Sie verringern somit den redundanten Teil des Codes. So können Fehler im Programmcode, an nur einer Stelle behoben werden.
- **Vererbung:** Bei der Vererbung wird eine Klasse von einer anderen Klasse abgeleitet. Dabei werden alle Inhalte der Basis Klasse zur Gänze übernommen, und stehen als Grundlage der neuen Klasse zur Verfügung. Die somit geerbten Eigenschaften können erweitert, geändert und überschrieben werden.
- **Polymorphie:** Polymorphie beschreibt die Möglichkeit Methoden und Eigenschaften in abgeleiteten Klassen völlig unterschiedlich zu implementieren. Die Verwendung in der Basisklasse erfolgt über die gleichen Namen, es werden aber die unterschiedlichen Implementierungen ausgeführt.

[37] Vgl. Doberenz/Gewinnus (2013), S. 68.

[38] Vgl. Doberenz/Gewinnus (2013), S. 157 ff.

Die Erstellung eines Programmes kann in vier Etappen unterteilt werden:[39]

- **Grafisches Konzept der Bedienoberfläche**: Hier werden im Startformular Schaltflächen, Eingabefenster oder Ausgabefenster definiert, welche hier ihre Position und Abmessungen erhalten.
- **Objekteigenschaften definieren**: In der zweiten Etappe werden die Eigenschaften dieser verwendeten Elemente definiert.
- **Objekte mit Ereignissen verbinden**: Hierbei wird festgelegt wie die unterschiedlichen Steuerelemente auf Eingaben oder Ereignisse reagieren.
- **Übersetzen des Quellcodes und Test des Programmes**: In dieser Etappe wird das erstellte Programm vom Compiler kompiliert. Dabei entsteht ein Zwischencode vom geschriebenen Programm, welcher auf jedem Rechner läuft. Voraussetzung ist hierbei, dass das .NET Framework installiert ist.

C#

Die Programmiersprache C# ist für das .NET Framework entwickelt worden. Diese Sprache verbindet die Vorteile aus den Programmiersprachen C++, Visual Basic, JavaScript und Java. Die verwendeten .NET Klassenbibliotheken sind zu einem großen Teil in C# programmiert. C# wird daher zur .NET Systemsprache. Für Windows, ist C++ die Systemsprache. Durch den Einsatz von C# lässt sich eine Anwendung objektorientiert programmieren und erhält somit einen strukturierten Aufbau.[40]

.NET

Unabhängig von der Wahl der Programmiersprache, wird der erstellte Programmcode in den Microsoft Intermediate Language Code (MSIL-Code) übersetzt. Dieser MSIL-Code ist Plattform und Hardware unabhängig und besitzt keine Information mehr, aus welcher Programmiersprache dieser kompiliert wurde. Beim Ausführen dieses Programmcodes wird er vom Just in Time Compiler (JIT-Compiler) in den Maschinencode übersetzt. Das .NET Programm wird bis zur Ausführung am gewünschten Rechner zweimal kompiliert. Für die Ausführung des Programmes wird der MSIL-Code und das installierte .NET Framework benötigt.[41]

[39] Vgl. Doberenz/Gewinnus (2013), S. 67 f.

[40] Vgl. Doberenz/Gewinnus (2013), S. 73.

[41] Vgl. Doberenz/Gewinnus (2013), S. 74.

Das .NET-Framework wird als Infrastruktur für die .NET-Plattform bezeichnet. Es besteht im Wesentlichen aus der Laufzeitumgebung (die Common Language Runtime (CLR)) und einer sehr umfangreichen Klassenbibliothek. Um die unzähligen Klassen logisch zu strukturieren gibt es Namespaces. Dieses Konzept ermöglicht es Klassen mit dem selben Namen in unterschiedlichen Namespaces unterzubringen. In der folgenden Abbildung werden die Komponenten des .NET-Frameworks dargestellt.[42]

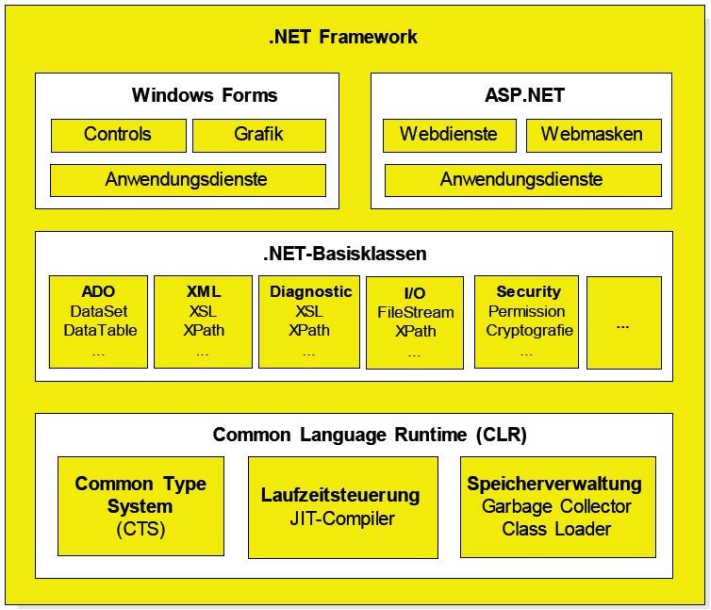

Abb. 24: .NET-Framework Komponenten, Quelle: Doberenz/Gewinnus (2013), S. 75.

Namespaces des .NET-Frameworks

Namespaces fassen alle Typen des .NET-Frameworks zusammen. Klassen bilden dabei Gruppen von Anwendungsgebieten. Die wichtigsten Namespaces werden in der folgenden Abbildung dargestellt.[43]

Namespace	... enthält Klassen für ...
System.Windows.Forms	... Windows-basierte Anwendungen
System.Collections	... Objekt-Arrays
System.Drawing	... die Grafikprogrammierung
System.Data	... den ADO-Datenbankzugriff
System.Web	... die HTTP-Webprogrammierung
System.IO	... Ein- und Ausgabeoperationen

Abb. 25: Namespaces des .NET-Frameworks, Quelle: Doberenz/Gewinnus (2013), S. 78.

Zusätzlich wird der Namespace System.NET für die FTP Kommunikation verwendet, welche im Unterkapitel 5.4 behandelt wird. Für das Speichern anfallender Konfigurationsdaten, wird der Namespace *System.Runtime.Serialzation* verwendet, welche im Unterkapitel 5.5 zum Einsatz kommt.

[42] Vgl. Doberenz/Gewinnus (2013), S. 75.

[43] Vgl. Doberenz/Gewinnus (2013), S. 78.

5.1 Draw Methoden

Die Projektierung des Software-Tools basiert auf den Standardklassen des .NET Framework. Durch die Verwendung von bereits inkludierten Standardklassen kann die Programmierung zeitreduziert erfolgen. Die Darstellung der Positionen einer Roboterbahn und die Anzeige des Skalierungsbildes im Hintergrund wird mit Hilfe von Klassen aus dem Namespace System.Drawing realisiert. Im folgenden Abschnitt wird auf die einzelnen Methoden eingegangen, die für die grafische Ausgabe benötigt werden.

Die Roboterbahn soll im Ausgabefenster angezeigt werden, welche sich in einem Koordinatensystem befindet. Das System.Drawing-Koordinatensystem hat dabei seinen Koordinatenursprung im linken oberen Eck des Ausgabefensters. Die X-Achse läuft nach rechts und die Y-Achse läuft nach unten ins positive wie in der nächsten Abbildung dargestellt.[44]

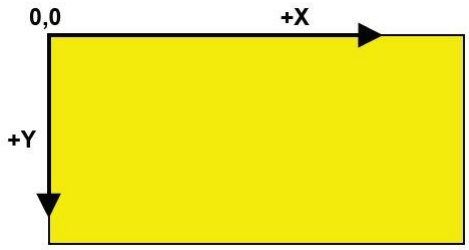

Abb. 26: .NET-Koordinatensystem, Quelle: Doberenz/Gewinnus (2013), S. 1624.

Standardmäßig wird für die Ausgabe die Maßeinheit Pixel verwendet. Dies kann jedoch mit der Eigenschaft *PageUnit* geändert werden.[45]

DrawLine

Das Zeichnen der Linien zwischen den einzelnen Punkten soll mit der *DrawLine* Methode erfolgen. Es werden hierfür zwei Punkte benötigt, die jeweils eine X und Y Komponente beinhalten müssen. Diese Übergabeparameter können als Koordinaten, oder als Struktur *Point* verwendet werden welche diese vordefinierte Struktur als Integer Zahl enthält. Durch die Struktur *PointF* können Gleitkommawerte dargestellt werden. Wie in Abb. 27 dargestellt verbindet die Methode *DrawLine* zwei Positionspaare.[46]

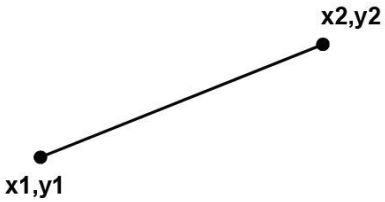

Abb. 27: Darstellung der DrawLine Methode, Quelle: Doberenz/Gewinnus (2013), S. 1633.

[44] Vgl. Doberenz/Gewinnus (2013), S. 1624.

[45] Vgl. Doberenz/Gewinnus (2013), S. 1624.

[46] Vgl. Doberenz/Gewinnus (2013), S. 1633.

DrawLines

Die Darstellung von zusammenhängenden Linien kann mit der Methode *DrawLines* realisiert werden. Dabei wird der Endpunkt als neuer Startpunkt der darauffolgenden Linie definiert. Dabei entsteht eine Verbindung von mehreren Punkten. Als Übergabewert wird ein Array von Punkten benötigt. Eine Realisierung dieser Methode wird in der nachfolgenden Abb. 28 gezeigt.[47]

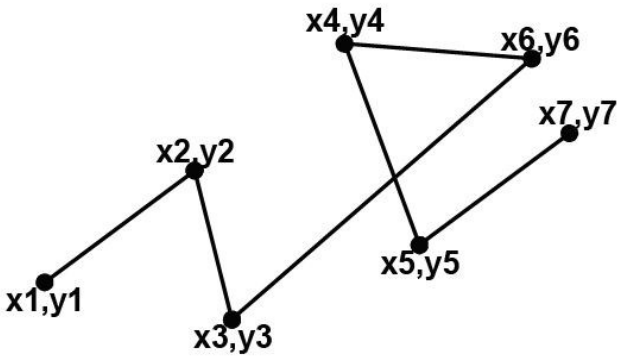

Abb. 28: Darstellung der DrawLines Methode, Quelle: Doberenz/Gewinnus (2013), S. 1635.

DrawArc

Für die Darstellung eines Bogenstückes soll die Methode *DrawArc* verwendet werden. Diese Methode arbeitet mit den Übergabeparametern Rechteck, dem Startwinkel und dem Bogenwinkel. Um einen Ellipsenbogen zu zeichnen wird ein Rechteck benötigt. Für einen Kreis sind die Seitenlängen des Rechtecks so zu definieren das ein Quadrat entsteht. Mit der Struktur *Rectangle* kann dieses Rechteck angelegt werden. Die Struktur beinhaltet den Startpunkt X, Y, die Höhe und die Breite des Rechtecks.[48]

Der Mittelpunkt des Kreises ist dabei der Mittelpunkt des Rechtecks. Das Rechteck des Kreisbogens und die Parameter Startwinkel und Bogenwinkel sind wie in Abb. 29 definiert.[49]

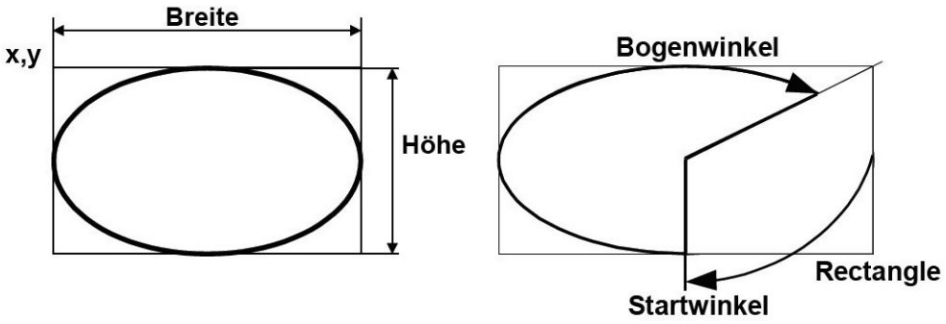

Abb. 29: Darstellung der DrawArc Methode, Quelle; Doberenz/Gewinnus (2013), S. 1640 f.

[47] Vgl. Doberenz/Gewinnus (2013), S. 1635.

[48] Vgl. Doberenz/Gewinnus (2013), S. 1635 f.

[49] Vgl. Doberenz/Gewinnus (2013), S. 1642.

DrawImage

Durch die Methode *DrawImage* kann eine Grafik ins Ausgabefenster geladen werden. Diese Funktion soll für das Einblenden des Hintergrundes verwendet werden. Die Übergabeparameter X und Y stellen dabei die Position dar, an der das Bild in dem Ausgabefenster angezeigt wird.[50]

Pen

Bei der Verwendung von Grafikmethoden wird ein initialisiertes Pen-Objekt benötigt, welches für die Darstellung von Linien verwendet wird. Das Pen-Objekt beinhaltet Informationen der darzustellenden Linie, wie zum Beispiel die Farbe, die Linienenden und die Dicke.[51]

5.2 Geometrische Transformation in C#

Es gibt zwei Arten der Transformation, welche für die Anpassung der Punkte in einem Koordinatensystem zur Verfügung stehen. Bei einer Koordinatentransformation wird das Koordinatensystem verändert, wobei bei einer geometrischen Transformation die Punkte innerhalb ihres Koordinatensystems verändert werden. Funktionen, wie Verschiebung, Skalierungen und Drehungen werden zur Veränderungen dieser Punkte verwendet.[52]

Um Anpassungen der gesamten Bahn und des Hintergrundbildes zu ermöglichen, werden diese Funktionen in die Software integriert. Die geometrische Transformation bezieht sich nicht auf die interne Transformation des Roboters, sondern auf die extrahierten Punkte der Bahnsteuerung und auf die Anpassung des Hintergrundbildes. In diesem Abschnitt werden diese Funktionen genauer betrachtet.

5.2.1 Allgemeiner Aufbau der Matrix-Klasse

Für die Transformation der Anzeigeobjekte soll die Matrix-Klasse zum Einsatz kommen. Die Matrix-Klasse wird vom Namespace System.Drawing.Drawing2D bereitgestellt und beinhaltet intern eine 3x3 Matrix. Es kann jedoch nur auf die ersten beiden Spalten zugegriffen werden. Die Werte der dritten Spalte sind Konstante und können nicht verändert werden. Diese Matrix wird in der nächsten Gleichung dargestellt.[53]

$$\begin{vmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ dx & dy & 1 \end{vmatrix} \tag{5.1}$$

Die Matrix-Klasse erlaubt eine sogenannte affine Transformation, welche durch die Aneinanderreihung einer linearen Transformation mit einer Verschiebung definiert wird. Eine lineare Transformation besteht

[50] Vgl. Doberenz/Gewinnus (2013), S. 1648 f.

[51] Vgl. Doberenz/Gewinnus (2013), S. 1654.

[52] Vgl. Fetzer/Fränkel (2012), S. 294.

[53] Vgl. Doberenz/Gewinnus (2013), S. 1745.

aus einer Multiplikation mit einer 2x2 Matrix und eine Verschiebung aus einer Addition einer 1x2 Matrix. Dieses Matrizenpaar wird in der Matrix-Klasse als 3x3 Matrix abgespeichert. Der lineare Teil beinhaltet die Variablen m11, m12, m21 und m22, während der Verschiebungsteil aus den Variablen dx und dy besteht, wie in Gleichung 5.2 und Abb. 30 ersichtlich ist. Die Variablen m11 und m22 werden für die Skalierung eines Elementes benötigt. Mit den Variablen m12 und m21 alleine wird eine Scherung eines Elementes ermöglicht, jedoch wird diese Transformationsart in dieser Arbeit nicht umgesetzt.[54]

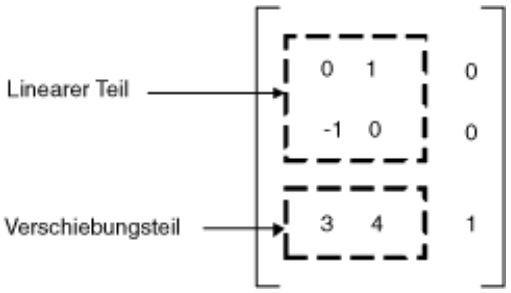

Abb. 30: Einteilung der Matrix-Klasse, Quelle: Microsoft (2017), Online-Quelle [17.Oktober.2017].

5.2.2 Verschiebung

Soll eine Verschiebung eines Elementes in der Grafikanzeige realisiert werden, kann dies über die Methode *Translate* der Matrix-Klasse erzielt werden. Die Übergabeparameter sind dabei dx und dy, welche sich in der folgenden Transformationsmatrix für eine Verschiebung wiederfinden.[55]

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{vmatrix} \tag{5.2}$$

In Abb. 31 wird eine initialisierte Matrix und die Methode *Translate* der Matrix-Klasse dargestellt. Die Übergabeparameter 20 und 10 bedeuten eine Verschiebung von 20 Einheiten in Richtung X und eine Verschiebung von 10 Einheiten in Richtung Y.

```
Matrix m = new Matrix();
m.Translate(20, 10);
```

Abb. 31: Code Beispiel der Transform Methode, Quelle: Eigene Darstellung.

[54] Vgl. Microsoft (2017), Online-Quelle [17.Oktober.2017]

[55] Vgl. Doberenz/Gewinnus (2013), S. 1746.

5.2.3 Skalierungen

Die Skalierung eines Elementes kann über die Methode *Scale* abgebildet werden. Mit dem Übergabeparameter m11 wird das Element in die X Richtung skaliert. M22 beschreibt die Skalierung in Y Richtung wie in Gleichung 5.3 ersichtlich ist.[56]

$$\begin{vmatrix} m11 & 0 & 0 \\ 0 & m22 & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad (5.3)$$

Für eine Skalierung bei dem das Grafikobjekt um das doppelte vergrößert werden soll, werden folgende Programmteile verwendet.[57]

```
Matrix m = new Matrix();
m.Scale(2, 2);
```

Abb. 32: Code Beispiel der Scale Methode, Quelle: Eigene Darstellung.

5.2.4 Drehungen

Um eine Drehung zu realisieren, müssen die Übergabeparameter m11, m12, m21 und m22 mit folgender Gleichung errechnet werden.

$$\begin{vmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad (5.4)$$

Wird die Matrix ohne Methode *Rotate* verwendet, müssen die Übergabewerte errechnet werden. Wird die Methode *Rotate* verwendet, wird als Übergabeparameter nur mehr dem gewünschten Dreh Winkel übergeben. Der Dreh Winkel im folgenden Beispiel beträgt 45 Grad.[58]

Die Methode *Rotate* wird folgendermaßen verwendet:

```
Matrix m = new Matrix();
m.Rotate(45);
```

Abb. 33: Code Beispiel der Scale Methode, Quelle: Eigene Darstellung.

[56] Vgl. Doberenz/Gewinnus (2013), S. 1746.

[57] Vgl. Doberenz/Gewinnus (2013), S. 1746.

[58] Vgl. Doberenz/Gewinnus (2013), S. 1747.

Um die Matrix Klasse auf ein Grafikelement anwenden zu können, muss die Matrix m an das Graphikelement übergeben.

```
e.Graphics.Transform = m;
```

Abb. 34: Code Beispiel Übergabe der Matrix, Quelle: Eigene Darstellung.

5.3 Modellierungssprache UML

Um die Dokumentation der Programarchitektur festzuhalten, soll die Modelliersprache Unified Modeling Language (UML) zum Einsatz kommen. Das Grobdesign für den Entwurf des Programmes soll vor der praktischen Umsetzung entworfen werden.

UML ist eine grafische Sprache für die Spezifizierung, Visualisierung, Konstruktion und Dokumentation von Softwaresystemen. Mit dieser, können auf standardisierte Weise, Informationen des Grundaufbaus eines Programmes, Systemfunktionen, programmierte Klassen, sowie Softwarekomponenten dargestellt werden.[59]

Für die Dokumentation soll das Klassendiagramm der UML eingesetzt werden. Klassendiagramme beschreiben die Struktur und die Architektur eines Systems, eine Grundlage für fast alle weiteren Beschreibungstechniken. Es werden damit Konzepte der realen Welt dargestellt.[60]

Aufgaben einer Klasse können sein:[61]

- Bildung einer konzeptuellen Einheit von Methoden und Attributen.
- Instanzen als Objekte ausführen
- Bedeutung von Objekten darstellen
- Beschreibung einer möglichen Implementierung
- Code innerhalb einer Klasse
- Anzahl aller Objekte die zu einem Zeitpunkt existieren
- Beschreibung möglicher Strukturen des Systems

Für die Erstellung eines Klassendiagrammes werden einige Begriffe benötigt. Die **Klasse** beinhaltet eine Vielzahl von Attributen und Methoden, welche das Verhalten einer instanziierten Klasse darstellen. Durch die Vererbung können Klassen miteinander verbunden werden. Der Klassenname hilft dabei bei der Identifizierung einer Klasse. Als **Attribut** werden Variablen in einer Klasse bezeichnet und besitzen einen Namen und einen Typ. In einer **Methode** werden Funktionen einer Klasse abgebildet und besteht aus zwei Teilen. Die Signatur beschreibt dabei die Sichtbarkeit, den Rückgabetyp, oder den Namen der Methode. Als Rumpf werden bei einer Methode ihre Deklarationen von lokalen Variablen und der eigentliche

[59] Vgl. Booch/Rumbaugh/Jacobson (2006), S. 15.

[60] Vgl. Rumpe (2012), S. 34.

[61] Vgl. Rumpe (2012), S. 34.

Anweisungsteil bezeichnet. **Modifikatoren** werden vor Methoden und Attributen gestellt, um Sichtbarkeit, Veränderlichkeit und Instanzierbarkeit darzustellen. Für die drei Modifikatoren *public, protected* und *private* wird ein +, # und ein - als Notation verwendet. **Konstanten** können nur gelesen werden und sind als *static* und *final* definiert. Attribute und Methoden können an eine andere Klasse vererbt werden. Die Unterklasse erbt dabei von der Oberklasse die Attribute und Methoden. Eine **Vererbung** wird durch einen Pfeil von der Unterklasse zur Oberklasse dargestellt.[62]

5.4 FTP Kommunikation

Bei einer FTP Kommunikation handelt es sich um eine sogenannte Datenbörse, die über das Internet erreichbar ist. Die dabei ausgetauschten Dateien sind dabei unverschlüsselt. Die Verbindung wird durch einen Hostnamen oder einer IP-Adresse hergestellt. Um eine FTP Verbindung aufzubauen wird ein FTP-Client Programm benötigt, das mit einem FTP-Server Kontakt aufnimmt. Auf dem Server muss der FTP-Dämon laufen und wird als eigener Prozess gestartet.

Bei der Kommunikation wird standardmäßig der Port 20 verwendet. Wurde eine Verbindung aufgebaut kann über den Benutzernamen und einem Passwort auf die vom Server freigegebenen Verzeichnisse zugegriffen werden. Der Zugriff erfolgt dann wie auf ein Verzeichnis am eigenen Rechner. Dateien einer FTP Übertragung können beispielsweise Textdateien oder Bilder sein, welche durch einen Download an den eigenen Rechner übertragen werden können. Besitzt der Ordner Schreibrechte, können Dateien auf das Verzeichnis des FTP Servers mit einem Upload auf den Server geladen werden.[63]

Für die Implementierung des Software-Tools in die Testumgebung, wird zusätzlich zum lokalen Öffnen des Robotercodes ein Datentransfer über FTP benötigt. Für diesen Zweck wird die .Net Klasse *FtpWebrequest* verwendet. Das Objekt wird mit der statischen Methode *Create* der *FtpWebrequest* erzeugt. Jedoch gibt diese bei der Instanziierung eine Referenz vom Typ *Webrequest* zurück und muss deshalb in eine *FtpWebrequest* Referenz gecastet werden.

Bei der Instanziierung wird zusätzlich der Dateipfad mitangegeben. Mit Hilfe der Eigenschaft *Credential* werden die Benutzerinformationen, Benutzernamen und Passwort festgelegt. Für den Download oder Upload einer Datei wird in der Eigenschaft *Method* die Methode *WebRequestMethods.Ftp.DownloadFile* oder *WebRequestMethods.Ftp.UploadFile,* angegeben. Um eine Abfrage über auszuführen wird die Methode *GetResponse* ausgeführt. [64]

[62] Vgl. Rumpe (2012), S. 35.

[63] Vgl. Ernst/Schmidt/Beneken (2016), S. 296.

[64] Vgl. Bayer (2010), S. 670.

Das zu übertragene File wird in einen Stream umgewandelt. Für eine FTP Kommunikation wird ein FTP-Stream benötigt. Das Grundprinzip eines Streams besteht darin, eine Verbindung von den temporären Daten des Programmes zu dauerhaft gespeicherten Daten herzustellen. Ein Stream stellt dabei eine allgemeine Ansicht einer Bytefolge bereit. Die Basisklasse der Stream Klassen ist die System.IO.Stream Klasse. Diese Verbindung zwischen Programm und Datei ist in der nachfolgenden Abbildung dargestellt.[65]

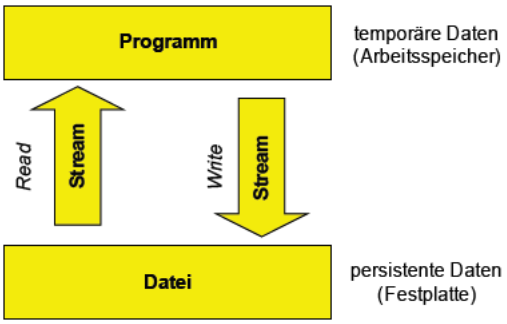

Abb. 35: Grundprinzip eines Streams, Quelle: Doberenz/Gewinnus (2013), S. 406.

Für die Implementierung der FTP Kommunikation und Speichern der Konfigurationsdatei sollen folgende Stream Klassen zum Einsatz kommen:[66]

- **FileStream:** Diese Klasse erlaubt die Erstellung einer Stream-Instanz die auf einer Datei basiert.
- **StreamReader:** Hierbei wird ein Textreader Objekt implementiert das Zeichen aus einem Bytestream einliest.
- **StreamWriter:** Durch diese Klasse wird ein Textwriter Objekt implementiert das Zeichen in einen ByteStream schreibt.
- **BinaryFormatter:** Diese Klasse kann ein Objekt durch Serialisierung in einen Stream schreiben oder durch Deserialisierung von dort lesen.

[65] Vgl. Doberenz/Gewinnus (2013), S. 405 f.

[66] Vgl. Doberenz/Gewinnus (2013), S. 406.

5.5 Speichern der Konfigurationsdatei

Bei der Einrichtung der Roboterbahn und des Hintergrundbildes entstehen Einstellungen, die ohne eine Speicherfunktion bei jedem Start des Software-Tools erneut zu tätigen sind. Daher sollen alle anfallenden Einstellungen in einem Konfigurationsfile gespeichert werden. Dieses File vereinfacht die Handhabung des Tools. Das konfigurierte Bild und die Roboterbahn können somit bei Programmstart automatisch geladen werden. Um Felder eines Konfigurationsobjekts in ein File schreiben zu können, müssen sie in sequenzielle Daten umgewandelt werden.

Dazu soll der Serialisierungs-Mechanismus des .NET-Frameworks zum Einsatz kommen. Durch diese Serialisierung können Daten in binär oder XML-Daten umgewandelt werden und somit permanent abgespeichert oder auch über ein Netzwerk versendet werden. Die Deserialisierung rekonstruiert so aus gespeicherten Daten wieder die originale Objektinstanz. Dazu können zwei verschiedene Serialisierungs Mechanismen des .NET-Framework verwendet werden. Zu diesen zählt die *Shallow*-Serialisierung der Klasse *System.Xml.Serialization.XmlSerializer* und die Deep-Serialisierung der Klassen *BinaryFormatter* und *SoapFormatter*.

Beide befinden sich im Namespace System.Runtime.Serialization. Für die Implementierung der Konfigurationsspeicherung soll die Deep-Serialisierung verwendet werden, da bei der Shallow-Serialisierung geschützte und private Objektfelder nicht berücksichtigt werden. Soll diese Serialisierung bei einer Klasse angewendet werden, muss die Klasse mit dem Attribut *Serializable* versehen werden.[67]

Für die Speicherung der Daten werden folgende Schritte benötigt. Wird eine Serialisierung mit der Methode *Serialize* durchgeführt, so wird ein Objekt in einem binären Stream serialisiert. Durch die Methode *Deserialize* wird dieser Stream wieder in ein Objekt eingelesen. Für diese Methoden soll eine Klasse *Config* erstellt werden, in der die zu speichernden Daten definiert werden. Wie schon erwähnt muss diese Klasse mit dem Attribut *Serializable* gekennzeichnet werden.[68]

Die somit behandelten theoretischen Themen können weiterführend zur Entwicklung des Software-Tools herangezogen werden.

[67] Vgl. Doberenz/Gewinnus (2013), S. 81.

[68] Vgl. Bayer (2010), S. 1108 f.

6 ENTWICKLUNG DES SOFTWARE-TOOL

Um einen Überblick über die Inhalte zu erhalten, wurde vor der Implementierung der einzelnen Funktionen ein Konzept erstellt, welches aus den definierten Basisfunktionalitäten besteht. Gestartet wurde dabei beim Einlesen einzelner G-Code Befehlen aus diesen die benötigten Informationen extrahiert wurden. Mit diesen Funktionen konnten die Grafischen Elemente angelegt und getestet werden. Die Darstellung der Roboterbahn wurde dabei in drei Unterabschnitten aufgeteilt. In weiterer Folge wurde auf dieser grafischen Anzeige das Ändern der verschiedenen Punkte aufgesetzt. Im Anschluss daran wurde die Darstellung des Hintergrundbildes und die dazugehörigen Konfigurationsmöglichkeiten implementiert. Mit der Herstellung des Robotercodes mit den geänderten Positionen konnten bereits erste Test des Tools ausgeführt werden.

Im weiteren Verlauf dieses Kapitels wird dieser Entwicklungsvorgang ausführlich behandelt.

6.1 Konzept des Programms

Um die Funktionen und den Aufbau des Programmes darzustellen, wurde ein Konzept des Programmes angefertigt. Ausgehend von dem Robotercode soll das Programm folgende Kernelemente beinhalten.

- Einlesen des Robotercodes
- Erstellung von Listen mit den eingelesenen Daten
- Berechnungen für die Anzeige der Bahn
- Darstellung der Roboterbahn
- Änderung der Bahn
- Einfügen und anpassen eines Hintergrundbildes
- Konfiguration speichern

In der nachfolgenden Abbildung ist dieses erarbeitete Konzept dargestellt.

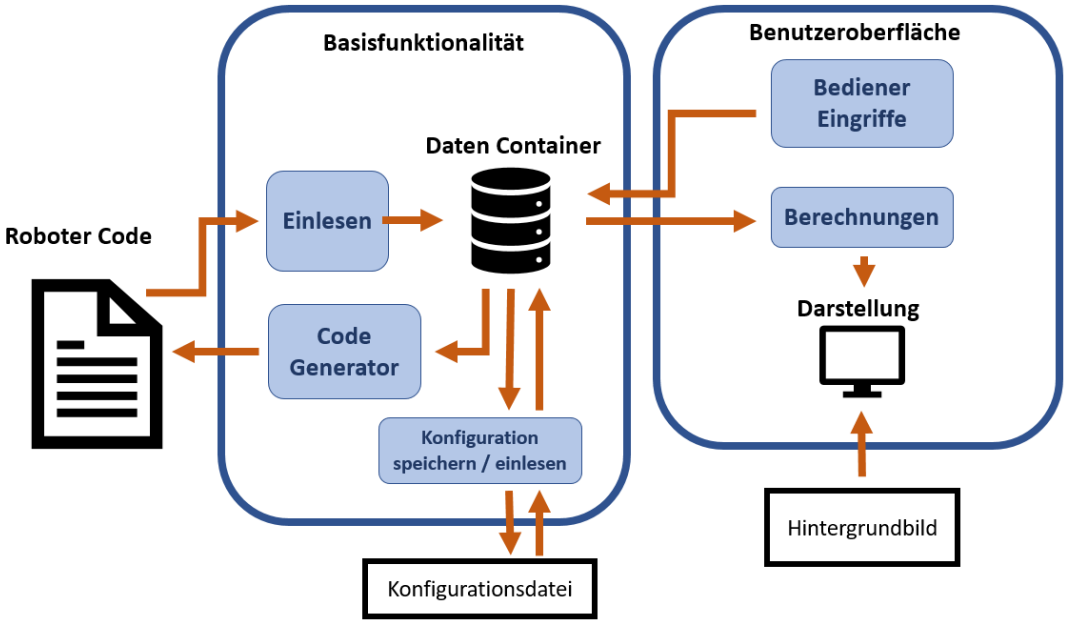

Abb. 36: Entwurf Programmaufbau, Quelle: Eigene Darstellung.

6.2 Umsetzung der einzelnen Programmteile

Die Umsetzung zur Erstellung des Softwareprototyps wurde anhand eines Robotercodes erstellt, der einen G01 und einen G03 Befehl beinhaltet. Mit diesem Roboter Test Programm wurden die einzelnen Programmteile des Software-Tools erstellt und getestet. Um die Komplexität der praktischen Umsetzung zu verringern beschränkt sich die Erklärung auf diese beiden Befehle. Weiters sind alle Koordinatenangaben und Längen in Millimeter angegeben.

6.2.1 Einlesen der Daten

Aufgabenstellung: Einlesen des Robotercodes und Erstellung einer Liste mit den benötigten Informationen der einzelnen Punkte. Die dadurch erstellte Liste wird für die Darstellung der Bahn benötigt. Jede Zeile besteht aus einer X, Y und einer Radius Komponente. Der Radius soll dabei aber nur bei einem G03 Befehl in die Liste übernommen werden.

Umsetzung: Der Robotercode wird zeilenweise eingelesen, jede Zeile wird zuerst auf die Fahranweisung durchsucht. Wird ein Fahrbefehl zum Beispiel ein G01 gefunden, wird diese Information für die aktuelle Zeile gespeichert. Danach erfolgt die Suche nach den X und Y Parametern. Die werden in die Zeichen nach einem X oder Y in eine Dezimalzahl umgewandelt und ebenfalls als Parameter für diese Zeile in die Liste abgespeichert. Wird eine Fahranweisung G03 gefunden wird auch der Parameter R, der den Radius des Kreises beschreibt in eine Dezimalzahl umgewandelt und zur Zeile hinzugefügt. Dieser Vorgang wird so lange durchgeführt, bis alle vorhandenen Zeilen des Robotercodes durchsucht wurden. Die daraus erstandene Liste enthält nun nur mehr die relevanten Informationen, die für die grafische Darstellung der Roboterbahn benötigt werden. Kommentare und Befehle die für die Darstellung der Bahn irrelevant sind werden nicht bearbeitet.

In der Abb. 37 ist der Beispielcode für die praktische Umsetzung dargestellt. In diesem Code sind vier Fahrbefehle und zwei Kommentare enthalten. Die Kommentare werden beim Einlesen des Files ignoriert. Der G0 Befehl in diesem Beispielcode wird für das Anfahren des Startpunktes benötigt, der die Z-Achse auf -47 mm positioniert. In diesem gezeigten Roboterprogramm startet der Ablauf beim Koordinatenursprung, die zu ändernde Bahn startet aber erst bei den Koordinaten X 40 und Y 40. Zuerst positioniert sich der Roboter auf die erste Position, danach wird die Z-Achse verfahren. Erst ab diesem Punkt wird die Roboterbahn im Software-Tool visualisiert.

```
;Roboterbahn Anfang
G01 X40 Y40
G0 X40 Y40 Z-47
G01 X40 Y140
G03 X90 Y187 R45
;Roboterbahn Ende
```

Abb. 37: Beispielcode für praktische Umsetzung, Quelle: Eigene Darstellung.

Die generierte Liste aus diesem File beinhalten nur mehr die relevanten Daten, nämlich Fahrbefehl, X-, Y-Koordinaten und den Radius R. Diese Liste ist in der nachfolgenden Abbildung angeführt.

Zeilennumer	Fahrbefehl	X	Y	R
2	G01	40	40	
4	G01	40	140	
5	G03	90	187	45

Abb. 38: Generierte Liste des eingelesenen Robotercodes, Quelle: Eigene Darstellung.

6.2.2 Anzeigen der Roboterbahn

Aufgabenstellung: Grafische Anzeige der generierten Punkte in der Liste. Dabei soll der Fahrbefehl G01 und G03, als zusammenhängende Bahn dargestellt werden.

Umsetzung: Bei der Umsetzung dieses Punktes wurden zwei Programmelemente angelegt. Für das Zeichnen der einzelnen Befehle wurde die Klasse Drawing erstellt. Sie beinhalten einerseits die Methode für eine lineare Verbindung zweier G01 Fahrbefehle und anderseits die Methode für die Darstellung eines G03 Kreisbogen Fahrbefehls.

- **Lineare Verbindung zweier G01 Punkte:** Für die Implementierung dieser Funktion wurde die Methode *DrawLine* verwendet. Diese Methode benötigt einen Start- und einen Endpunkt für die Darstellung einer Linie. Im angeführten Beispielcode wird als Startpunkt die Zeile zwei und als Endpunkt die Zeile drei an die Methode übergeben. Als zusätzlichen Übergabeparameter muss ein Pen angelegt werden. Diese Klasse definiert ein Objekt für die Darstellungseigenschaften einer Linie. Die Linien werden im Software-Tool rot und einer Pixelstärke von vier dargestellt. Die Instanziierung dieses Pen Objekts ist in der Abb. 39 ersichtlich.

```
//Pen anlegen
Pen myPen = new Pen(Color.Red, 4);
```

Abb. 39: Instanziierung Pen, Quelle: Eigene Darstellung.

Im folgender Abbildung ist die Methode DrawG01 dargestellt.

```
private void DrawG01(Graphics g, Gcode startPoint, GCode_G01 endPoint)
{
    g.DrawLine(myPen, startPoint.Point, endPoint.Point);
}
```

Abb. 40: Methode DrawG01, Quelle: Eigene Darstellung.

Im Ausgabefenster wird die eingelesene Bahn wie in Abb. 41 dargestellt. Der Koordinatenursprung liegt dabei im liken oberen Eck. Um den Bezug zu den einzelnen Punkten in der Liste herzustellen,

wurden in der Abb. 41 die Punktinformationen hinzugefügt. In der Software sind diese jedoch nur in einer Tabelle im linken Bereich des Software-Tools ersichtlich.

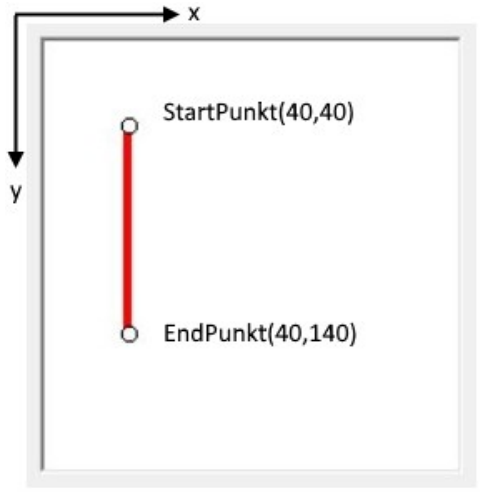

Abb. 41: Darstellung einer Linie der eingelesenen Punkte, Quelle: Eigene Darstellung.

- **Verbindung zweier Punkte mit einem Kreisbogen:** Um einen Kreisbogen mit der Methode *DrawArc* zu zeichnen sind mehrere Berechnungen im Vorfeld erforderlich. Wie in Unterkapitel 5.1 dargestellt, wird die Seitenlänge und der Startpunkt eines Quadrats das den Kreis beinhaltet benötigt. Zu diesem Zweck wurde der Mittelpunkt des Kreises errechnet. Für die einzelnen Berechnungen wurde die Klasse *Calculate* angelegt.
  Für die Berechnung der Mittelpunkt Koordinaten wurde in der Klasse *Calculate* die Methode *Center* hinzugefügt. Bei einem Fahrbefehl G03 ist die angegebene Position der Endpunkt des Kreisbogens und der Punkt vor dem G03 Befehl der Startpunkt des Kreisbogens. Wie in Abb. 42 sind zwei Punkte am Umfang des Kreises bekannt. Durch die folgende geometrische Lösung kann der Mittelpunkt bestimmt werden:

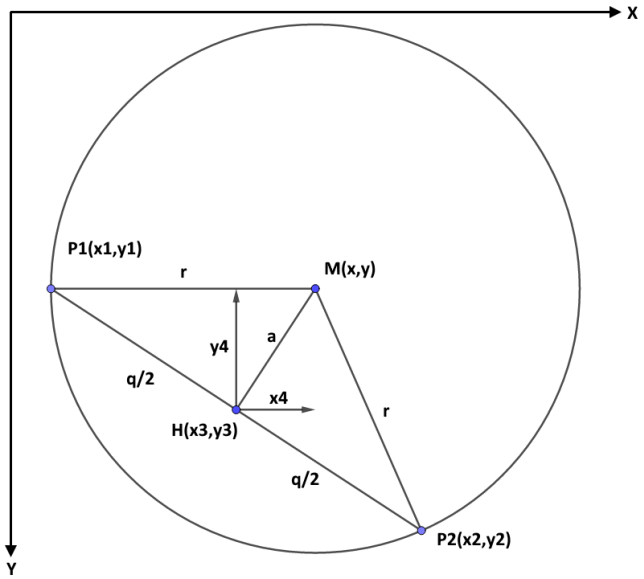

Abb. 42: Geometrische Darstellung für Mittelpunkts berechnung, Quelle: Eigene Darstellung.

Allgemeine Gleichung für Punkt M(x,y):

$$x = x_3 + x_4 \tag{6.1}$$

$$y = y_3 + y_4 \tag{6.2}$$

Zuerst wird die Länge der Hypotenuse q mit folgender Gleichung berechnet:

$$q = \sqrt[2]{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{6.3}$$

Die Koordinaten x_3 und y_3 des Punktes H werden über die nachfolgende Berechnung ermittelt:

$$x_3 = \frac{x_1 + x_2}{2} \tag{6.4}$$

$$y_3 = \frac{y_1 + y_2}{2} \tag{6.5}$$

Berechnung der Strecke a:

$$a = \sqrt[2]{r^2 - (\frac{q}{2})^2} \tag{6.6}$$

Berechnung des Einheitsvektors:

$$Einheitsvektor = (\frac{y1 - y2}{q}, \frac{x2 - x1}{q}) \tag{6.7}$$

Für die Bildung der Koordinaten des Mittelpunkts werden die zwei zusätzlichen Längen x_4 und y_4 benötigt. Dazu wird der Einheitsvektor für die X bzw. Y Komponente mit dem X bzw. Y Anteil des Einheitsvektors multipliziert. Der Mittelpunkt ergibt sich durch eine Addition der Länge x_3 und der Länge x_4 sowie y_3 und y_4.
Berechnung der x_4 und y_4 Komponente:

$$x_4 = a \cdot \frac{y_1 - y_2}{q} \tag{6.8}$$

$$y_4 = a \cdot \frac{x_2 - x_1}{q} \tag{6.9}$$

Die Methode Center ist folgendermaßen aufgebaut und besitzt als Rückgabewert den Mittelpunkt als Koordinate X und Y.

```
public static void Center(double x1, double y1, double x2, double y2, double radius, out double x, out double y)
{
    double radsq = radius * radius;
    double q = Math.Sqrt(((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 - y1)));

    double x3 = (x1 + x2) / 2;
    x= x3 + Math.Sqrt(radsq - ((q / 2) * (q / 2))) * ((y1 - y2) / q);

    double y3 = (y1 + y2) / 2;
    y = y3 + Math.Sqrt(radsq - ((q / 2) * (q / 2))) * ((x2 - x1) / q);
}
```

Abb. 43: Methode Center, Quelle: Eigene Darstellung.

Durch den Erhalt der Koordinaten des Kreismittelpunktes kann der Startpunkt des benötigten Quadrats errechnet werden. Dazu wird jeweils der Radius des Kreises von der X und Y Komponente des Mittelpunktes subtrahiert. Die Seitenlänge des Quadrats ergibt sich aus dem doppelten Radius. Dieser Zusammenhang ist in der nächsten Abbildung ersichtlich.

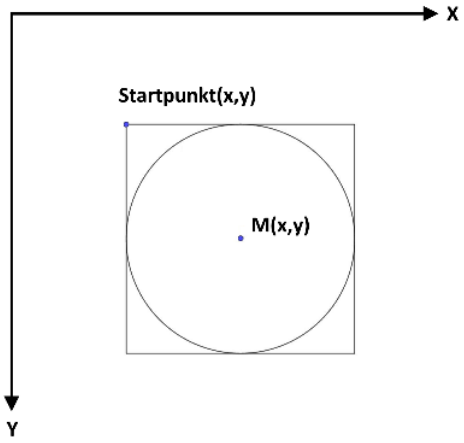

Abb. 44: Darstellung des errechneten Quadrats, Quelle: Eigene Darstellung.

Um dieses Quadrat als Objekt weiter zu verwenden wurde im Quellcode eine Rechteck Klasse instanziiert welche den Startpunkt und die Größe beinhaltet.

Für die Darstellung des gewünschten Kreisbogens wird, wie in Abb. 29 dargestellt, der Start- und Bogenwinkel benötigt. Die Berechnung dafür ist so aufgebaut, dass der Programmteil als eigene Methode ausgeführt ist und so ein redundanter Programmcode vermieden wird. Die Berechnung der beiden Winkel wurde wiederum mit einer geometrischen Herangehensweise gelöst.

In Abb. 45 ist der Lösungsweg für den Startwinkel abgebildet, welche in der Farbe orange dargestellt ist. Um den Startwinkel zu berechnen kann der Winkel α mit Hilfe des Dreieckes bestimmt werden. Der Winkel α ergibt sich genau aus der Hälfte des Startwinkels. Die Schritte für die Berechnung des Startwinkels sind nach der Abb. 45 angeführt.

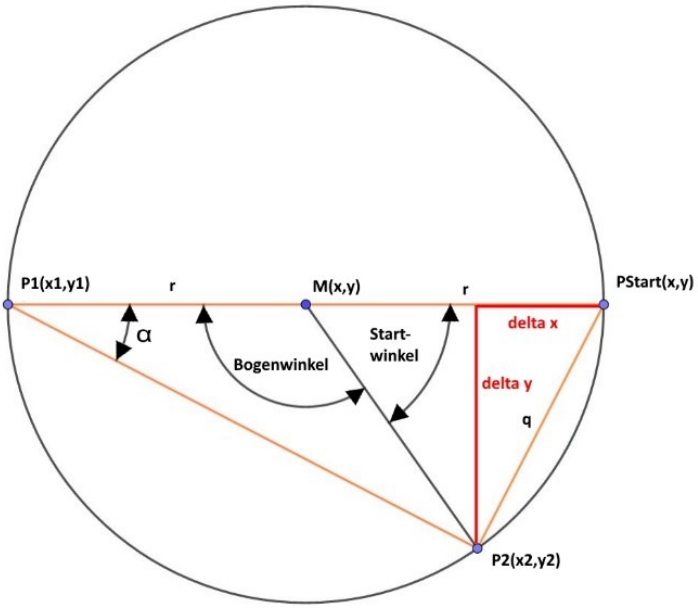

Abb. 45: Geometrische Bestimmung des Start- und Bogenwinkels, Quelle: Eigene Darstellung.

Für die Berechnung des Abstandes q muss ein fiktiver Punkt (PStart), welcher die Null Achse der Winkelberechnung bildet, eingefügt werden. Durch Subtraktion der jeweiligen Koordinaten von PStart und P2 kann delta X und delta Y bestimmt werden. Mit Hilfe des Pythagoras wird die Strecke q berechnet.
Der Startwinkel lässt sich nun folgendermaßen bestimmen:

$$Startwinkel = (\sin^{-1}(\frac{\frac{q}{2}}{r})) * 2 \qquad (6.10)$$

Dieselben Rechenschritte sind für die Berechnung des Bogenwinkels erforderlich. Hierbei wird das jeweilige delta zwischen den Punkten P1 und P2 gebildet.

Mit Hilfe der Parameter für die Methode *DrawArc* wird sie für jeden G03 Endpunkt in der vorhandenen Liste ausgeführt.

```
//Darstellung des Kreisbogens
g.DrawArc(myPen, rec, Convert.ToSingle(StartAngel), Convert.ToSingle(SweepAngel));
```

Abb. 46: Methode *DrawArc*, Quelle: Eigene Darstellung.

Zusätzlich zu dem Kreisbogen wird der Mittelpunkt des Kreisbogens dargestellt. Mit Hilfe dieses Punktes kann der Radius des Kreisbogens verändert werden. Diese Berechnung wird von der Methode *RadiusP* ausgeführt. Durch die Formel des Einheitskreises und dem Winkel des Mittelpunktes auf dem Kreisbogen α, wird jeweils ein Faktor für die X und Y Richtung gebildet. Durch Multiplikation mit dem Radius ergibt sich der X und Y Abstand zum Mittelpunkt des Kreises. Werden diese Abstände zum Mittelpunkt addiert ergibt sich der Mittelpunkt des Kreisbogens. Die nachfolgende Abbildung zeigt diesen Zusammenhang.

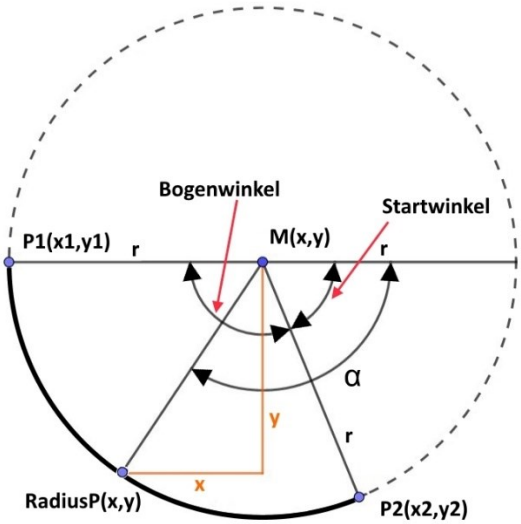

Abb. 47: Geometrische Bestimmung des Mittelpunktes am Kreisbogen, Quelle: Eigene Darstellung.

Berechnung des Winkels α:

$$\alpha = Startwinkel + \frac{Bogenwinkel}{2} \tag{6.11}$$

Berechnung x und y Koordinaten vom Mittelpunkt des Kreisbogens:

$$x = \cos(\alpha) \cdot r \tag{6.12}$$

$$y = \sin(\alpha) \cdot r \tag{6.13}$$

Durch die Addition dieser beiden x- und y-Koordinaten und den Mittelpunktkoordinaten M(x,y) wird der Mittelpunkt am Kreisbogen errechnet.

Im Ausgabefenster wird die Linie zwischen den ersten beiden Punkten und der Kreisbogen für einen G03 Fahrbefehl angezeigt. Ebenso wird wie in Abb. 48 gezeigt der Mittelpunkt am Kreisbogen markiert dargestellt. Wie in Abb. 38 beträgt der nun berechnete Radius des Kreisbogens 45 mm.

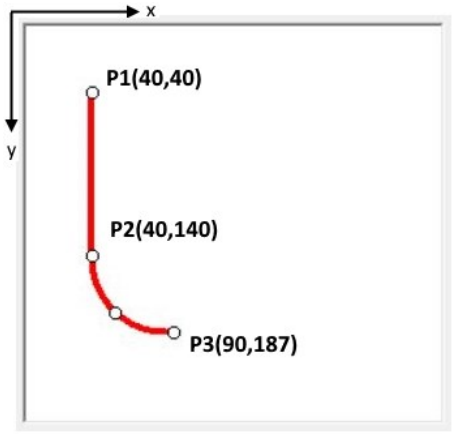

Abb. 48: Darstellung der eingelesenen Punkte, Quelle: Eigene Darstellung.

Mit der Methode *Drawing* kann eine Roboterbahn mit G01 und G03 Fahrbefehlen dargestellt werden. Durch diese grafische Darstellung verschwindet die Syntax des Robotercodes im Hintergrund und vereinfacht somit die programmierte Bahn. Zusätzlich werden die eingelesenen Positionen in einer Tabelle angezeigt. Damit soll die Übersicht der enthaltenen Koordinaten gewehrleistet werden.

Der nächste Abschnitt beinhaltet die Realisierung der Adaptierung dieser Punkte.

6.2.3 Ändern einzelner Punkte

Aufgabenstellung: Eine Adaptierung der Roboterbahn soll mittels einer Drag and Drop Funktion ermöglicht werden. Zusätzlich soll es ermöglicht werden die Positionen über eine zusätzliche Eingabe verändern zu können. Dabei soll es nicht möglich sein, Änderungen der Befehlszeilen im Robotercode durchzuführen. Um den BedienerInnen des Software-Tools die Handhabung zu erleichtern, soll eine mögliche Änderung eines Punktes visualisiert werden.

Umsetzung: Für die Verschiebung einzelner Punkte sind mehrere Funktionen im Einsatz. Durch die Änderung des Mauszeigers, welche das Überfahren eines Punktes auslöst, wird beim Anklicken dieses Punktes eine Adaptierung dieses Punktes ermöglicht. Dadurch werden zwei Arten von möglichen Änderungen aktiviert. Wird der Punkt mit Hilfe eines Klicks markiert, kann eine Änderung über Schiebebalken erfolgen. Hierfür stehen drei Schiebebalken zur Verfügung, welche die Parameter des Punktes wiederspiegeln. Bei einem G01 Fahrbefehl können nur die X und Y Komponenten geändert werden. Bei einem Mittelpunkt am Kreisbogen kann der Radius des dazugehörigen G03 Befehls angepasst werden.

Bleibt die linke Maustaste an einem bestimmten Punkt gedrückt, kann über die Drag and Drop Funktion dieser an eine neue Position verschoben werden. Hierzu wird beim Anklicken einer Position die aktuelle Mausposition gespeichert und als Offset zur ausgewählten Position addiert. Im Hintergrund wird diese Änderung in die generierte Liste übergeben. Die grafische Anzeige der Bahn wird kontinuierlich neu

aufgebaut, um die Verschiebung des Punktes während des Ziehens anzuzeigen. Beim Loslassen der Maustaste wird diese Position abgespeichert.

Die Änderungen die diese zwei Arten ermöglichen, werden durch Grenzen, die sich durch den Radius und der Größe des Anzeigefensters ergeben eingeschränkt. Somit lässt sich eine Generierung von unerreichbaren Positionen für den jeweiligen Roboter ausschließen.

Die hierfür verwendeten Methoden wurden von der Onlinequelle (Stephens (2017), Online-Quelle [14.November.2017]) bezogen. Für die Implementierung in das Software-Tool wurden diese jedoch nach Bedarf modifiziert.

6.2.4 Hintergrundbild einfügen und konfigurieren

Aufgabenstellung: Um den Bezug zwischen der Roboterbahn und dem Werkstück herzustellen, soll eine Grafik als Hintergrund der angezeigten Bahn eingefügt werden. Auf dem Bild ist eine Draufsicht des zu bearbeitenden Bauteils abgebildet. Zusätzlich ist der Koordinatenursprung und die Richtung der X- und Y-Achsen eingezeichnet. Diese Grafik wird für die Erstellung des Roboterprogrammes und der ersten Teach-In Verfahren benötigt. Der Koordinatenursprung deckt sich dabei mit dem Anwenderkoordinatensystem des jeweiligen Roboters, welches im Unterkapitel 4.1 unter Bezug zwischen Bauteil und Roboterbahn behandelt wird. Es sollen folgende Bearbeitungsmöglichkeiten des Bildes programmiert werden:

- Bild einfügen
- Bild skalieren
- Bild drehen
- Bild verschieben

Diese Änderungen der Grafik sollten sich dabei nicht auf die dargestellte Bahn auswirken.

Umsetzung: Um eine Grafik in den Anzeigebereich des Software-Tools zu laden wurde die Methode *DrawImage* verwendet. Unter dem Button Einstellungen wird der Pfad des einzufügenden Bildes hinterlegt. Durch die abgeschlossene Eingabe des Speicherortes wird das Bild im Anzeigebereich dargestellt und steht somit für die Anpassungen zur Verfügung. Mit der integrierten Funktion „Skalieren“ kann das Bild auf die Größe des dargestellten Bildes skaliert werden.

Dazu wird ein bekannter Abstand am Bild benötigt. Durch einen definierten Abstand der am Bild eingetragen ist und einem zweiten Wert der diesen Abstand in Pixel angibt wurde ein Skalierungsfaktor generiert. Um den Abstand im Anzeigefenster zu erhalten kann über die Maus eine Skalierungslinie ins Bild gezeichnet werden. Dadurch wird ein Startpunkt und Endpunkt der Linie erzeugt.

Die Länge dieser Linie wird mit Hilfe des pythagoreischen Lehrsatzes errechnet, welche die Länge der Linie in der Einheit Pixel angibt. Wird der Abstand in Millimeter durch die Länge in Pixel dividiert, erhält man den Skalierungsfaktor für das eingefügte Bild. Diese Skalierung wird nur auf die Hintergrundgrafik angewendet.

In der nachfolgenden Abbildung wird das zur Implementierung generierte Bild mit den Skalierungspunkten und dem Roboteranwenderkoordinatensystem dargestellt.

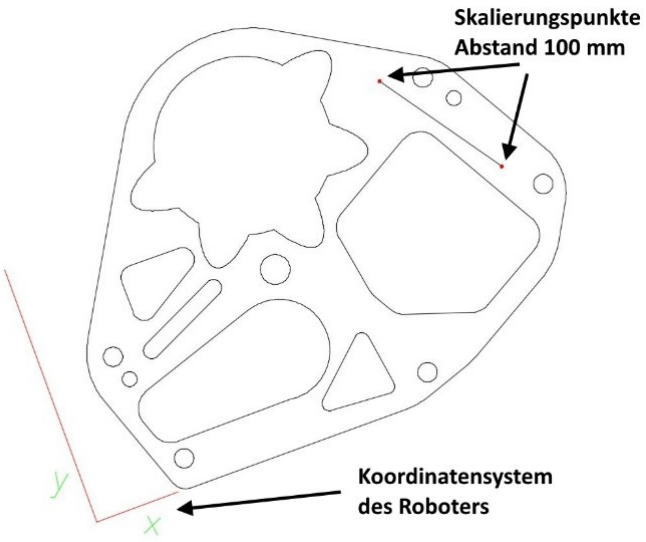

Abb. 49: Darstellung der Skalier Funktion, Quelle: Eigene Darstellung.

Für die Drehung des Bildes wurde eine Funktion implementiert, welche sich ebenfalls über eine hinzugefügte Linie den zu verdrehenden Winkel errechnet. Dazu muss diese Linie auf der X-Achse des dargestellten Koordinatensystems platziert werden. Die Drehung des Bildes erfolgt dabei immer in Bezug auf die X-Achse des Anzeigebereichs. Bei der Implementierung der Drehfunktion stellte sich jedoch heraus, dass durch eine Drehung, die von der Matrixklasse ausgeführt wird, das ganze Koordinatensystem gedreht wird, welche sich auch auf die nachfolgende dargestellte Bahn auswirkte.

Wird eine Drehung durch die Matrixklasse durchgeführt, werden alle Elemente die nach dieser Codezeile im Anzeigefenster dargestellt werden, in dieses gedrehte Koordinatensystem eingefügt. Dies bedeutet, dass das anzuzeigende Bild sich in diesem Koordinatensystem befindet. Nach dem Einfügen des Bildes muss daher für die nachfolgenden Elemente das Koordinatensystem wieder zurückgedreht werden. Wichtig ist hierbei, dass das zuvor eingefügte Bild sich nach wie vor in diesem gedrehten Koordinatensystem befindet. Dieser Vorgang wird in der Abb. 50 schematisch dargestellt.

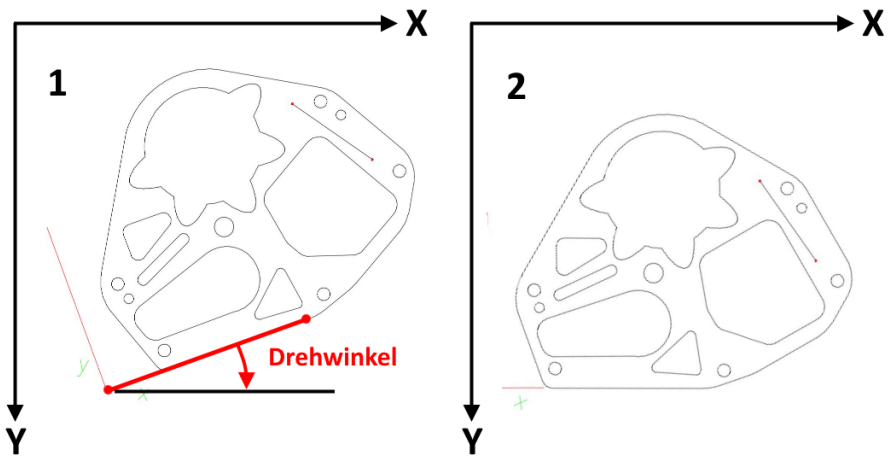

Abb. 50: Darstellung der Drehfunktion, Quelle: Eigene Darstellung.

Soll das Bild verschoben werden, bezieht sich die Verschiebung auf das Koordinatensystem bei dem das Bild eingefügt wurde. Um ohne komplizierte Berechnungen die Verschiebefunktion in das Software-Tool zu implementieren, muss eine definierte Reihenfolge der Transformationsbefehle eingehalten werden. Die Verschiebung des Bildes wird wie zuvor bei der Drehung durch die Verwendung der *Transform*-Methode der Matrixklasse realisiert. Die Parameter für die Verschiebung werden in einer eigenen Methode ermittelt (Dabei wird bei einem Mausklick die aktuelle Position der Maus im Anzeigefenster zwischengespeichert).

Wird mit gedrückter Maustaste die Mausposition verschoben, so wird die so entstehende Differenz an die *Transform*-Methode übergeben. Der Aufruf dieser Methode muss vor der *Rotate*-Methode erfolgen, da sich dieser Offset sonst auf das gedrehte Koordinatensystem bezieht. Zum Rücksetzen des Koordinatensystems nach dem Einfügen der Hintergrundgrafik wird eine initialisierte Matrix an das Anzeigefenster übergeben. Durch das Rücksetzen der Transformationen ergibt sich keine Verdrehung, oder Verschiebung der dargestellten Bahn.

Diese Funktionen können über drei Kontrollkästchen separat angewählt werden. Erst durch diese Anwahl können die verschiedenen Hilfslinien in das Bild gezeichnet werden. In der folgenden Abbildung sind diese Kontrollkästchen ersichtlich.

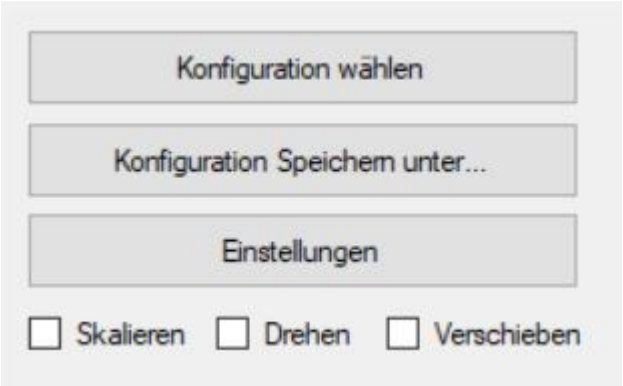

Abb. 51: Kontrollkästchen der Transformationsfunktionen, Quelle: Eigene Darstellung.

6.2.5 Speichern der geänderten Bahn

Aufgabenstellung: Wiederherstellen des Roboterfiles mit geänderten Koordinaten der einzelnen Punkte. Dabei ist zu beachten, dass sich außer diesen Koordinaten nichts an diesem File ändern soll.

Umsetzung: Im Zuge der Umsetzung dieser Funktion war es nicht ausreichend, wie Anfangs angenommen, nur die einzelnen Fahrbefehle einzulesen. Da ursprünglich für die „Einlesen“ Funktion nur einzelne G-Code Befehle verwendet wurden, konnte auf etwaige Kommentarzeilen im Robotercode nicht weiter eingegangen werden. Um jedoch den Erhalt des ursprünglichen Roboterprogrammes beim Abspeichern zu gewährleisten, mussten beim Einlesen einige Änderungen durchgeführt werden.

Wie schon erwähnt, enthält das einzulesende File zusätzlich zu den gezeichneten G-Code Kommandos auch Kommentare sowie nicht grafisch dargestellte Anweisungen. Um diese Informationen aber beim Speichern wiederherstellen zu können, müssen diese natürlich eingelesen und zwischengespeichert werden.

Beim Einlesen des Codes wird für jeden Codeabschnitt ein Listeneintrag vom Typ *ICodeSegment* erzeugt, welcher folgende Informationen enthält:

- **Code**: Unter dieser Variable wird der Inhalt des eingelesenen Codeabschnittes abgespeichert.
- **EndOfLine**: Werden beim Einlesen in einer Zeile mehrere G-Code Befehle detektiert, werden diese separat eingelesen und jeweils eigene Listeneinträge erzeugt. Damit diese Codeabschnitte wiederhergestellt werden können wird mitgespeichert, ob nach dem jeweiligen Codeabschnitt ein Zeilenumbruch erfolgen soll. Wenn das nicht der Fall sein sollte, wird der nachfolgende G-Code Befehl in dieselbe Zeile geschrieben. Besteht die Zeile nur aus einem Befehl oder einem anderen Inhalt wird diese Variable auf eins gesetzt.
- **IsGCode**: Um für das Zeichnen nicht verwertbare Zeilen von den Fahrbefehlen zu unterscheiden, wird diese Variable bei einem G-Code Befehl auf eins gesetzt.

Die Liste für das Zeichnen der Bahn besteht aber nur aus Instanzen der Klassen GCode_G01, GCode_G02 und GCode_G03. Diese Klassen sind von der Klasse GCode abgeleitet und erben die Attribute dieser. Um diese Liste zu erzeugen, werden aus der Gesamtliste nur diejenigen Elemente gefiltert, welche IsGCode auf eins gesetzt haben.

Für die Wiederherstellung des originalen Codes wird das vorhandene File neu aufgebaut. Dazu werden die einzelnen Elemente aus der Liste sequentiell in das File geschrieben. Enthält das Element die Information EndOfLine, so wird die Ausgabe des darauffolgenden Befehls in der nächsten Zeile fortgesetzt. Somit wird der ursprüngliche Code wiederhergestellt.

6.2.6 Konfiguration speichern

Aufgabenstellung: Mit dieser Funktion sollen alle Einstellmöglichkeiten rund um das Software-Tool gespeichert werden können. Zu diesen zählen beispielsweise der Dateipfad des Robotercodes und das Hintergrundbild. Dadurch soll bei einer erneuten Änderung der Roboterbahn das Konfigurieren der Roboterbahngrafik entfallen. Für die Konfiguration soll ein eigener Dateityp erstellt werden.

Umsetzung: Für die zu speichernden Variablen wurde eine neu Klasse mit dem Namen Config erstellt. In dieser Klasse befindet sich die Hintergrundgrafik selbst, der Dateipfad zum Robotercode und die X und Y Position, der Drehwinkel und der Skalierungsfaktor der Hintergrundgrafik. Für das Abspeichern der Konfiguration wurden drei Schaltflächen hinzugefügt (wie in Abb. 52 gezeigt), über die sich die Funktionen steuern lassen.

Mit der Schaltfläche „Konfiguration wählen“ ist es möglich, eine Konfigurationsdatei zu öffnen. Die Schaltfläche „Konfiguration speichern unter“ ermöglicht das Abspeichern einer neu angelegten Konfiguration. Im Einstellungsdialog besteht die Möglichkeit den Dateipfad zum Robotercode einzugeben und die Hintergrundgrafik einzufügen. In der folgenden Abbildung werden diese Einstellmöglichkeiten aufgezeigt.

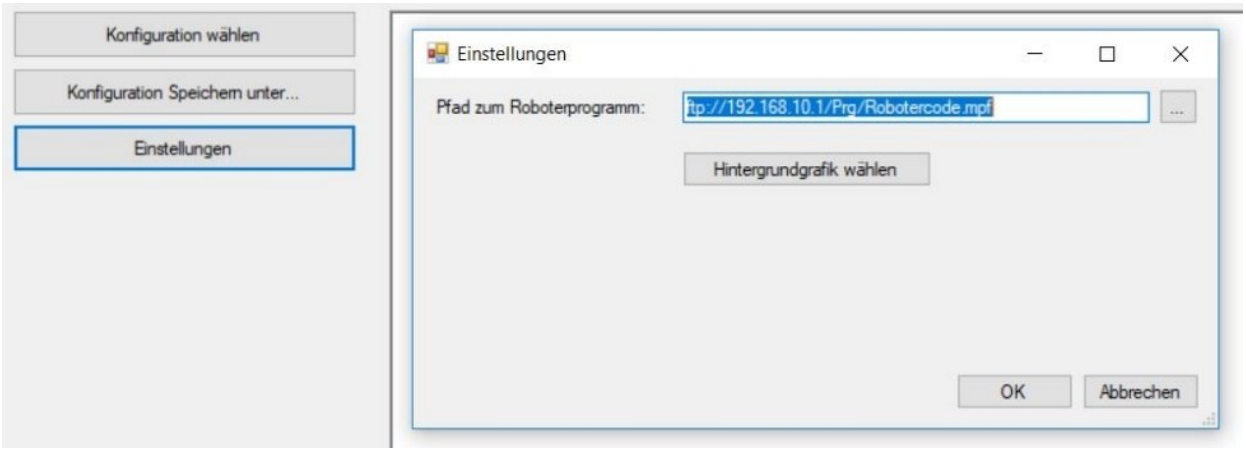

Abb. 52: Einstellungen im Software-Tool, Quelle: Eigene Darstellung.

Um eine mögliche falsche Eingabe abzubrechen, wurden diese Einstellungen jedoch nicht direkt im Objekt der Klasse Config gespeichert. Dazu wurde eine Kopie dieses Objekts angelegt. Der Einstellungsdialog arbeitet nur mit den Daten der Kopie, um nach erfolgreicher Eingabe, mit einem Klick auf OK das Originalobjekt zu überschreiben. Das Erzeugen dieser Kopie wurde mit einem *Memory Stream* umgesetzt. Durch Serialisierung des Config Objektes entsteht ein *binär Memory Stream* der durch Deserialisierung eine Kopie dieses Objektes erzeugt. In diese Kopie wird der Pfad und die Hintergrundgrafik gespeichert. Wird der Einstellungsdialog abgebrochen wird diese Kopie verworfen. Wird die Eingabe jedoch bestätigt, so überschreibt diese Aktion die Ursprungseinstellungen. Die nachfolgende Abbildung stellt den Ablauf dieser Funktion dar.

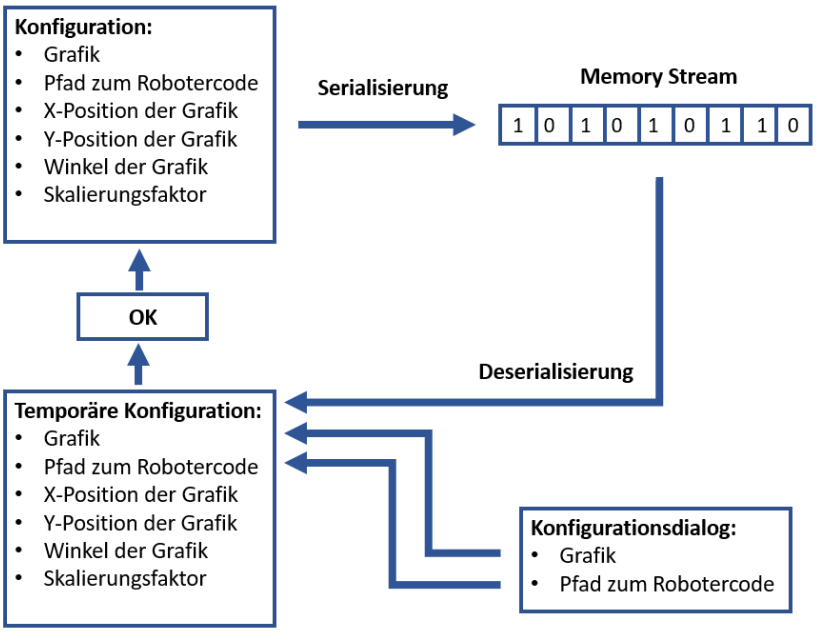

Abb. 53: Darstellung des Einstellungsdialoges, Quelle: Eigene Darstellung.

Das Abspeichern und Laden der Konfiguration basiert auf demselben Prinzip wie beim Erstellen der Kopie des Einstellungsobjektes. Mit Hilfe der Methode *File.Create* wird ein File und ein zugehöriger *FileStream* erstellt. Beim Serialisieren des Einstellungsobjektes werden die Daten nun in diesen *FileStream* geschrieben. Beim Laden wird mit der Methode *File.OpenRead* ein *read-only FileStream* für eine bestehende Datei erzeugt. Die Deserialisierung erzeugt aus dem File-Stream ein Config Objekt. Beide Varianten sind in der nachfolgenden Abbildung dargestellt.

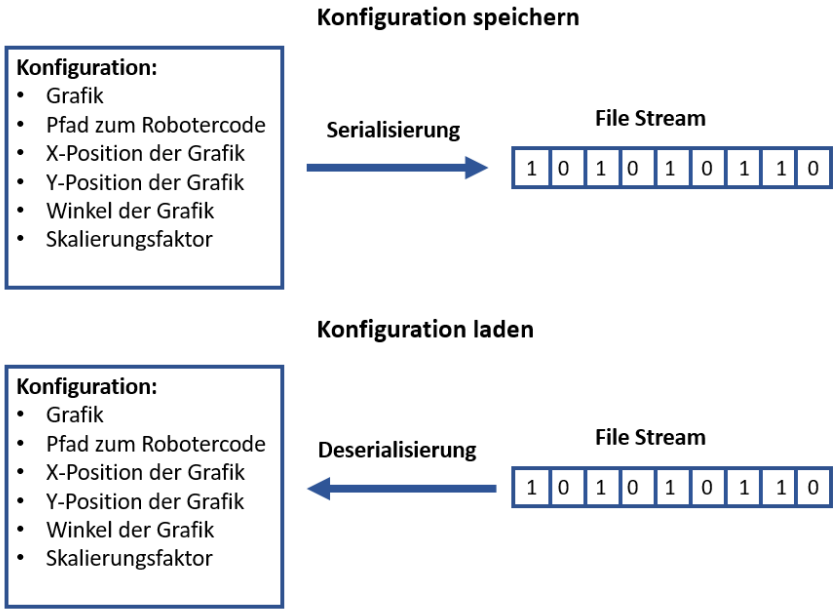

Abb. 54: Darstellung Konfiguration speichern und laden, Quelle: Eigene Darstellung.

6.2.7 Einlesen und Schreiben der Daten über FTP

Aufgabenstellung: Erweiterung der „Datei einlesen und schreiben“-Funktion, um den Robotercode direkt vom Roboter der Testumgebung einlesen zu können. Die bereits bestehende lokale Lösung soll bestehen bleiben.

Umsetzung: Die Robotersteuerung der Testumgebung besitzt einen FTP Server. Das Programm wurde so erweitert, dass das Schreiben und Lesen zusätzlich zu einem *FileStream* auch über die Stream-Implementierungen der Klassen *FtpWebResponse* und *FtpWebRequest* funktioniert. An die Klasse *FtpWebRequest* werden dazu Informationen über den Dateipfad, den Benutzer, das Passwort und die Betriebsart Upload und Download übergeben.

Mit der Methode *GetRequestStream* bekommt man ein Stream Objekt zurück, welches dann der *SaveStream* Methode übergeben wird. Als Beispiel wird der Code für das Abspeichern des Robotercodes herangezogen. Die Methode „Save“ beinhaltet das Abspeichern des Robotercodes auf dem lokalen Rechner und das Abspeichern über FTP. Welche davon aktiv ist, wurde über eine Zeichenabfrage im Dateipfad des Roboterprogrammes realisiert. In der folgenden Abbildung ist der beschriebene Programmcode dargestellt.

```
public void Save(string codeFilePath)
{
    if (codeFilePath.StartsWith("ftp"))
    {
        var reqFTP = (FtpWebRequest)FtpWebRequest.Create(new Uri(codeFilePath));
        reqFTP.Credentials = new NetworkCredential("Administrator", "Administrator");
        reqFTP.Method = WebRequestMethods.Ftp.UploadFile;

        using (Stream ftpStream = reqFTP.GetRequestStream())
        {
            SaveStream(ftpStream);
        }

        FtpWebResponse response = (FtpWebResponse)reqFTP.GetResponse();
    }
    else
    {
        using (Stream fileStream = File.Create(codeFilePath))
        {
            SaveStream(fileStream);
        }
    }
}
```

Abb. 55: Darstellung Save Methode, Quelle: Eigene Darstellung.

Für den Test der Datenübertragen mittels FTP Kommunikation wurde vorerst der Arbeitsordner des Roboters mit Hilfe des Windows Explorers geöffnet. Dabei wurde auch ein Testproramm in diesen Ordner kopiert um mit dem Software-Tool darauf zugreifen zu können. Danach wurde diese Datei mit dem Software-Tool geöffnet. Für diesen Vorgang wurde der Dateipfad im Einstellungsdialog mit dem Kürzel „ftp://“ und der IP-Adresse des Roboters erweitert. Durch diese Funktionserweiterung kann ein lokales File, wie auch eine Datei am Robotersystem geöffnet werden.

Um den Aufbau und die Struktur des Software-Tools zu dokumentieren wird im nachfolgenden Unterkapitel das Klassendiagramm des Tools erklärt.

6.3 Darstellung der Softwarearchitektur

Jede Klasse in .Net ist von Object abgeleitet. Sie ist die übergeordnete Basisklasse aller Klassen von dem .NET-Framework und stellt den Stamm der Typenhierarchie dar. Ebenso stellt sie abgeleiteten Klassen einfache Dienste zur Verfügung. Object bietet unter anderem die Methode *ToString()* an, die von allen abgeleiteten Klassen verwendet werden kann.

Bei den Klassen GCode, und den Ableitungen GCode_G01 – Gcode_G03 wurde das Prinzip der Vererbung angewandt. Die abgeleiteten Klassen übernehmen dabei die Methoden und Eigenschaften der Basisklasse und erweitern bzw. überschreiben diese je nach Aufgabenstellungen.

Zusätzlich wurde von den Möglichkeiten von Interfaces Gebrauch gemacht. Generell kann man Interfaces einsetzen, wenn man verschiedene Klassen, die nicht voneinander abgeleitet sind, gemeinsam verwenden möchte. Im Software-Tool leiten die Klassen GCode und Comment von *ICodeSegment* ab. Damit ist es möglich beide in die Liste mit den Programmabschnitten einfügen zu können.

Die Klasse Config implementiert *ISerializable* um die Serialisierung über den *BinaryFormatter* zu steuern. Mit Hilfe vom Interface *INotifyPropteryChanged* werden die Controls im Einstellungsdialog über Zustandsänderungen im Config Objekt informiert und damit ein Refresh der Benutzeroberfläche ausgelöst. In der Abb. 56 (zur besseren Darstellung im Querformat) auf der nächsten Seite ist das Klassendiagramm des Software-Tools ersichtlich.

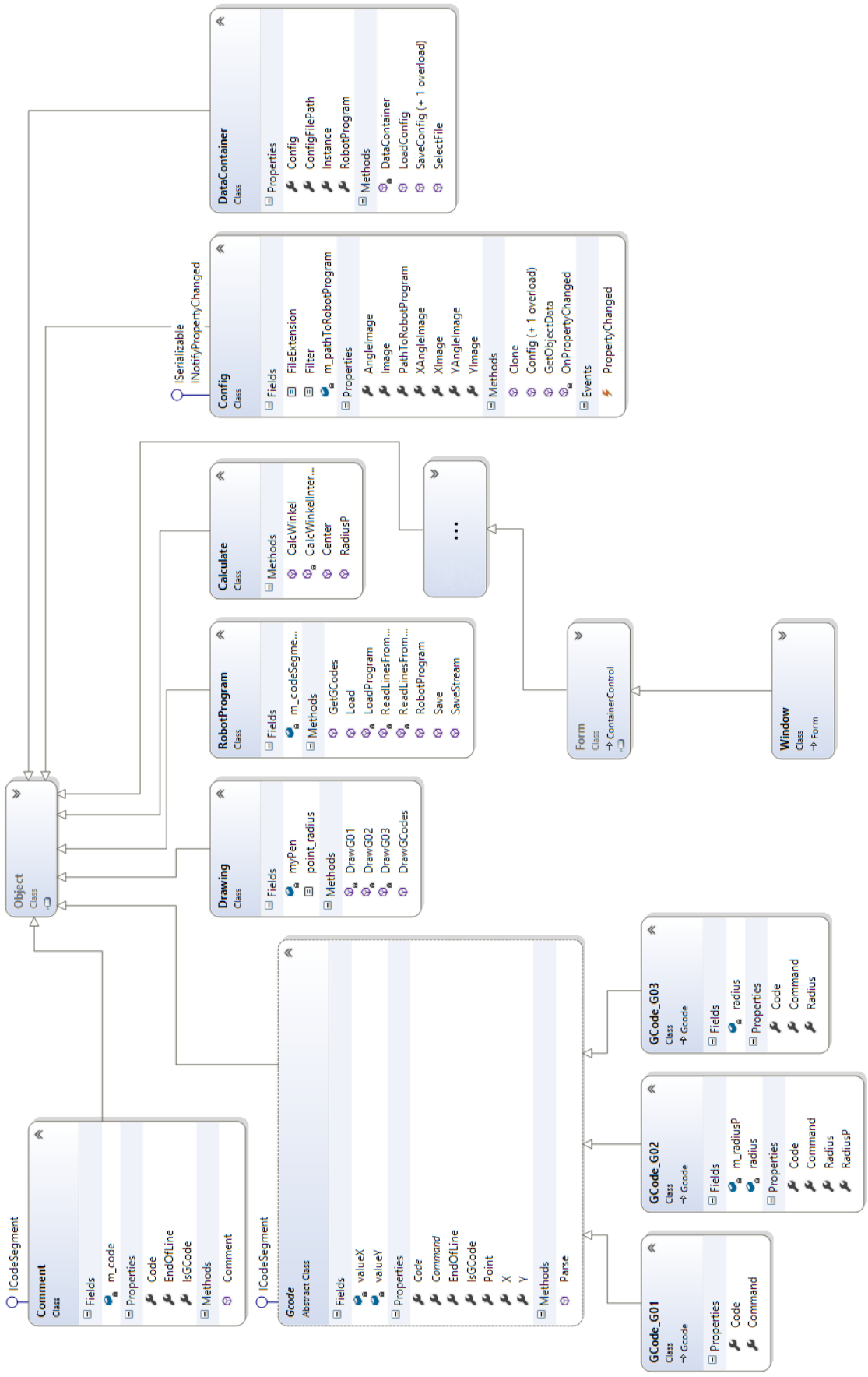

Abb. 56: Darstellung der Softwarearchitektur, Quelle: In Anlehnung an Microsoft Visual Studio 2017.

Im nachfolgenden Kapitel wird die Implementierung des Software-Tools in die Testumgebung dargestellt.

7 TESTUMGEBUNG

Dadurch dass dieser Softwareprototyp auf den G-Code aufgebaut ist, wurde nach einem Robotersystem mit einer CNC-Steuerung als Testumgebung recherchiert. Das Augenmerk lag dabei auf einem Prototyp Roboter der Firma M&R Automation welcher für Handlings- und Dispensieraufgaben entwickelt wurde.

7.1 SCARA Roboter

Die Abkürzung SCARA bedeutet in engl. Selective Compliance Assembly Robot Arm. Dieser Roboterarm besitzt eine Vier-Achs-Kinematik und gehört zu den offenen kinematischen Aufbauarten. Dabei wurden als Achsen, vier elektrische Antriebe eingesetzt. An den Flansch dieses Roboters können verschiedene Endeffektoren montiert werden. Als Endeffektor wird das montierte Werkzeug wie zum Beispiel ein Bauteilgreifer bezeichnet. Gesteuert wird dieser Roboterarm über eine SPS der Firma B&R, welche mit Programmiersprachen der IEC 61131-3, International Electrotechnical Commission (IEC), programmiert werden kann.

Diese Steuerung bildet als Standardapplikation eine SCARA Kinematik ab und kann dadurch wie ein Industrieroboter angesteuert und programmiert werden. Das SPS Programm übernimmt dabei die Steuerung der Achsantriebe und bietet eine Schnittstelle an, über die Roboterprogramme aus Motion-Befehlen eingelesen und verarbeitet werden können. Neben dieser Programmierung mittels Motion-Befehlen kann alternativ der G-Code eingesetzt werden, welche dadurch eine optimale Testumgebung für das Software-Tool darstellt.

7.2 Implementierung

Die Implementierung wurde in zwei Teile unterteilt. Um die Software testen zu können musste zuerst die Testumgebung vorbereitet werden. Dieser Punkt beinhaltet die Erstellung einer Testkontur und die Generierung des dazugehörigen Roboterprogramms. Mit Hilfe dieser Testumgebung konnte das Software-Tool getestet werden.

Vorbereitung der Testumgebung

Als Testkontur wurde eine Zeichnung der Dichtungsfläche eines Getriebedeckels angefertigt. Diese abgebildete Kontur bildet keine originale Darstellung eines Getriebes ab und wurde mit Hilfe eines Computer Aided Design (CAD) Zeichenprogrammes erstellt. Um das Anwenderkoordinatensystem am Roboter anlegen zu können, wurde der Koordinatenursprung mit den dazugehörigen X und Y Richtungen in die Zeichnung mit eingetragen. Zusätzlich wurden zwei Punkte mit einem definierten Abstand für die Bildskalierung eingezeichnet.

Anhand des Bildes wurde ein Roboterprogramm erstellt. Dabei wurden bereits die einzelnen Fahrbefehle mit den dazugehörigen Punkten erzeugt. Durch dieses offline vorbereitete Programm wurde eine Roboterbahn auf dieses Bild gelegt. Zusätzlich zu den Fahrbefehlen beinhaltet der Code noch die Definition des Anwenderkoordinatensystems und Fahrbefehle die sich in der Ebene des Bildes befinden. Darin ist die Grundposition der vier Achsen angelegt, welche als Startposition eines Roboterprogramms herangezogen

wird. Fahrbefehle die sich wie erwähnt nicht in der Arbeitsebene des Roboters befinden, werden für Vorpositionen verwendet, welche vor dem Startpunkt der Bahn eingefügt werden. Durch diese Vorpositionen kann eine Position oberhalb des Startpunktes programmiert werden. Die Positionierung auf den Startpunkt erfolgt dann nur mehr über die Zustellung der Z-Achse.

Das Auftragen einer Dichtmittelkontur wurde durch einen Stift simuliert, der am Roboterflansch befestigt wurde. Somit kann über die Z-Achse eine Zustellung dieses Stiftes auf die Arbeitsebene des Roboters realisiert werden. Als Arbeitsebene wurde eine Platte in den Arbeitsbereich des Roboters montiert, welche die Zeichenfläche darstellte. Dadurch konnte eine ideale Testumgebung für das Roboterprogramm geschaffen werden.

Für den Test des geschriebenen Files wurde dieses über FTP in den Arbeitsordner der Steuerung übertragen. Um das Programm am Roboter zu starten, muss das geladene File angewählt werden. Dieser Vorgang muss über das Bediengerät des Roboters ausgeführt werden. Beim Laden des Files wird die Syntax des geschriebenen Codes überprüft. Dabei traten jedoch Fehler auf, die einen Test des Programmes nicht zuließen. Bei der Fehlersuche stellte sich heraus, dass die Fahrbefehle eines Kreisbogens die Ursache für diese Fehler wahren.

Nach einer Recherche in der G-Code Syntaxbeschreibung der B&R Steuerung konnte ein Unterschied in der jeweiligen Codezeile gefunden werden. Der Fehler lag dabei bei der Radius Komponente des Fahrbefehls G02 und G03. Die Beispiel Codezeile der B&R Programmhilfe unterschied sich dahingegen, dass hinter dem R noch ein = Zeichen stand. Durch das Hinzufügen dieses Ist-Gleich-Zeichens konnte das Programm fehlerfrei angewählt werden.

Diese Änderung des Robotercodes hatte eine Anpassung des Software-Tools zur Folge. Beim Einlesen der einzelnen Codezeilen musste dieses zusätzliche Zeichen mit abgefragt werden. Somit wird für die Generierung der Radius Komponente eines Kreisbogens erst nach diesem Zeichen der tatsächliche Wert eingelesen.

Bevor das Programm jedoch gestartet werden konnte, musste das Anwenderkoordinatensystem definiert werden. Dazu wurde ein 1:1 Ausdruck der generierten Dichtfläche angefertigt und auf der Zeichenplatte befestigt. Der eingezeichnete Koordinatenursprung gibt dabei den Ursprung des Anwenderkoordinatensystems an. Um die Koordinaten dieses Punktes im Weltkoordinatensystems des Roboters zu bestimmen, musste der TCP genau auf diesen Punkt bewegt werden.

Zu diesem Zweck wurde ein Stift am Roboter befestigt. Die Spitze dieses Stiftes stellt dabei den TCP Punkt dar. Das Einlernen dieses Anwenderkoordinatensystems wurde mit Hilfe des Bedienpanel des Roboters realisiert. Die Koordinaten dieses durch das Teach-In-Verfahren bestimmten Punktes, konnten als Ursprung des Anwenderkoordinatensystems ins Roboterprogramm übernommen werden. Mit dieser Vorgehensweise sind alle Punkte der Test Bahn eindeutig in diesem Koordinatensystem definiert.

Mit der Definition des Anwenderkoordinatensystems konnte das Programm getestet werden. Durch das Abfahren des Programmes wurde die Bahn auf die dargestellte Dichtfläche gezeichnet. Das Ergebnis dieses ersten Tests ist in der folgenden Abbildung dargestellt.

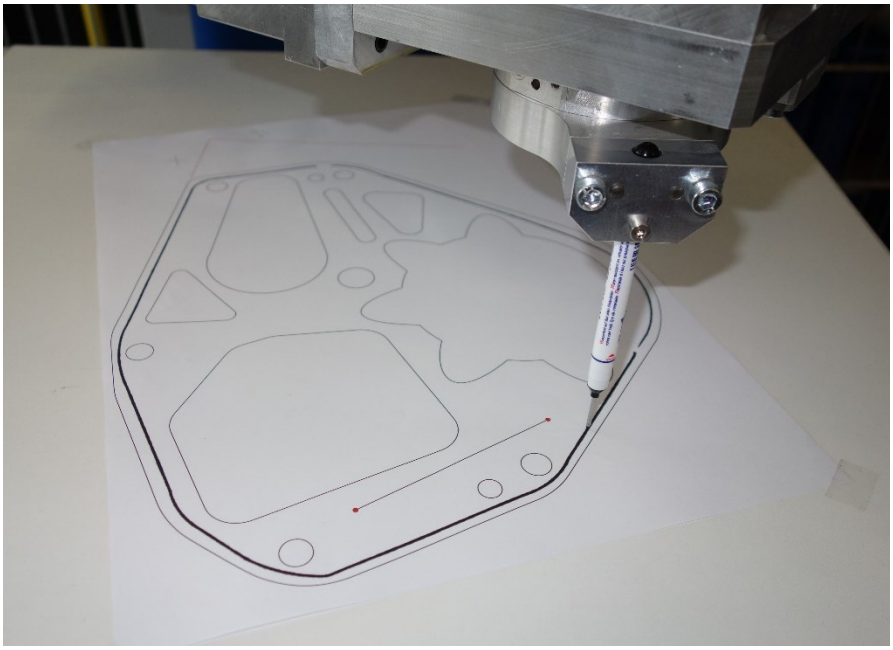

Abb. 57: Ergebnis der ersten Tests am SCARA Roboter, Quelle: Eigene Darstellung.

Durch die Schaffung dieser Testumgebung konnte das vorbereitete Roboterprogramm unter realen Bedingungen getestet werden. Bei einer näheren Betrachtung der gezeichneten Bahn, sind jedoch Nachpositionierungen einzelner Punkte erforderlich, um eine optimale Bahnführung auf der Dichtfläche zu erreichen. Um diese benötigten Adaptierungen durchzuführen wurde mit dem Test des Software-Tools gestartet.

Test des Software-Tools

Durch die implementierte Funktion, das Roboterfile über die FTP Schnittstelle aus dem und in den Roboter zu laden, wurde das bereits lauffähige Roboterprogramm vom Software-Tool eingelesen. Die erstellte Roboterbahn wird im grafischen Anzeigebereich des Tools dargestellt. Bei der Darstellung der Bahn musste jedoch festgestellt werden, dass sich das Roboterkoordinatensystem von dem des Software-Tools unterscheidet.

Betrachtet man die unter dem Abschnitt 3.1.5 und dem im Unterkapitel 5.1 dargestellten Koordinatensysteme, so unterscheidet sich die Richtung der Y-Achse. Durch diese Unterscheidung wird in der Ausgabe des Software-Tool der G03 Fahrbefehl zwar gegen den Urzeigersinn richtig dargestellt, jedoch am Roboter nicht. Diese Richtungsumkehr der Y-Achse hat eine Spiegelung um die X-Achse zur Folge. Zur Lösung dieses Problems wurde im Software-Tool der Koordinatenursprung verändert.

Wie in Unterkapitel 5.1 dargestellt, liegt der originale Koordinatenursprung in der linken oberen Ecke des Ausgabefensters. Die positive Richtung der Y-Achse läuft somit nach unten. Durch die Spiegelung an der X-Achse zeigt diese jedoch nach oben. Die eingelesene Bahn wird somit oberhalb der X-Achse im nicht sichtbaren Teil des Anzeigefensters dargestellt. Durch die Verschiebung des Koordinatenursprungs des Software-Tools an die linke untere Ecke konnte die Bahn wieder angezeigt werden.

Die Spiegelung wurde mit Hilfe der Scale Funktion der Matrix Klasse realisiert. Wird eine Skalierung mit den Parametern eins für X und minus 1 für Y durchgeführt, so ändert sich die Richtung der Y-Achse. Zusätzlich musste der Code für die G02 und G03 Befehle getauscht werden, damit die Richtung dieser Kreisbogen wieder stimmte.

Würde jedoch das Koordinatensystem des Roboters geändert werden, laufen die Kreisbogenfahrbefehle in die falsche Richtung. Aus diesem Grund wurde die vorher beschriebene Vorgehensweise angewandt.

Die somit eingelesene Bahn ist in der nachfolgenden Abbildung dargestellt. Dabei ist auch der neue Koordinatenursprung des Software-Tools und die Skalierungspunkte A und B zu sehen.

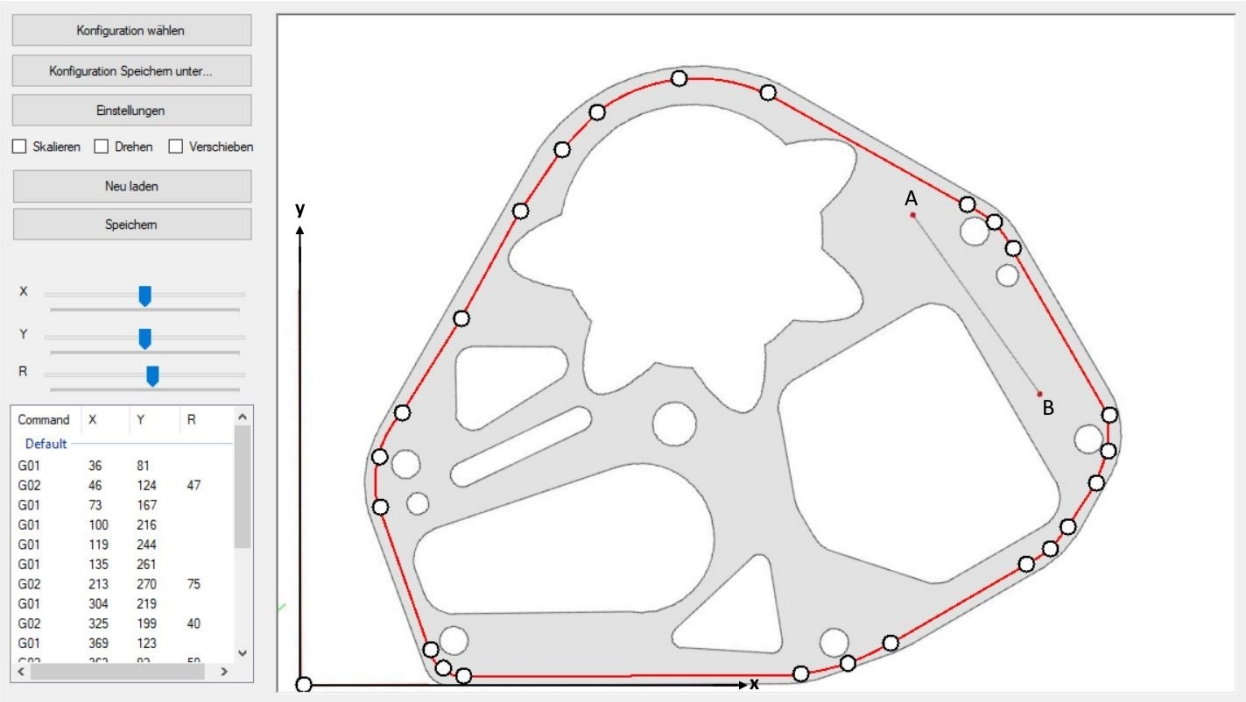

Abb. 58: Oberfläche des Software-Tools, Quelle: Eigene Darstellung.

Für den ersten Adaptierungsversuch wurde die Funktion des Hintergrundbildes nicht mitgetestet. Für den Änderungsversuch wurde ein beliebiger Punkt der dargestellten Bahn geändert. Diese Änderung wurde an den Roboter übertragen. Nach dem erneuten fehlerfreien Anwählen und Starten des Roboters wurde die geänderte Bahn auf die Arbeitsfläche gezeichnet. Durch die Änderung der Position durch das Software-Tool konnte die Änderung somit ohne zusätzliche Eingabe über das Bedienpanel erfolgen. Das Anwählen und Starten des Programmes müssen jedoch von Hand durchgeführt werden.

Ein Vergleich mit dem ersten Zeichenversuch ergab, dass sich außer der adaptierten Position, am restlichen Verhalten des Roboters nichts änderte.

Um die Bahn auf der Dichtfläche zu optimieren, wurde die erstellte Dichtflächengrafik mit dem Tool geöffnet. Zunächst wurde das Hintergrundbild auf die richtige Größe der dargestellten Bahn skaliert. Dazu wurden die beiden Punkte mit Hilfe der Skalierungsfunktion in der Grafik eingetragen. Mit der Eingabe des tatsächlichen Abstandes der beiden Punkte, wurde das Hintergrundbild auf die Größe der Bahn skaliert. Durch die Drehung des Bildes konnte die eingezeichnete X-Achse, der X-Achse des Software-Tool zugeordnet werden. Da die Bahn des eingelesenen Hintergrundbildes nicht mit der Position der

dargestellten Bahn übereinstimmte, wurde der eingezeichnete Koordinatenursprung durch verschieben des Bildes an den Koordinatenursprung des Software-Tools angepasst. Das Optimierungspotential des ersten Tests wurden auch im Anzeigefenster des Tools sichtbar. Mit Hilfe des Software-Tools wurde die Roboterbahn optimiert und erneut auf den Roboter geladen.

Durch die Implementierung des Software-Tools in die Testumgebung konnten alle umgesetzten Funktionen getestet werden.

7.3 Evaluierung

Um die Ergebnisse dieser Arbeit beurteilen zu können, wurde eine Evaluierung durchgeführt. Durch diese werden oftmals Entscheidungen getroffen, die für die Weiterführung eines Projektes ausschlaggebend sind. Auch bei dieser Arbeit stellt sich die Frage, ob durch den Einsatz des Software-Tools eine Zeitersparnis bei der Adaptierung von 2D Roboterbahnen zu erreichen ist und ob es für einen Laien in der Roboterprogrammierung ebenfalls möglich ist, Adaptierungen an einem Roboterprogramm durchzuführen. Weiters können durch eine Evaluierung Daten und Informationen nach einem vorgegebenen Muster erhoben und dokumentiert werden.

Für die Evaluierung wurden drei Aufgabenstellungen vorbereitet, welche zur Beurteilung herangezogen wurden. Als Basis dieser Aufgaben wurde die erstellte Testumgebung in diese Evaluierung mit eingebunden. Durch den Test des Software-Tools wurde ein Anwendungsfall einer Dispensieranwendung simuliert. Die somit entstandene Roboterbahn wurde auf der Zeichenfläche so eingelernt, dass die Bahn des Roboters sich optimal an die Dichtfläche anpasste. Da das Software-Tool zur Adaptierung und Optimierung solcher Bahnsegmente entwickelt wurde, werden bei dieser Evaluierung Aufgabenstellungen angelegt, die eine Änderung der Bahn erforderlich machen. Dazu wurden drei Aufgabenstellung definiert, welche mehreren Testpersonen zur Umsetzung vorgelegt wurden.

Aufgabenstellung 1: Bei dieser Aufgabenstellung geht es um die Optimierung eines G01 Fahrbefehles. Dazu wird ein Punkt von der Roboterbahn absichtlich außerhalb der Dichtfläche der Zeichnung gelegt. Die Testperson hat die Aufgabe diesen Punkt zu ändern und somit eine Optimierung durchzuführen. Der zu ändernde Punkt wird für diese Aufgabenstellung mit dem Buchstaben A bezeichnet. Um die Vergleichbarkeit zu gewährleisten wird auch die Zielposition mit dem Buchstaben B in die Zeichnung der Bahn eingetragen.

Aufgabenstellung 2: Diese Aufgabenstellung beinhaltet die Adaptierung eines Kreisbogen-anfangspunktes. Wie in der Aufgabenstellung 1 wird auch hier ein Punkt der Bahn geändert, um danach eine Optimierung an diesem Punkt durchzuführen. In diesem Fall wird der zu ändernde Punkt mit C bezeichnet und die Sollposition mit D.

Aufgabenstellung 3: Diese Aufgabenstellung wurde erstellt, um das Ergebnis der Adaptierung eines Kreisbogenmittelpunktes aufzuzeichnen. Mit dieser Adaptierung wird der Radius eines G02 oder G03 Fahrbefehls geändert. Der zu ändernde Punkt ist mit den Buchstaben E gekennzeichnet, F markiert die Zielposition.

Als Ausgangssituation wurde der Roboter in die Grundstellung gesteuert. Um diese Aufgabenstellungen an die Testpersonen zu übergeben, wurde für jede einzelne Aufgabenstellung ein Blatt angefertigt. Auf denen sich einerseits eine kurze Beschreibung und anderseits die Zeichnungen mit den vorgegebenen Punkten befindet.

Die Evaluierung des Software-Tools wurde in zwei Methoden unterteilt. Durch die Evaluierung soll einerseits gezeigt werden, ob sich eine Zeitersparnis auch für Roboterprogrammierer einstellt und ob sich diese vorgegebenen Änderungen auch ohne spezielle Kenntnisse durchführen lassen. Dazu werden für beide Bereiche dieselben Aufgabenstellungen genutzt. In den folgenden Unterpunkten wird auf diese zwei Evaluierungsarten eingegangen. In der nachfolgenden Abbildung ist der angefertigte Aufgabenbogen ersichtlich.

Aufgabenstellung der Evaluierung des Software-Tools

Für diese Aufgabenstellungen wurden Roboterprogramme am Roboter der Testumgebung erstellt, durch diese der SCARA Roboter eine Bahn auf die Arbeitsfläche zeichnet. Diese erstellten Programme sind im Arbeitsordner der Robotersteuerung hinterlegt. Ebenso steht die Hintergrundgrafik mit der abgebildeten Dichtfläche für die Adaptierungen zur Verfügung. In den Abbildungen eins bis drei sind die Ausgangssituationen der drei Aufgabenstellungen dargestellt. Um einen optimalen Bahnverlauf zu gewährleisten, sind folgende Adaptierungen durchzuführen.

Aufgabenstellung 1: Ändern des Punktes A auf den Punkt B

Aufgabenstellung 2: Ändern des Kreisbogenanfangspunktes C auf den Punkt D

Aufgabenstellung 3: Ändern des Kreisbogenmittelpunktes E auf den Punkt F

Wird zur Lösung dieser Aufgabenstellung das Software-Tool verwendet, müssen folgende Schritte behandelt werden:

- Öffnen des Robotercodes
- Hintergrundgrafik einfügen und konfigurieren (skalieren, drehen und verschieben)
- Optimierung der Bahn durchführen
- Änderungen speichern und Roboterprogramm neu laden und danach starten

Bei einer Adaptierung des Roboterprogrammes ohne des Software-Tools ist es nur das Roboter Bediengerät zulässig.

Die Aufgabenstellung ist dann erfüllt, wenn der Roboter die Bahn optimiert abfährt.

Abbildungen der Aufgabenstellungen:

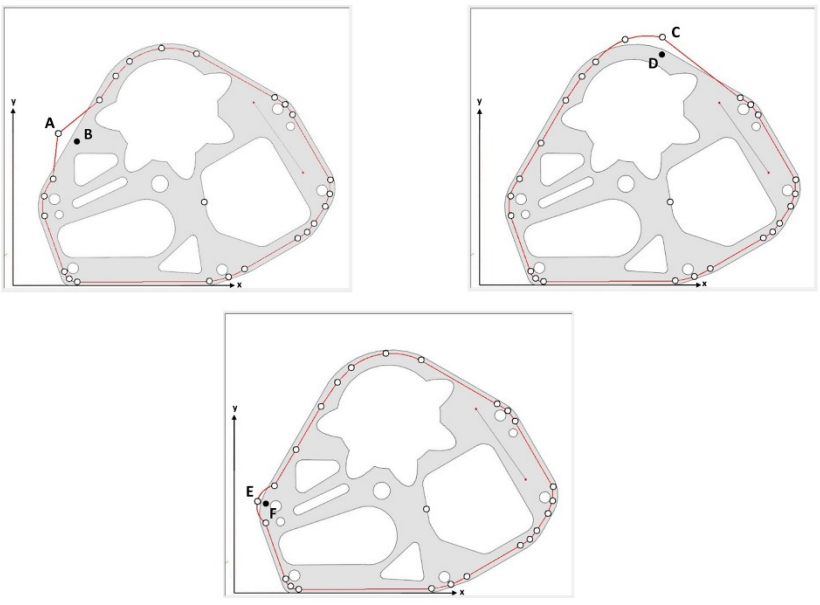

Abb. 59: Screenshot der Aufgabenstellungen, Quelle: Eigene Darstellung.

Szenario 1: Bei der ersten Evaluierungsmethode wurden Testpersonen ausgewählt, welche Erfahrungen im Bereich der Roboterprogrammierung haben. Bei dieser Methode wurden die Testpersonen auf die Funktionen des Software-Tools eingeschult. Ebenso wurden der Testaufbau und das Handling mit dem Roboter ausführlich behandelt.

Um den Unterschied hinsichtlich der Adaptierungszeit zu erhalten, mussten die Aufgabenstellungen auf zwei verschiedene Arten durchgeführt werden. Als erstes musste die Testperson die definierten Änderungen über das Roboterpanel durchführen. Danach wurden die Aufgaben durch das Software-Tool ausgeführt.

Für die Dokumentierung wurde eine Tabelle erstellt wo die einzelnen Ergebnisse der Aufgabenstellungen eingetragen wurden. Die folgenden Beurteilungskriterien wurden dafür definiert:

- **benötigte Zeit:** Hierbei wurde die Zeit gestoppt wie lange die jeweilige Testperson an der Aufgabenstellung arbeitete. Gestoppt wurde von der ersten Eingabe bis hin zur fertigen Lösung.
- **Anzahl der Testanläufe:** Bei einer Adaptierung eines Roboterprogrammes werden bis zum Erhalt der Lösung mehrere Anläufe benötigt. Diese verlängern wiederum die Zeit bis zum vollständigen Ergebnis. Diese Anzahl kann auch als Indikator genutzt werden, wie bedienerfreundlich das jeweilige System ist.
- **Zeitersparnis:** Dieses Kriterium stellt das Ergebnis der jeweiligen Aufgabe dar. Eine positive Zeit spiegelt dabei eine Zeitersparnis des Software-Tools gegenüber der herkömmlichen Änderungsweise dar.

Ergebnis: In der Tabelle 2 sind die Ergebnisse von vier verschiedenen Testpersonen ersichtlich. Erfolgte die Adaptierung über das Roboterpanel so wurde ein erhöhter Zeitbedarf festgestellt. Diese Tatsache beruht darauf, dass die zu adaptierenden Roboterprogramme zuerst gestartet wurden um die zu korrigierenden Punkte zu finden. Dazu wurde das Programm von den Testpersonen im Einzelschritt abgefahren. Im Einzelschrittmodus muss jeder Fahrbefehl des Programmes einzeln gestartet werden. Dadurch konnte der zu ändernde Punkt gefunden werden.

Weiters wurde am Roboter der Teach-In Modus gestartet um den Stift an die korrekte Position zu bringen. Diese Positionen wurden von der jeweiligen Testperson auf ein Blatt Papier geschrieben, um ihn im Positionsanzeigefensters des Roboters der zugehörigen Position zuzuweisen. Diesen Vorgang wiederholten die Testpersonen solange, bis sich der Roboter auf der gewünschten Bahn bewegte. Ohne das Roboterprogramm im Vorhinein abzufahren, konnte die zu adaptierende Position nicht gefunden werden.

Im Gegensatz zur herkömmlichen Adaptierung des Programmes mit dem Roboterpanel entfällt dieser Schritt bei der Verwendung des Software-Tools. Durch die Visualisierung der Bahn und des konfigurierten Hintergrundbildes, konnte das Optimierungspotential auf einen Blick erkannt werden. Auch diese Tatsache schlug sich in der Adaptierungszeit negativ nieder.

Wurde die Änderung durch das Roboterpanel durchgeführt, musste sich die Testperson einen Überblick über das vorhandene Anwenderkoordinatensystem machen. Denn durch das Teach-In Verfahren musste der Roboter in diesem Koordinatensystem bewegt und positioniert werden. Durch die Verwendung des Software-Tools konnten diese Änderungen der Bahn intuitiv durchgeführt werden.

Dieser Teil der Evaluierung zeigte, dass durch die Verwendung des Roboterpanels und des Software-Tools dasselbe Ergebnis erzielt werden kann. Jedoch kann für diese Adaptierung einer 2D Roboterbahn der Einsatz des Software-Tools eine Zeitersparnis erzielt werden. Die Ergebnisse dieser Evaluierung sind in folgender Tabelle festgehalten.

	mit Roboterpanel		mit SoftwareTool		
Aufgabenstellung	**benötigte Zeit**	**Anzahl der Testanläufe**	**benötigte Zeit**	**Anzahl der Testanläufe**	**Zeitersparnis**
Ändern des Punktes A auf den Punkt B	7 min 40 s	2	2 min 43 s	1	**4 min 57 s**
Ändern des Kreisbogenanfangspunktes C auf den Punkt D	8 min 5 s	3	2 min 55s	1	**5 min 10 s**
Ändern des Kreisbogenmittelpunktes E auf den Punkt F	7 min 40 s	1	3 min 4 s	1	**4 min 36s**

Tabelle 2: Ergebnis der Evaluierung, Vergleich Roboterpanel mit Software-Tool, Quelle: Eigene Darstellung.

Szenario 2: Bei der zweiten Evaluierungsmethode wurden Testpersonen ausgewählt, die keinerlei Erfahrungen in der Roboterprogrammierung haben. Diese Personen wurden nur auf das Software-Tool eingeschult, dies dauerte in der Regel 5 Minuten. Erst danach wurden ihnen die Aufgabenstellungen für die Evaluierung gezeigt. Auch hier wurde die Zeit jeder einzelnen Aufgabe gestoppt und aufgezeichnet. Die Aufgabenstellungen für dieses Evaluierungsszenario sind dieselben wie zuvor beim Szenario 1.

Ergebnis: Das Einlesen des Roboterprogrammes und die Konfiguration der Hintergrundgrafik stellte keine Schwierigkeit für die Testpersonen dar. Diese konnten ohne zusätzliche Hilfestellungen die Aufgabenstellung bewältigen. Wie in der Tabelle 3 ersichtlich, konnte das Ziel der Aufgabenstellungen in fast derselben Zeit erreicht werden wie bei den Testpersonen mit Roboterkenntnissen. Weiters wurde keine Schulung des Robotersystems benötigt, um die vorgegebenen Aufgabenstellungen zu lösen. Auch hier wurde beim ersten Adaptierungsversuch das Ziel bereits erreicht, wie auch die nachfolgende Tabelle zeigt.

	mit SoftwareTool	
Aufgabenstellung	**benötigte Zeit**	**Anzahl der Testanläufe**
Ändern des Punktes A auf den Punkt B	3 min 25 s	1
Ändern des Kreisbogenanfangspunktes C auf den Punkt D	2 min 33 s	1
Ändern des Kreisbogenmittelpunktes E auf den Punkt F	2 min 31 s	1

Tabelle 3: Ergebnis der Evaluierung, Testpersonen ohne Vorkenntnisse, Quelle: Eigene Darstellung.

8 ERGEBNISSE & DISKUSSION

Durch die Einarbeitungsphase im theoretischen Teil und der definierten Zielsetzung konnte durch die Programmierung ein Tool geschaffen werden, das auf die Adaptierung von 2D Bahnsteuerungen spezialisiert ist. Ebenfalls wird der eingelesene Robotercode so verarbeitet, dass davon nur mehr die Informationen verwendet werden, die für die Darstellung der Bahn benötigt werden. Somit wird auch ein ungewolltes Ändern von Positionen des restlichen Roboterprogrammes unterbunden.

Weiters lässt sich durch das Hintergrundbild eine grafische Beziehung zwischen der Roboterbahn und dem zu bearbeiteten Bauteils herstellen. Einen weiteren Vorteil bringt die Kommunikation über die FTP Verbindung, die eine Änderung von Roboterprogrammen direkt am Robotersystem durchführen lässt. Durch die Implementierung einer speicherbaren Konfiguration, können nachfolgende Adaptierungen eines bereits optimierten Programmes ohne erhöhten Konfigurationsaufwand bewerkstelligt werden.

Für den weiteren Einsatz dieses Tools wird eine Implementierung einer weiteren Roboterprogrammiersprache angedacht. Dadurch ließe sich das Tool universeller einsetzen. Bei der Einrichtung der Testumgebung, insbesondere beim Anlegen des Roboterprogrammes, wurde ein weiteres Optimierungspotential gefunden. Dadurch, dass immer ein Fahrbefehl mit dem nachfolgenden Fahrbefehl verbunden wird, konnte bei der Erstellung des Roboterprogrammes immer derselbe Fahrbefehl mit den gleichen Positionen eingefügt werden. So lagen diese Positionen beim Einlesen in das Software-Tool übereinander und konnten durch das Tool an die richtige Stelle platziert werden.

Aus diesem Ablauf heraus könnte eine Codegenerierungs-Funktion entstehen. Ebenso wäre es sinnvoll die Bearbeitung der Hintergrundgrafik zu erweitern, um verzerrte Bilder des zu bearbeitenden Bauteils für den Bezug zwischen Bahn und Bauteil herzustellen.

Mit etwas Abstand betrachtet lässt sich nun sagen, dass man durch die Verwendung des Software-Tools eine Verkürzung der Arbeitszeit erzielen kann. Ebenso wird es ermöglicht auch ohne spezielle Roboterkenntnisse eine Adaptierung an einer Roboterbahn durchzuführen. Dies stellt einen gelungenen Erfolg der implementierten Arbeit dar.

LITERATURVERZEICHNIS

Gedruckte Werke (13)

ABB (Hrsg.) (2004-2016): *Technisches Referenzhandbuch RAPID Instruktionen, Funktionen und Datentypen*, D Auflage

ABB (Hrsg.) (2004-2016): *Technisches Referenzhandbuch RAPID Überblick*, D Auflage

ABB (Hrsg.) (2016): *Bedienungsanleitung IRC5 mit FlexPendant*, C Auflage

Auto-Leebmann GmbH - BMW / MINI Vertragshändler (2017): *leebmann24* https://www.leebmann24.de/sechskantschraube-m10x70-3er-07119921506.html [Stand: 12.11.2017]

Bartenschlager, Jörg; Hebel, Hans; Schmidt, Georg (1998): *Handhabungstechnik mit Robotertechnik*, Vieweg, Wiesbaden

Bayer, Jürgen (2010): *Das C# 2010 Codebook*, Addison-Wesley Verlag, München

Böge, Alfred; Böge, Wolfgang (Hrsg.) (2014): *Handbuch Maschinenbau*, 22 Auflage, Springer

Booch, Grady; Rumbaugh, James; Jacobson, Ivar (2006): *Das UML Benutzerhandbuch*, Addison-Wesley Verlag

Doberenz, Walter; Gewinnus, Thomas (2013): *Visual C# 2012 Grundlagen und Profiwissen*, 1 Auflage, Carl Hanser Verlag, München

Ernst, Hartmut; Schmidt, Jochen; Beneken, Gerd (2016): *Grundkurs Informatik*, 6 Auflage, Springer Vieweg, Wiesbaden

Fetzer, Albert; Fränkel, Heiner (2012): *Mathematik 1, Lehrbuch für ingenieurwissenschaftliche Studiengänge*, 11 Auflage, Springer, Berlin

Fritsch, Thomas (2017): *www.foundry-planet.com* https://www.foundry-planet.com/de/equipment/detailansicht/2881/?cHash=2015a5b3177a191f50ae0c780205a419 [Stand: 13.Oktober.2017]

Hesse, Stefan (2000): *Fertigungsautomatisierung*, Vieweg, Wiesbaden

Kief, Hans B.; Roschiwal, Helmut A.; Schwarz, Karsten (2017): *CNC Handbuch*, 30 Auflage, Carl Hanser Verlag, München

Microsoft (2017): *msdn.microsoft.com* https://msdn.microsoft.com/de-de/library/8667dchf(v=vs.110).aspx [Stand: 17.Oktober.2017]

Microsoft (2017): *msdn.Microsoft.com* https://msdn.microsoft.com/de-de/library/system.io.stream(v=vs.110).aspx [Stand: 26.November.2017]

Online-Quellen (6)

Rumpe, Bernhard (2012): *Agile Modellierung mit UML*, 2 Auflage, Springer Vieweg, Aachen

Stephens, Rod (2017): *csharphelper*
http://csharphelper.com/blog/2014/10/draw-and-move-line-segments-in-c/ [Stand: 14.November.2017]

Streetlights Vertriebs GmbH (2010): *streetlights*
http://www.streetlights.de/9014190_Getriebedeckel_Original.html [Stand: 30.November.2017]

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

CAD	Computer Aided Design
CLR	Common Language Runtime
CNC	Computerized Numerical Control
FTP	File Transfer Protocol
IEC	International Electrotechnical Commission
JIT-Compiler	Just in Time Compiler
MoveC	Move Circular
MoveL	Move Linear
MSIL-Code	Microsoft Intermediate Language Code
NC	Numerical Control
PtP	Point to Point
SCARA	Selective Compliance Assembly Robot Arm
SPS	Speicher Programmierbare Steuerung
TCP	Tool Center Point
UML	Unified Modeling Language

ANHANG

Umfragebogen Benutzerfreundlichkeit bestehender Robotersoftware-Pakete:

Befragung Benutzerfreundlichkeit Robotersoftware Tools

Unternehmen:	
Position im Unternehmen:	

	ABB Robot Studio				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

	Kuka WorkVisual				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

	Fanuc ROBOGUIDE				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

	Hirata BASIC Development Enviroment				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

	Siemens Robot Expert				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

Befragung Benutzerfreundlichkeit von RobotersoftwareTools Oktober 2017.xlsx

	Stäubli VAL 3				
Die Robotersoftware ...	trifft nicht zu	trifft eher nicht zu	trifft eher zu	trifft zu	keine Erfahrung
ist intuitiv Bedienbar					
bietet eine ausreichende Dokumentation					
ist einfach verständlich					
lässt eine effektive Arbeitsweise zu					
lässt eine Änderung am laufenden Robotersystem zeitoptimiert durchführen					

Befragung Benutzerfreundlichkeit von RobotersoftwareTools Oktober 2017.xlsx