TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Diplomarbeit

Bauteilprogrammierung an einer 5-Achs-CNC Werkzeugmaschine im Konstruktionskoordinatensystem

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs (Dipl.-Ing. oder DI), eingereicht an der TU Wien, Fakultät für Maschinenwesen und Betriebswissenschaften, von

Bernhard Josef MAYRHOFER-HUBER

Mat.Nr.: 01226770

unter der Leitung von

Univ.-Prof. Dipl. Ing. Dr.techn Burkhard KITTL

Institut für Fertigungstechnik und Photonische Technologien, E311

Wien, Nov 2021

Ich nehme zur Kenntnis, dass ich zur Drucklegung dieser Arbeit nur mit Bewilligung der Prüfungskommission berechtigt bin.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht. Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein. Ich nehme zur Kenntnis, dass die vorgelegte Arbeit mit geeigneten und dem derzeitigen Stand der Technik entsprechenden Mitteln (Plagiat-Erkennungssoftware) elektronisch-technisch überprüft wird. Dies stellt einerseits sicher, dass bei der Erstellung der vorgelegten Arbeit die hohen Qualitätsvorgaben im Rahmen der geltenden Regeln zur Sicherung guter wissenschaftlicher Praxis „Code of Conduct“ an der TU Wien eingehalten wurden. Zum anderen werden durch einen Abgleich mit anderen studentischen Abschlussarbeiten Verletzungen meines persönlichen Urheberrechts vermieden.

Stadt und Datum

Unterschrift

Danksagungen

Ich möchte mich herzlichst bei meinen Eltern bedanken, mit deren Unterstützung dies erst möglich war. Des Weiteren möchte ich mich bei meinen Geschwistern für die Unterstützung bedanken. Bei Herrn Kettl Gerald und Herrn Wimmer Andreas möchte ich mich für die Betreuung von Seiten der Firma Fill GmbH bedanken. Außerdem möchte ich mich bei meinem Betreuer an der Technischen Universität Wien Kittl Burkhard bedanken.

Inhaltsverzeichnis

English Abstract

The Diploma thesis describe the transformation of coordinates at a machine tool. It is used by a machine operator to correct values in another coordinatesystem at a machine tool. The correction happens at the drifts in the NC-program. Drifts simplity NC-programming and reading of NC-programs. I used a Sinumerk 840 sl control system from Siemens, which offers the machine manufacturer its own extension in the form of an application. This application is programmed in C++ and using the Qt library for the Graphic User Interface (GUI). I use the JavaScript Object Notation (JSON) data format to store information. The application runs on the Human Machine Interface (HMI). This application makes it easier for the machine operator to make corrections to the machine tool. This has the consequence that less mistakes happen by correction.

Deutsche Kurzfassung

In dieser Diplomarbeit beschreibe ich, wie eine Koordinatentransformation an einer Werkzeugmaschine stattfinden kann. Diese Transformation benötigt der_die Maschinenbediener_in, um Korrekturarbeiten aus einem anderen Koordinatensystem als dem Maschinenkoordinatensystem an der Werkzeugmaschine vorzunehmen. Dabei erfolgen die Korrekturen an den Nullpunktverschiebungen im NC-Programm. Nullpunktverschiebungen erleichtern die NC-Programmierung und das Lesen der NC-Programme. Als Steuerung nütze ich eine Sinumerk 840 sl von Siemens, welche dem Maschinenhersteller eine eigene Software–Erweiterung in Form von einer Anwendung bietet. Diese Anwendung wird in C++ mit Hilfe der Qt Bibliothek für das Gaphic User Interface (GUI) geschrieben. Für das Speichern von Information nütze ich das JavaScript Object Notation (JSON) Datenformat. Die Anwendung läuft auf dem Human Machine Interface (HMI). Durch diese Anwendung kann der_die Maschinenbediener_in einfacher Korrekturen an der Werkzeugmaschine vornehmen. Dies hat zur Folge, dass weniger Fehler beim Korrigieren geschehen.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BTKS	Bauteilkoordinatensystem
BTNP	Bauteilnullpunkt
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computerized Numerical Control
GUI	Graphic User Interface
HMI	Human Machine Interface
INI	Initialisierungsdatei
JSON	JavaScript Object Notation
KKS	Konstruktionskoordinatensystem
KS	Koordinatensystem
LKW	Lastkraftwagen
M	Maschinennullpunkt
MessKS	Messkoordinatensystem
MKS	Maschinenkoordinatensystem
N	Werkzeugaufnahmepunkt
NC	Numerical Control
NCK	Numerical Control Kernel
NCU	Numerical Control Unit
NPV	Nullpunktverschiebung
R	Referenzpunkt
T	Werkzeugträgerbezugspunkt
W	Werkstücknullpunkt
WKS	Werkstückkoordinatensystem
XML	Extensible Markup Language

1. Einleitung

Um ein Werkstück in Serie zu fertigen, erstellt der_die Konstrukteur_in unter anderem eine Computer Aided Design (CAD)-Zeichnung des zu fertigenden Werkstücks. Dabei nützt der_die Konstrukteur_in ein frei wählbares Koordinatensystem. In der Arbeitsvorbereitung beschreibt ein_e Programmierer_in die Bearbeitung mit einem Computer Aided Manufacturing (CAM)-System und erstellt das dazu gehörende Numerical Control (NC)-Programm für die Maschine. Dieses NC-Programm bezieht sich auf das Koordinatensystem der Maschine und nicht mehr auf das Koordinatensystem der Konstruktion. In den meisten Fällen müssen an der Maschinen Korrekturen im NC-Programm durchgeführt werden. Dafür wird ein Werkstück zum Prüfen gefertigt. Ein_e Mitarbeiter_in vermisst das Werkstück im Konstruktionskoordinatensystem und kontrolliert, ob alle Maße in den Toleranzen liegen. Die Abweichungen korrigiert der_die NC-Programmierer_in anschließend im NC-Programm. Danach wird das nächste Werkstück gefertigt. Dieser Vorgang wiederholt sich so lange, bis alle Toleranzen eingehalten werden. Außerdem erfolgt bei größeren Stückzahlen in regelmäßigen Abständen eine zusätzliche Kontrolle als Qualitätssicherung. Diese zusätzliche Messung läuft ebenfalls wie oben beschrieben ab.

In dieser Diplomarbeit beschäftige ich mich mit der Koordinatentransformation vom Konstruktionskoordinatensystem ins Maschinenkoordinatensystem. Beim Erstellen des NC-Programms übernimmt das CAM-System die Transformation. Die Korrekturwerte vom Vermessen liegen aber wieder im Konstruktionskoordinatensystem vor und eine neue Transformation ist nötig. Nimmt der_die NC-Programmierer_in die Korrekturen im CAM-System vor, erstellt das System wieder ein neues NC-Programm. Das hat den Nachteil, dass das Korrigieren im CAM-System im Vergleich zur Korrektur direkt im NC-Programm viel Zeit und Ressourcen benötigt. Bei einer Korrektur direkt im NC-Programm müssen die Korrekturwerte zuvor in das Maschinenkoordinatensystem transformiert werden. Das macht der_die Maschinenbediener_in meist intuitiv, in manchen Fällen wird eine Vorlage benützt und im besten Fall steht eine Tabelle zum Umrechnen zur Verfügung. Ziel dieser Arbeit ist eine einfachere Transformation der Korrekturen in das Maschinenkoordinatensystem direkt an

der Werkzeugmaschine.

Für die Transformation schreibe ich ein Programm in C++, das auf der Steuerung der Maschine läuft. Zur Umsetzung nutze ich die Sinumerk-Steuerung von Siemens. Diese Steuerung bietet die Möglichkeit, eigene Erweiterungen in Form von Anwendungen zu erstellen, die auf der Steuerung laufen und vereinfacht das Korrigieren. In dieser Anwendung soll die Transformation vom Konstruktionskoordinatensystem in das Maschinenkoordinatensystem stattfinden, sodass der_die Maschinenbediener_in nur noch die Korrekturwerte eintragen muss. Des Weiteren nutze ich die Nullpunktverschiebung im NC-Programm, welche ein übersichtliches Programmieren ermöglicht. Zudem bezieht sich jede Bearbeitung auf eine NPV, wodurch die Korrektur direkt an den NPVs erfolgt.

In dieser Diplomarbeit wird nicht auf eine automatische Messdatenrückführung eingegangen. Automatische Korrekturen könnten ohne vollständige Überprüfungen erheblichen Schaden an der Werkzeugmaschine verursachen. Des Weiteren werden Korrekturen nur mittels Nullpunktverschiebungen vorgenommen und nicht an den Spindelpositionen der Werkzeugmaschine. Dafür müsste die Anwendung das NC-Programm interpretieren können. Außerdem verzichte ich auf eine Transformation während der Bearbeitung. Das macht die Sinumerik-Steuerung bereits mit den Befehlen „Cyclus800“ und „TRAORI“. Diese Befehle funktionieren jedoch nur mit Werkzeugmaschinen, die nur eine Spindel haben.

Für diese Diplomarbeit erstelle ich mittels C++ und der Qt Bibliothek eine Anwendung auf der Sinumerik. Zuerst erfolgt der Aufbau des Graphic User Interface (GUI), um die NC-Programme einzulesen. Außerdem benutze ich das GUI, um die Korrekturwerte von dem_r Maschinenbediener_in zu erhalten. Danach beschreibe ich, wie die Umrechnung im Hintergrund der Anwendung erfolgt. Die Ergebnisse werden am Bildschirm für den_die Maschinenbediener_in angezeigt. Des Weiteren verwende ich noch ein JavaScript Object Notation (JSON) Datenformat, um die Information zu speichern. Als Ergebnis gebe ich der Maschine ein NC-Programm mit den korrigierten Nullpunktverschiebungen zurück.

2. Grundlagen

2.1. Fertigungstechnischer Hintergrund

2.1.1. 5-Achsen Werkzeugmaschine

In der DIN 69651 ist die Werkzeugmaschine folgendermaßen definiert:
„Mechanisierte und mehr oder weniger automatisierte Fertigungseinrichtung, die durch relative Bewegung zwischen Werkstück und Werkzeug eine vorgegebene Form am Werkstück oder eine Veränderung einer vorgegebenen Form an einem Werkstück erzeugt.“[1]

Regel definiert die Werkzeugmaschine im Buch „Werkzeugmaschine“ so:
„Unter einer Werkzeugmaschine sind alle Maschinen zu verstehen, die der Fertigung mechanischer Komponenten definierter, reproduzierbarer Form mit Hilfe von Werkzeugen dienen. Die Formgebung geschieht durch eine mechanisiert angetriebene und geführte Relativbewegung zwischen Werkzeug und Werkstück, die sich in Prozess- und Vorschubbewegung unterteilen lässt. Realisiert werden diese durch eine Energiezufuhr elektrischer, hydraulischer, pneumatischer und nicht ausschließlich manueller Art.“[2]

Bahmann W. beschreibt die Werkzeugmaschine im Buch „Werkzeugmaschinen kompakt“ wie folgt:
„Die Werkzeugmaschine (auch als Fertigungsmittel oder Fertigungseinrichtung bezeichnet) dient der Erzeugung von Werkstücken mittels Werkzeugen entsprechend der gegebenen Fertigungsaufgabe. Die Werkzeugmaschine gibt dem Werkstoff durch Urformende, umformende, trennende und/oder fügende Verfahren die geforderte geometrische Form und Oberflächengestalt sowie die gewünschten Abmessungen.“ [3]

Von einer Werkzeugmaschine wird gesprochen, wenn folgende Eigenschaften erfüllt sind [1][2][4][5][6]:

- die Maschine benötigt Energie in Form von Strom, Hydraulik oder Pneumatik

- die Maschine nützt Verfahren der Fertigungstechnik zur Bearbeitung eines Werkstücks
- die Maschine verwendet Werkzeuge zur Bearbeitung des Werkstücks
- eine relative Bewegung findet zwischen Werkzeug und Werkstück statt
- die gefertigten Werkstücke sind Teile von Baugruppen

Zum dreidimensionalen Bearbeiten eines Werkstückes benötigt man drei unabhängige Achsen. Diese sind die x-, y- und z-Achse. Bei komplexeren Werkstücken reichen drei Achsen nicht mehr aus, die Maschine muss das Werkstück oder Werkzeug in die zu bearbeitende Ebene drehen. Das Werkstück oder die Werkzeugachse rotiert dabei um die A- und B-Achse. Die A-Achse liegt in der x-Achse, die B-Achse liegt in der y-Achse und die C-Achse liegt in der z-Achse. Mit zwei drehenden Achsen bearbeitet die Maschine das Werkstück an fünf Seiten in drei Dimensionen. Für die letzte Seite platziert der_die Maschinenbediener_in das Werkstück neu in der Maschine. Mit den Rotationsachsen A und B besitzt die Maschine fünf Achsen. [3]

2.1.2. Numerische Steuerung

Eine Numerische Steuerung wird mit NC abgekürzt und kommt vom amerikanischen „Numerical Control“. Die Steuerung ist für die Bewegung, Positionierung und die Drehzahl der Spindel sowie für die Vorschubgeschwindigkeit und Hilfsfunktionen zuständig. Moderne NCs sind mit einen Mikrorechner ausgestattet. Dieser wertet die Signale von Positions-, Drehwinkel- und Zustandssensoren aus. Damit kann die NC ihren Ist-Wert mit dem Soll-Wert vergleichen und die Position genauer regeln. Eine CNC kann auch mehrere Achsen gleichzeitig regeln. Solche NCs werden als Computerunterstützte Numerische Steuerung (CNC) bezeichnet, was sich vom amerikanischen „Computerized Numerical Control“ ableitet. Die Steuerung verarbeitet Befehle in Form von Zahlen und Buchstaben. Diese sind in einem NC-Programm zusammengefasst und werden von der Steuerung abgearbeitet. Die NC-Programme sind auf der Steuerung gespeichert und gelangen über einen Datenträger oder einen Netzanschluss in die Maschine. [4][7][3][4]

2.1.3. CAD/CAM-Programm

Das Schreiben von NC-Programmen ist für die meisten industriell hergestellten Teile sehr aufwendig. Darum wurde die rechnergestützte Programmierung entwickelt. Hierfür wird die Geometrie aus den meist vorhandenen „Computer Aided Design“ (CAD)-Zeichnungen

genützt. Ein_eine Programmierer_in legt die Bearbeitungsstrategie fest, erstellt eine Werkzeugliste und wählt die Maschine mit deren Kinematik aus. Dazu nützt der_die Programmierer_in eine Computer Aided Manufacturing (CAM)-Software, welche oft schon im CAD-Programm integriert ist. Aus der angegebenen Information erstellt die CAM-Software die NC-Programme und die Werkzeugliste. Oft existiert ein Hauptprogramm mit den Bearbeitungen und Unterprogrammen, die die Nullpunktverschiebungen, die Korrekturwerte und die Zyklen von wiederkehrenden Merkmalen beinhalten. [7][8][9]

2.1.4. Bezugssysteme

Damit Steuerung und Maschine zusammenarbeiten können, benötigen sie ein gemeinsames Bezugssystem. Dieses besteht aus einem Bezugspunkt und einem Koordinatensystem. Dabei ist der Koordinatenursprung meist der Bezugspunkt.[4]

Bezugspunkt

Eine Werkzeugmaschine besitzt meist mehrere Bezugspunkte, wie z.B. Referenzpunkt R, Werkzeugträgerbezugspunkt T oder Werkzeugaufnahmepunkt N.

Die hier wichtigsten Punkte sind [7][4]:

- **Maschinennullpunkt M:** Der Maschinennullpunkt ist der Ursprung des Maschinenkoordinatensystems. An diesem Punkt orientiert sich die Steuerung. Jeder andere Punkt bezieht sich auf diesen Punkt. Der Maschinenhersteller legt den Maschinennullpunkt fest, danach ist eine Änderung nicht mehr möglich.
- **Werkstücknullpunkt W (Bauteilnullpunkt BTNP):** Der Werkstücknullpunkt, oder auch Bauteilnullpunkt genannt, ist der Ursprung des Werkstückkoordinatensystems. Im Programm ist dieser mit einer Nullpunktverschiebung wählbar. Bei Frästeilen liegt der Werkstücknullpunkt oft in einer Ecke oder in der Symmetrieebene des Werkstücks.
- **Nullpunktverschiebung NPV:** Die NPV liegt meist auf der Oberfläche des Werkstücks und ist nach der Bearbeitung ausgerichtet. Hier wird der Verfahrensweg nur in eine Richtung programmiert. Bei gedrehten Werkstücken oder schrägen Flächen vereinfacht sich somit das NC-Programm.

Koordinatensystem

Um die Zerspanungsbewegungen einer Werkzeugmaschine festzulegen, ist ein Koordinatensystem notwendig. Die DIN 66217 [10] schreibt ein rechtwinkeliges nach rechts drehendes Koordinatensystem mit den Achsen x, y und z vor. [7]

Dabei kennt die Maschine folgende Koordinatensysteme:

- **Maschinenkoordinatensystem MKS:** Das Koordinatensystem muss den Maschinenachsen zugeordnet werden, um die Bewegungen in einer CNC-Maschine programmieren zu können. Dabei ist die z-Achse die Spindelachse und zeigt vom Werkstück zum Werkzeug. Die x-Achse liegt parallel zur Aufspannfläche des Werkstücks und ist meistens in horizontaler Richtung angeordnet. Aus der Rechtsdrehung ergibt sich die Richtung der y-Achse.
- **Werkstückkoordinatensystem WKS (Bauteilkoordinatensystem BTKS):** Dieses Koordinatensystem hat den Ursprung im Werkstücknullpunkt. Die Achsen verlaufen entlang der Kanten oder Symmetrieebenen des Werkstücks.
- **Nullpunktverschiebung NPV:** Ihr Ursprung ist meist durch ein Merkmal definiert. Bei der NPV liegt die z-Achse meist parallel zur Spindelachse. Die x- und y-Achse liegen auf der Werkstückfläche oder einer Symmetrieebene und sind von der Bearbeitung her definiert.

Konstruktionskoordinatensystem (KKS)

Auch außerhalb der Werkzeugmaschine betrachtet man das Werkstück in Koordinatensystemen. Im CAD-Programm zeichnet der_die Kontrolleur_in das Bauteil in einem eigenen Koordinatensystem. Dieses Koordinatensystem ist oft nicht identisch mit dem späteren Koordinatensystem der Maschine. Die Bauteile werden zu Baugruppen zusammengefügt. Dabei benötigt die Baugruppe wieder ein eigenes Koordinatensystem, auf welches sich die Positionen und Lagen der einzelnen Bauteile beziehen. Dieses Koordinatensystem wird als Konstruktionskoordinatensystem (KKS) bezeichnet. Das KKS kann außerhalb der Baugruppe liegen und unabhängig von allen Koordinatensystemen in der Werkzeugmaschine sein.

2.2. Programmierhintergrund

2.2.1. C++

Die Programmiersprache C++ wurde 1979 von Bjarne Stoustup bei AT&T entwickelt und ist die Erweiterung der Programmiersprache C. C++ ermöglicht eine effiziente maschinennahe Programmierung. Diese wird für Betriebssysteme, eingebettete Systeme, virtuelle Maschinen, Treiber und Signalprozesse benötigt. Außerdem können auch Anwendungsprogramme mit hohem Abstraktionsniveau objektorientiert geschrieben werden. Diese kommen zum Einsatz, wenn hohe Anforderungen an die Effizienz gestellt werden oder die Leistung der vorhandenen Hardware begrenzt ist. Die Sprache C++ hat einen überschaubaren Befehlssatz (Sprachkern), dessen Funktionalität durch eine hierarchisch strukturierte Standardbibliothek erweitert wird. In den Header-Dateien definiert man globale Variablen, Funktionen, Methoden und Unterprogramme. Im Hauptprogramm können diese Header-Dateien und Standardbibliotheken eingebunden werden.[11]

2.2.2. Qt

Qt ist eine Klassenbibliothek für graphische Oberflächen. Dabei kann die Bibliothek von verschiedenen Programmiersprachen genutzt werden. Die norwegische Firma Trolltech entwickelte Qt, sie wurde aber 2008 von der finnischen Firma Nokia übernommen und 2011 von Digia Qt gekauft. Neben der kommerziellen Version steht auch eine kostenlose Version zur Verfügung. Zum Erstellen eines C++ Programms mit Qt benötigt man die Qt-Klassenbibliothek und einen unterstützenden C++-Compiler.
Für die graphische Oberfläche steht der Qt-Designer zur Verfügung. Dieser erstellt eine XML-Datei im User-Interface-Format „.ui“. Qt bietet auch eigene Klassen, Konstruktoren sowie öffentliche und private Funktionen. Der Quellcode unterscheidet sich jedoch erheblich vom standardisierten C++. Das beginnt schon beim Einbinden von Qt-spezifischen Header-Dateien, z.B. *# include< QTCore/QWidget>*. Das C++ in Qt ist dem ISO C++ ähnlich, jedoch existieren Unterschiede. Qt nützt den Datentyp *QString* und nicht den C++ Datentyp *std::string*. Dieser wird von vielen Qt-spezifischen Funktionen gefordert.
Hierzu muss ein Datentyp *QString* definiert werden oder ein Datentyp *std::string* in einen Datentypen *QString* konvertiert werden. Anschließend kann ein Datentyp *QString* wieder in einen Datentyp *std::string* konvertiert werden. Mit Qt können für die meisten Betriebssysteme Programme erstellt werden, jedoch müssen sie für jedes Betriebssystem extra kompiliert werden. [12]

2.2.3. JavaScript Object Notation(JSON)

Das JavaScript Object Notation (JSON) ist ein kompaktes Datenformat in Textform. JSON-Datein dienen zum Datenaustausch zwischen Anwendungen und zum Speichern von strukturierten Daten. Damit werden allgemeine Datenstrukturen in Schlüssel-Wert-Paaren beschrieben. Beim Schlüssel-Wert-Paar *„NPV“:510* ist *„NPV“* der Schlüssel und *510* ist der Wert. Der Schlüssel gibt den Namen des Datenpunktes an. Für Datenpunkte gibt es folgende Datentypen:

Nullwert	ist definiert durch das Schlüsselwort Null.
Boolesche Werte	werden durch die Schlüsselwörter *true* und *false* beschrieben.
Zahlen	sind durch eine Folge von Ziffern 0 . . . 9 dargestellt. Mit einem Minus werden negative Zahlen definiert. Ein Punkt leitet eine Dezimalzahl ein. Durch den Buchstaben e bzw. E und nachfolgende Vorzeichen und Ziffern kann eine exponentielle Zahl erstellt werden.
Zeichenketten	beginnen und enden mit einem Anführungszeichen. Sie können Unicode-Zeichen und Escape-Sequenzen enthalten.
Arrays	beginnen und enden mit eckigen Klammern. Die Eigenschaften werden durch ein Komma getrennt.
Objekte	beginnen und enden mit geschweiften Klammern. Die Eigenschaften sind durch ein Komma getrennt. Objekte ohne Eigenschaften sind zulässig.
Eigenschaften	bestehen aus einem Schlüssel und einem Wert, die durch einen Doppelpunkt getrennt sind. Jeder Schlüssel darf in einem Objekt nur einmal vorkommen. Der Schlüssel ist eine Zeichenkette. Der Wert kann ein Nullwert, ein Boolescher Wert, eine Zahl, eine Zeichenkette, ein Array oder ein Objekt sein.

Aus diesen Elementen kann man Datenstrukturen erstellen. Dabei können die einzelnen Daten ineinander verschachtelt sein, z.B. wenn ein Array aus einem weiteren Array besteht. JSON benützt standardmäßig UTF-8 als Zeichenkodierung, aber auch UTF-16 und UTF-32 sind möglich.[13]

2.3. Mathematischer Hintergrund

2.3.1. Matrizen

Als Matrix bezeichnet man eine rechteckige Anordnung von Elementen. Dabei werden die horizontalen Reihen „Zeilen“ und die vertikalen Reihen „Spalten“ genannt. Ein Eintrag in

einer Matrix heißt Matrixelement. Das können Zahlen, Variablen oder Funktionen sein. Die Anordnung der Elemente lautet $a_{ij}(i = 1 \dots m, j = 1 \dots n)$ in einer $m \times n$ Matrix, wobei m die Zeilen angibt und n die Spalten. Bei einer Matrix mit m Zeilen und n Spalten gibt m die Zeilendimension und n die Spaltendimension der Matrix an. Eine Matrix mit gleicher Anzahl an Zeilen und Spalten $(n \times n)$ ist eine quadratische Matrix. Zur Kennzeichnung einer Matrix schreibt man diese mit einem unterstrichenen Großbuchstaben z.B. $\underline{A}$, $\underline{B}$. [14]

$$\underline{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{2.1}$$

Spaltenmatrix

Eine Spaltenmatrix, oder auch Spaltenvektor genannt, ist eine Matrix mit nur einer Spalte $n \times 1$. Diese stellt man mit einem unterstrichenen Kleinbuchstaben dar z.B. $\underline{x}$, $\underline{y}$. [14]

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{2.2}$$

Einheitsmatrix I

Die Elemente $a_{11}, a_{22}, \dots, a_{nn}$ in einer $n \times n$ Matrix werden als Hauptdiagonale von $\underline{A}$ bezeichnet. Bei einer Einheitsmatrix $\underline{I}$ bestehen die Elemente der Hauptdiagonalen aus dem Zahlenwert Eins. Die restlichen Elemente haben den Zahlenwert Null. [14]

$$\underline{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

Einheitsvektor e_i

Bei einem Einheitsvektor sind alle Elemente gleich Null, bis auf das i-te Element, welches gleich Eins ist: $e_j = 1$ für $j = i$, $e_j = 0$ für $j \neq i$. [14]

$$\underline{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{2.4}$$

Multiplikation einer Matrix mit einem Spaltenvektor

Die Multiplikation einer $m \times n$ Matrix $\underline{A}$ mit einem $n \times 1$ Spaltenvektor $\underline{x}$ ergibt einen Vektor $\underline{y}$ mit der Dimension $m \times 1$. [14]

$$\underline{y} = \underline{A}\underline{x} \tag{2.5}$$

Das i-te Element von y ist gegeben durch eine Skalarmultiplikation des i-ten Zeilenvektors $\underline{A}$ mit dem Spaltenvektor $\underline{x}$:

$$y_i = \begin{bmatrix} a_{i1} & a_{i2} & \dots & a_{in} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \tag{2.6}$$

Determinanten

Die Determinante ist eine mathematische Kennzahl, die einiges über eine $n \times n$ Matrix aussagt. Die Berechnung der Determinante erfolgt nach einer vorgegebenen Rechenvorschrift. Die Determinante der Matrix $\underline{A}$ wird durch das Symbol $det\underline{A}$ gekennzeichnet.[14] Für die Berechnung der Determinante einer beliebigen $n \times n$ Matrix wird der Entwicklungssatz von Laplace verwendet,

$$det\underline{A} = \sum_{j=1}^{n} (-1)^{i+j} \cdot a_{ij} \cdot detA_{ij} \qquad \text{Entwicklung nach der i-ten Zeile} \tag{2.7}$$

$$det\underline{A} = \sum_{i=1}^{n} (-1)^{i+j} \cdot a_{ij} \cdot detA_{ij} \qquad \text{Entwicklung nach der j-ten Spalte} \tag{2.8}$$

wobei A_{ij} die $(n-1) \times (n-1)$ Untermatrix von $\underline{A}$ ist, die durch Streichen der i-ten Zeile und j-ten Spalte entsteht.

In dieser Arbeit wird die Regel von Sarrus verwendet,

$$\begin{aligned} det\underline{A} = \quad & a_{11} \cdot a_{22} \cdot a_{33} + a_{12} \cdot a_{23} \cdot a_{31} + a_{13} \cdot a_{21} \cdot a_{32} - \\ & a_{13} \cdot a_{22} \cdot a_{31} - a_{11} \cdot a_{23} \cdot a_{32} - a_{12} \cdot a_{21} \cdot a_{33} \end{aligned} \tag{2.9}$$

da die Koordinatentransformation nur eine 3×3 Matrix benötigt.

Inverse einer Matrix

Die Inverse einer $n \times n$ Matrix $\underline{A}$ wird mit $\underline{A}^{-1}$ bezeichnet und folgende Beziehung muss gelten,

$$\underline{A}\underline{A}^{-1} = \underline{A}^{-1}\underline{A} = \underline{I} \tag{2.10}$$

wobei $\underline{I}$ die Einheitsmatrix ist. Eine Matrix $\underline{A}$ besitzt nur eine Inverse Matrix $\underline{A}^{-1}$, wenn die Matrix $\underline{A}$ nicht singulär ist. Die Matrix $\underline{A}$ ist singulär, wenn die Zeilen oder Spalten voneinander linear abhängig sind. Die Singularität kann mit der Determinante überprüft werden. Eine Matrix mit Determinante ungleich Null besitzt keine Singularität und hat eine Inverse.[14] Die Inverse kann über die Adjunkte berechnet werden [15]. Dabei lässt sich die Adjunkte der Matrix $\underline{A}$ wie folgt darstellen [16]:

$$adj(A) = \begin{bmatrix} det\begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} & -det\begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix} & det\begin{pmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \\ -det\begin{pmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{pmatrix} & det\begin{pmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{pmatrix} & -det\begin{pmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{pmatrix} \\ det\begin{pmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{pmatrix} & -det\begin{pmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{pmatrix} & det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \end{bmatrix}^T \tag{2.11}$$

Die Inverse mit der Adjunkte und Determinante wird angeschrieben als:

$$A^{-1} = \frac{1}{detA} \cdot adjA \tag{2.12}$$

2.3.2. Vektoren

Ein Vektor besteht aus einer skalaren Größe und einer Richtungsinformation.
Ein Skalar ist eine Größe, die durch eine Zahl ausgedrückt werden kann z.B. Zeit, Masse, Länge.
Die Richtungsinformation kann durch eine Richtung oder mit ihrem Anfangspunkt und Endpunkt vollständig definiert sein. Der Vektor wird mit einem Buchstaben und einem Pfeil darüber gekennzeichnet, z.B. $\vec{a}, \vec{v}, \vec{x}, \vec{y}, \vec{r}$. [14]

Gebundener Vektor

Ein gebundener Vektor ist im Raum fixiert und darf daher nicht verschoben werden. Zwei gebundene Vektoren sind nur dann gleich, wenn ihre Länge, der Anfangspunkt und der Endpunkt gleich sind. Ein Beispiel für einen gebundenen Vektor ist der Ortsvektor. Der Ortsvektor von Anfangspunkt Q zum Endpunkt P wird folgendermaßen angegeben: r_{PQ}. Bei einem Anfangspunkt im Koordinatenursprung und dem Endpunkt P gibt man nur den Endpunkt r_P an.[14]

Rechtsdrehendes kartesisches Koordinatensystem

Vektoren sind erst dann praktisch nutzbar, wenn sie mathematisch eindeutig dargestellt werden können. Dafür wird ein kartesisches Koordinatensystem eingeführt. Dieses besteht aus drei Orthogonalen, d.h. zueinander senkrecht stehenden Achsen x, y, z. Dadurch kann ein Punkt P mit seinen Koordinaten x, y, z entlang der Achsen eindeutig festgelegt werden. Zusätzlich zur Orthogonalität des Koordinatensystems wird gefordert, dass es sich um ein rechtsdrehendes Koordinatensystem handelt. Damit ist eine konsistente Vektorrechnung möglich. [14]

Länge eines Vektors

Die Länge a eines Vektors $\vec{a}$ wird als Betrag oder Norm $|a|$ bezeichnet und lässt sich mit Hilfe des *Satz von Pythagoras* folgendermaßen berechnen: [14]

$$a = |a| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2.13}$$

Vektordarstellung mit Basisvektoren

Im rechtsdrehenden kartesischen Koordinatensystem sind die drei orthogonalen Basisvektoren $\underline{e}_x, \underline{e}_y, \underline{e}_z$ definiert.
Der Basisvektor e_x liegt auf der x-Achse und zeigt in die positive x-Richtung. Die Basisvektoren e_y und e_z liegen auf der y- und z- Achse und zeigen in die positive Richtung. Als Matrix angeschrieben lauten die Basisvektoren:

$$\underline{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \underline{e}_y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \underline{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.14}$$

Jeder Basisvektor ist ein Einheitsvektor mit der Länge Eins:

$$|\underline{e}_x| = |\underline{e}_y| = |\underline{e}_z| = 1 \tag{2.15}$$

Ein Vektor $\vec{a}$ lässt sich also als Linearkombination aus den Basisvektoren im Koordinatensystem KS wie folgt darstellen:

$$\vec{a}_{KS} = a_x \underline{e}_{x|KS} + a_y \underline{e}_{y|KS} + a_z \underline{e}_{z|KS} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{KS} \tag{2.16}$$

Einheitsvektor Berechnung

Einen gegebenen, vom Nullvektor verschiedenen Vektor $\vec{a}$ kann man normieren, indem man ihn durch seine Länge (Norm) dividiert:

$$\vec{e}_a = \frac{\vec{a}}{|\vec{a}|} \tag{2.17}$$

Als Matrix angeschrieben:

$$|\underline{a}| = \sqrt{x^2 + y^2 + z^2} \tag{2.18}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{x}{|\underline{a}|} \\ \frac{y}{|\underline{a}|} \\ \frac{z}{|\underline{a}|} \end{pmatrix} \tag{2.19}$$

2.3.3. Koordinatentransformation

Koordinatensysteme

Die relative Lage des Punktes P wird durch ein Koordinatensystem und einen Bezugspunkt definiert.

Koordinatentransformation

In Abb. 2.1 sind zwei kartesische Koordinatensysteme dargestellt. Das xy-System mit dem Ursprung O und das $\bar{x}\bar{y}$-System mit Ursprung $\bar{O}$. Dabei ist das $\bar{x}\bar{y}$-System gegenüber dem xy-System um den Betrag a bzw. b in der x- bzw. y-Richtung verschoben (translatorisch). Außerdem ist dass $\bar{x}\bar{y}$-System gegenüber dem xy-System um den Winkel φ gedreht (Rotation).

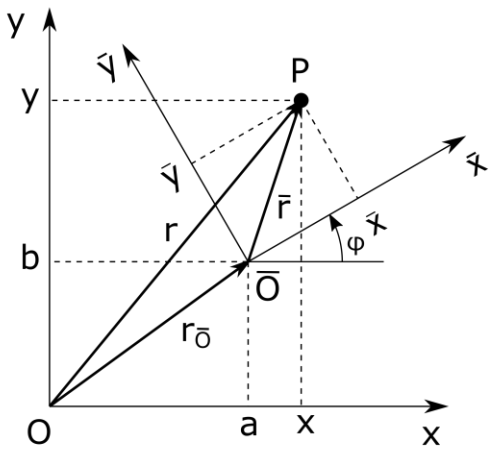


Abbildung 2.1.: Koordinatentransformation

Translation des Koordinatensystems

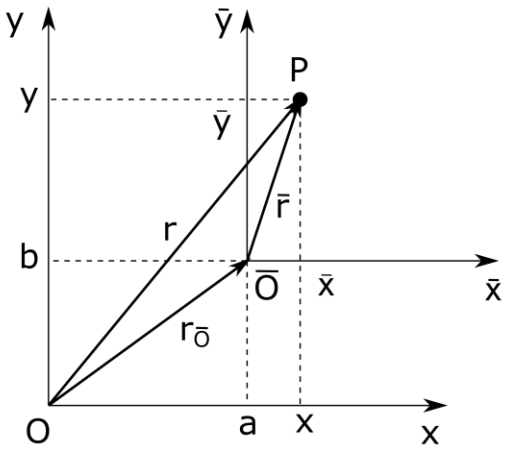


Abbildung 2.2.: Translation des Koordinatensystems

Die Translation ist eine Parallelverschiebung des Koordinatensystems. Wie Abb. 2.2 zeigt, bleibt die $\bar{x}$-Achse immer parallel zur x-Achse. Selbes gilt für die $\bar{y}$- und $\bar{z}$-Achse zur y- und z-Achse. Dabei wird der Ursprung $\bar{O}$ um den Ortsvektor $\vec{r}_{\bar{O}}$ gegenüber dem Ursprung O verschoben. Die Beziehung der Translation lautet:

$$\underline{x} = \underline{r}_{\bar{O}} + \underline{\bar{x}} \qquad bzw. \qquad \underline{\bar{x}} = \underline{x} - \underline{r}_{\bar{O}} \tag{2.20}$$

Als Matrix angeschrieben:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} \qquad bzw. \qquad \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{2.21}$$

Rotation des Koordinatensystems

Bei der Rotation (Drehung) wird der Ursprung O beibehalten. Das xy-System dreht sich um die z-Achse mit dem Drehwinkel φ, wie Abb. 2.3 zeigt. Dabei ist ein positiver Drehwinkel φ, eine Drehung gegen den Uhrzeigersinn und ein negativer Drehwinkel φ, eine Drehung im

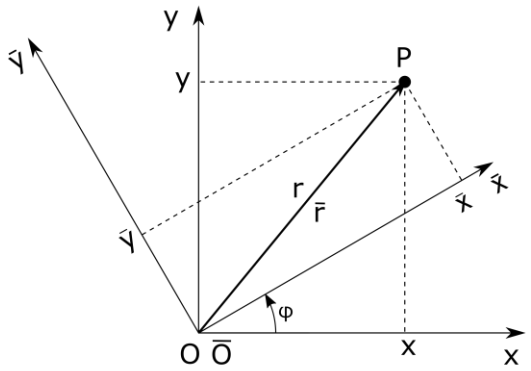


Abbildung 2.3.: Rotation des Koordinatensystems

Uhrzeigersinn möglich. Aus Bild 2.3 lassen sich folgende Zusammenhänge ablesen:

$$\bar{x} = x \cdot cos\varphi + y \cdot sin\varphi \qquad und \qquad \bar{y} = -x \cdot sin\varphi + y \cdot cos\varphi \tag{2.22}$$

Die Gl. (2.22) als Matrix angeschrieben:

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \begin{bmatrix} cos\varphi & sin\varphi & 0 \\ -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \underline{\bar{r}} = \underline{\underline{R}}\underline{r} \tag{2.23}$$

Dabei ist die erste Spalte $b_{\bar{x}} = [cos\varphi, \ -sin\varphi, 0]$ der erste Basisvektor des gedrehten Koordinatensystems, die zweite Spalte $b_{\bar{y}} = [sin\varphi, cos\varphi, 0]$ der zweite Basisvektor und die dritte Spalte $b_{\bar{z}} = [0, 0, 1]$ der dritte Basisvektor.

Translation und Rotation des Koordinatensystems

Bei dieser Koordinatentransformation wird das $\bar{x}\bar{y}-$System gegenüber dem $xy-$System sowohl gedreht als auch verschoben. Die Reihenfolge ist dabei nicht wichtig. Zuerst verschieben und dann drehen liefert das gleiche Ergebnis wie zuerst drehen und dann verschieben.
Die Beziehung der Transformation zwischen $xy-$System und $\bar{x}\bar{y}-$System lautet:

$$\bar{x} = (x-a) \cdot cos\varphi + (y-b) \cdot sin\varphi \quad und \quad \bar{y} = -(x-a) \cdot sin\varphi + (y-b) \cdot cos\varphi \tag{2.24}$$

und als Matrix angeschrieben:

$$\underline{\bar{r}} = \underline{R}(\underline{r} - \underline{r_{\bar{O}}}) \tag{2.25}$$

Umgeformt auf $\underline{r}$ lautet die Formel:

$$\underline{r} = \underline{R}^{-1}\underline{\bar{r}} + \underline{r_{\bar{O}}} \qquad \text{mit} \qquad \underline{R}^{-1} = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

Nimmt man den Vektor $\underline{r_{\bar{O}}}$ in die Matrix auf, erhöht sich die Dimension um Eins. Die Spalte $n+1$ enthält die Werte des Vektors $\underline{r_{\bar{O}}}$. Die Zeile $n+1$ besteht aus Nullen, bis auf $a_{n+1,n+1}$, dessen Wert ist Eins. Der alte Spaltenvektor lautet $\underline{\bar{r}} = (\bar{r}_x, \ldots, \bar{r}_z, 1)$ und der neue Vektor ist $\underline{r} = (r_x, \ldots, r_z, 1)$. Die Gleichung lautet nun:

$$\begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} cos\varphi & -sin\varphi & 0 & a \\ sin\varphi & cos\varphi & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{r}_x \\ \bar{r}_y \\ \bar{r}_z \\ 1 \end{bmatrix} \tag{2.27}$$

3. Hauptteil

Als Werkzeugmaschine wird eine 5-Achsen Syncromill der Firma Fill benützt. Diese kann eine oder zwei Spindeln besitzen und um die A- und B-Achse rotieren. Als CNC nützt der Maschinenhersteller eine Sinumerik 840 sl, es wird ein offenes System angeboten, damit der Maschinenhersteller eigene Anwendungen hinzufügen kann und wird in C++ programmiert [3]. Die Anwendungen laufen auf dem Human Machine Interface (HMI), da die Numerical Control Kernel (NCK) nur für die bearbeitenden Prozesse genutzt werden soll. Die Anwendung ist für die Software Stand 4.93 ausgelegt.

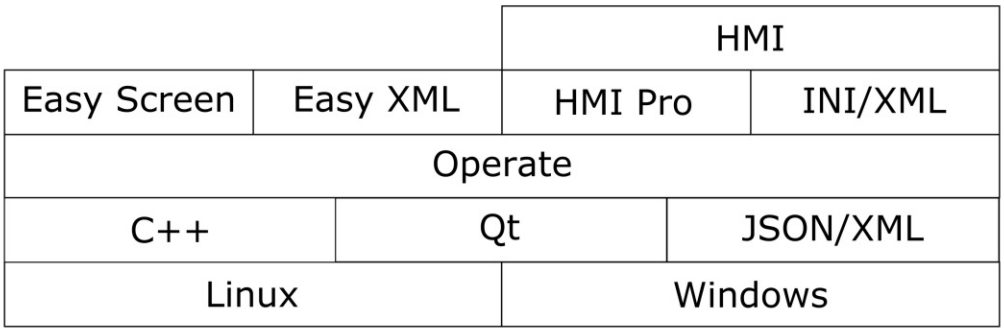


Abbildung 3.1.: Aufbau Sinumerik Steuerung

Abb. 3.1 zeigt den Aufbau der Steuerung. Die Sinumerik ist auf Linux sowie auch auf Windows Betriebssystemen nutzbar. Hier wird mittels C++ programmiert. Für das graphische User Interface (GUI) steht Qt zur Verfügung und für den Datenaustausch ein JSON-Datenformat. So ist auch der Operator programmiert. Dieser stellt HMI Pro für eigene Erweiterungen zur Verfügung. Dabei nutzt HMI Pro INI- und XML-Datenformate zum Speichern. Die HMI Pro läuft auf der HMI. Auf der NCK fügt man Erweiterungen mit Easy Screen hinzu. Dabei werden Daten der Form Easy XML zum Speichern genutzt. Da HMI pro und Easy Screen nur einen kleinen Teil des Umfanges von C++ bieten, wird die Anwendung in C++

programmiert. Damit ist auch sichergestellt, dass keine Änderungen von außenstehenden Personen möglich sind.

Bauteilprogrammierung

Zum Fertigen eines Bauteils benötigt die Steuerung das dazugehörende NC-Programm. Hier stehen alle Informationen über die Bearbeitung, wie z.B. Bewegung, Positionierung und Drehzahl der Spindel. Die Information setzt sich aus Befehlen oder auch Wörtern zusammen. Dabei ergeben mehrere Wörter immer einen Satz. Das Programm wird von der Steuerung dann satzweise abgearbeitet. [7]
Das NC-Programm ist relativ einfach bei Bauteilen, deren Flächen parallel zu den Ebenen des Maschinenkoordinatensystems (MKS) sind. Kompliziert ist die Beschreibung von Flächen, die beliebig im Raum liegen. Benützt das NC-Programm das MKS, dann ist das eine Bearbeitung in allen drei Raumrichtungen. Dadurch wird das NC-Programm unübersichtlich und ein Lesen der Sätze fast unmöglich. Deswegen fügt man ein neues Bezugssystem ein. Dieses nennt man auch Nullpunktverschiebung (NPV) und es wird mit den Wörtern G510-G599 (bei Siemens) bezeichnet. Die NPV liegt auf der zu bearbeitenden Fläche und ist gegenüber dem MKS beschrieben. Dadurch vereinfacht sich das NC-Programm auf eine zweidimensionale Bearbeitung.
Für die Bearbeitung einer beliebigen Fläche im Raum muss die Maschine das Bauteil und das Werkzeug zueinander ausrichten. Das geschieht in einer 5-Achsen-Werkzeugmaschine entweder durch das Rotieren des Bauteils oder der Spindel um zwei Achsen.
Programmieren im Konstruktionskoordinatensystem bedeutet, dass die Nullpunktverschiebungen sich auf das Konstruktionskoordinatensystem beziehen. Die Anwendung rechnet die Koordinaten auf das MKS um, sodass die Steuerung damit arbeiten kann.

3.1. Transformation Rotation Application

Die Werkzeugmaschine verfährt mit der Spindel absolut im Raum. Jede NPV rechnet die Steuerung auf das MKS zurück. Jeder Punkt, der im NC-Programm vorkommt, ist ein fixer Punkt im MKS.
Die Lage des Bauteils ändert sich durch eine Rotation. Jedoch bleibt das MKS unverändert. Dadurch stimmen die Bearbeitungen nicht mehr mit der Lage des Bauteils überein. Um die Rotation zu berücksichtigen, muss dies der Steuerung mitgeteilt werden. Dafür stehen bei der Sinumerik der Cyclus 800 und der TRAORI zur Verfügung. Die Steuerung berechnet den

Punkt nach der Rotation und fährt diesen an. Das funktioniert ausgezeichnet bei Maschinen mit einer Spindel und einem MKS. Betrachten wir eine Werkzeugmaschine mit einer Steuerung und zwei Spindeln, die sich gegenüber befinden, dann zeigen die z-Achsen zueinander, die y-Achsen sind gleich und die x-Achsen zeigen auseinander. Dementsprechend ergeben sich auch die Vorzeichen für die Rotationsachsen. Der Cyclus 800 und der TRAORI berechnen für eine Spindel die Rotation um die B-Achse richtig, für die andere Spindel jedoch falsch, da die Vorzeichen der B-Achse nicht gleich sind. Eine positive Drehung um die eine Achse bedeutet eine negative Drehung für die andere Achse. Dies hat zur Folge, dass die Spindeln nicht gleich verfahren, sondern in x-Richtung auseinanderfahren.

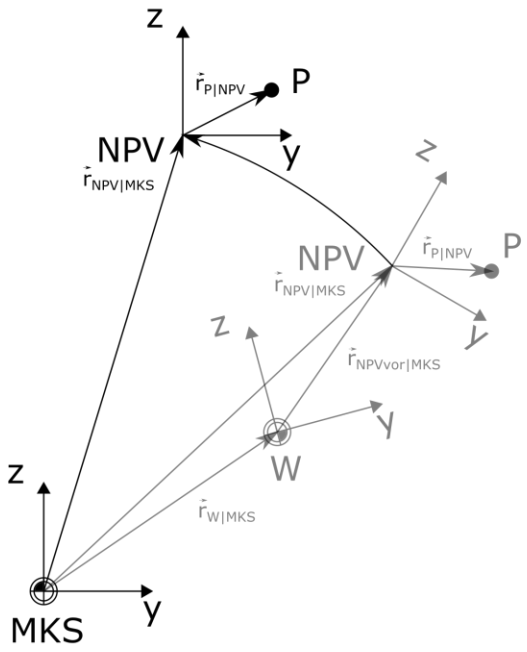


Abbildung 3.2.: Rotation der NPV im TRAF

Damit die Steuerung dies berücksichtigt, wurde die Transformation-Rotation-Application-Fill (TRAF) geschrieben. Sie berechnet die Rotationen bei Maschinen mit mehreren Spindeln. Der korrigierte Ortsvektor vom Maschinennullpunkt zur NPV lautet:

$$\vec{r}_{NPV\_M} = \vec{r}_{MEAF\_M} + \vec{r}_{NPVnach\_M} + \vec{r}_{G54\_M} + \vec{r}_{NPVvor\_G54} \tag{3.1}$$

Dabei ist $\vec{r}_{MEAF\_M}$ ein Korrekturwert für die Wärmeausdehnung, $\vec{r}_{G54\_M}$ der Ortsvektor zum Werkstücknullpunkt und $\vec{r}_{NPVvor\_G54}$ der Ortsvektor von Werkstücknullpunkt zur NPV. Mit Berücksichtigung der Rotationsmatrix $\underline{R}$ folgt:

$$\vec{r}_{NPV\_M} = \vec{r}_{MEAF\_M} + \vec{r}_{NPVnach\_M} + \underline{\underline{R}}(\vec{r}_{G54\_M} + \vec{r}_{NPVvor\_G54}) \tag{3.2}$$

Die Gl. (3.2) als Matrix angeschrieben lautet:

$$\underline{r}_{NPV\_M|MKS} = \underline{r}_{MEAF\_M|MKS} + \underline{r}_{NPVnach\_M|MKS} + \underline{\underline{R}} \cdot (\underline{r}_{G54\_M|MKS} + \underline{r}_{NPVvor\_G54|MKS}) \tag{3.3}$$

$$\begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{MKS} = \begin{bmatrix} x_{MEAF\_M} \\ y_{MEAF\_M} \\ z_{MEAF\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} x_{NPVnach\_M} \\ y_{NPVnach\_M} \\ z_{NPVnach\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \left(\begin{bmatrix} x_{G54\_M} \\ y_{G54\_M} \\ z_{G54\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} x_{NPVvor\_G54} \\ y_{NPVvor\_G54} \\ z_{NPVvor\_G54} \end{bmatrix}_{MKS} \right) \tag{3.4}$$

Der Ortsvektor vom Maschinennullpunkt zur NPV von Gl. (3.4) lautet:

$$\begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{MKS} = \begin{bmatrix} x_{G54\_M} \\ y_{G54\_M} \\ z_{G54\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} x_{NPVvor\_G54} \\ y_{NPVvor\_G54} \\ z_{NPVvor\_G54} \end{bmatrix}_{MKS} \tag{3.5}$$

Die Rotationsmatrix um die y-Achse:

$$\underline{\underline{R}}_{Y(\beta)} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{3.6}$$

Die Rotationsmatrix um die x-Achse:

$$\underline{\underline{R}}_{X(\alpha)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{3.7}$$

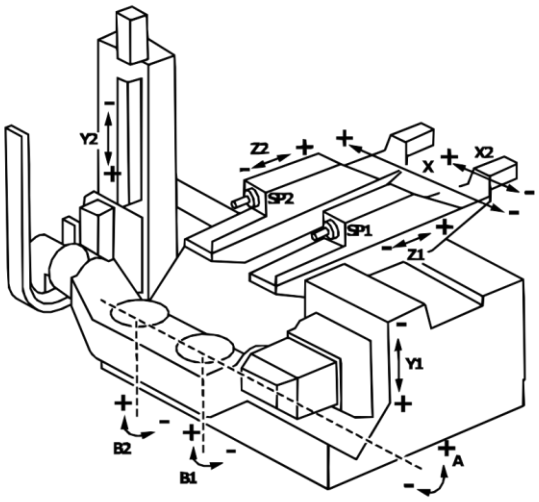


Abbildung 3.3.: Achsenbezeichnung Syncromill C21 63-600

Mit der Rotationsmatrix aus Gl. (3.6) und (3.7) ergibt sich eine Rotation um die y-Achse und anschließend um die x-Achse wie folgt:

$$\underline{\underline{R}}_{(\alpha,\beta)} = \underline{\underline{R}}_{Y(\beta)}\underline{\underline{R}}_{X(\alpha)} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}$$
$$\underline{\underline{R}}_{(\alpha,\beta)} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta) & sin(\alpha) & cos(\alpha)cos(\beta) \end{bmatrix} \tag{3.8}$$

Aus Gl. (3.5) und (3.8) ergibt sich der um BA gedrehte Ortsvektor von M zur NPV:

$$\begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{(\alpha,\beta)|MKS} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta) & sin(\alpha) & cos(\alpha)cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{MKS} \tag{3.9}$$

$$\begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{(\alpha,\beta)|MKS} = \begin{bmatrix} x_{NPV\_W} \cdot cos(\beta) + z_{NPV\_M} \cdot sin(\beta) \\ x_{NPV\_M} \cdot sin(\alpha)sin(\beta) + y_{NPV\_M} \cdot cos(\alpha) - z_{NPV\_M} \cdot sin(\alpha)cos(\beta) \\ -x_{NPV\_M} \cdot cos(\alpha)sin(\beta) + y_{NPV\_M} \cdot sin(\alpha) + z_{NPV\_M} \cdot cos(\alpha)cos(\beta) \end{bmatrix}_{MKS} \tag{3.10}$$

Gl. (3.10) in Gl. (3.1) ergibt damit:

$$\begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{(\alpha,\beta)|MKS} = \begin{bmatrix} x_{MEAF\_M} \\ y_{MEAF\_M} \\ z_{MEAF\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} x_{NPVnach\_M} \\ y_{NPVnach\_M} \\ z_{NPVnach\_M} \end{bmatrix}_{MKS} + \begin{bmatrix} x_{NPV\_M} \\ y_{NPV\_M} \\ z_{NPV\_M} \end{bmatrix}_{(\alpha,\beta)|MKS} \tag{3.11}$$

Die Steuerung berücksichtigt die Transformation mit dem TRAF-Befehl. Dafür ruft das NC-Programm jede NPV in Form eines TRAF-Befehls auf. Die Liste aller NPV steht in einem NC-Unterprogramm, welches am Beginn des NC-Teilprogrammes aufgerufen wird. Durch das Aufrufen des Unterprogrammes bei jedem Bauteil können Wärmeausdehnungen bei jedem Bauteil berücksichtigt werden. Außerdem verhindert man manuelle Änderungen an den NPV.

Der TRAF-Befehl ist folgendermaßen aufgebaut:

TRAF(npv, w, xVor, yVor, zVor, alpha, beta, xNach, yNach, zNach, meaf, spi, bea)	
Parameter	**Bedeutung**
npv	Nullpunktverschiebung G510 = 10, G511 = 11 usw.
w	Bauteilkoordinatensystem G54 = 1, G55 = 2 usw.
xVor	x Koordinate vor der Drehung
yVor	y Koordinate vor der Drehung
zVor	z Koordinate vor der Drehung
alpha	Drehung um A
beta	Drehung um B
xNach	Δx Koordinate nach der Drehung
yNach	Δy Koordinate nach der Drehung
zNach	Δz Koordinate nach der Drehung
meaf	Berücksichtigung der Wärmeausdehnung MEAF ja = 1, nein = 0
spi	Anzahl der Spindeln 1 oder 2
bea	Bearbeitung Position stehend = 1, hängend = 2

Tabelle 3.1.: TRAF Befehl

3.2. Aufbau der Anwendung

3.2.1. Umgang mit den Dateien

Die NPVs sind in ein Unterprogramm geschrieben, welches das Hauptprogramm aufruft. Für die neue Anwendung wird dieses Unterprogramm mit den NPVs genutzt. Dafür liest die Anwendung das Unterprogramm und schreibt die Informationen in eine neue Datei. Dort liegen die NPVs in einem JSON-Format vor. Dadurch lassen sich die Daten einfach in einem Programm bearbeiten.

Für die Umrechnung benötigt die Anwendung noch eine Transformationsmatrix. Diese Matrix errechnet sich aus der Lage des KKS. Die Informationen darüber werden auch in einem JSON-Datenformat gespeichert. Damit kann die Anwendung die NPVs in das KKS umrechnen. Am Bildschirm der Steuerung können nun die Maschinenbediener Korrekturen an den NPVs vornehmen. Die Korrektur rechnet die Anwendung zurück in das MKS und gibt ein

neues Unterprogramm aus. Mit diesem Unterprogramm kann die Steuerung alle NPVs direkt setzen. Abb. 3.4 zeigt die verschiedenen Daten. Dabei erfolgen Änderungen nur über die JSON-Dateien.

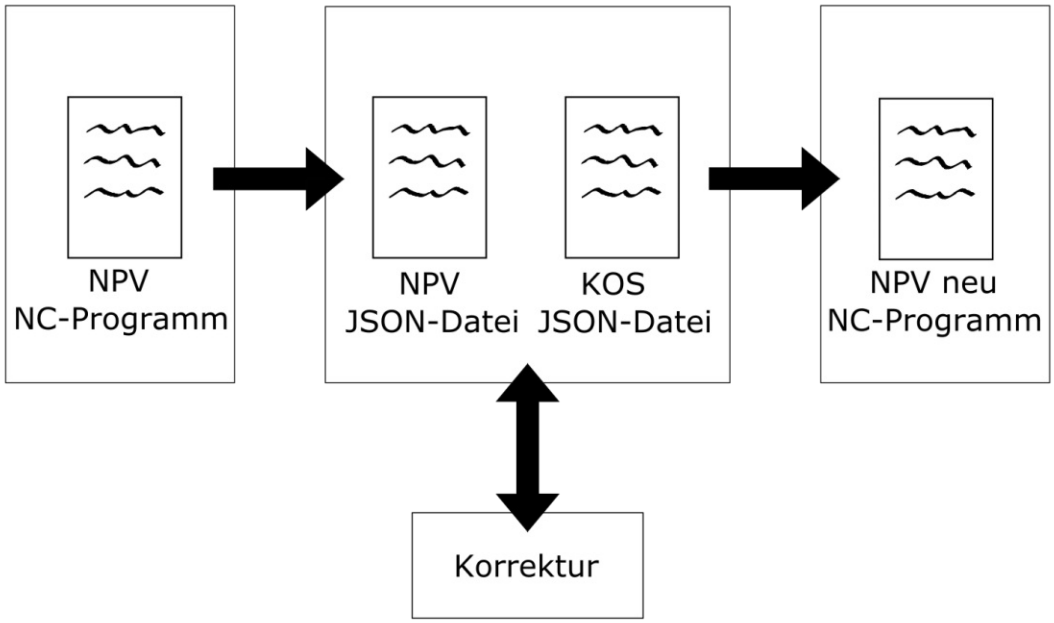


Abbildung 3.4.: Verwendete Dateitypen und deren Umwandlung

Abb. 3.5 erklärt die Transformationen der Werte. Die Anwendung liest die NPVs aus der JSON-Datei ab und rechnet sie in das KKS. Diese Information wird nicht gespeichert, sondern nur dem_der Bediener_in angezeigt. Findet eine Korrektur statt, rechnet die Anwendung nur den Korrekturwert in das MKS zurück, da die ursprünglichen Werte des Bereichs vorhanden sind und nur eine Korrektur hinzukommt. Danach addiert die Anwendung die Korrektur zur NPV.

3.3. Aufbau des Graphic-User-Interface (GUI)

Die Bedieneroberfläche setzt sich aus den Bedienbereichen zusammen. Ein Bedienbereich besteht im einfachsten Fall aus einem Dialog. Somit besteht eine Anwendung aus einem oder mehreren Dialogen. Ein Dialog setzt sich aus einem oder mehreren Screens zusammen. Der Screen definiert die Softkeys und zeigt die Form an. Jeder Screen kann eine oder mehrere Formen enthalten. Die Form bildet die Anzeige auf der Steuerung. Dabei können sich auch

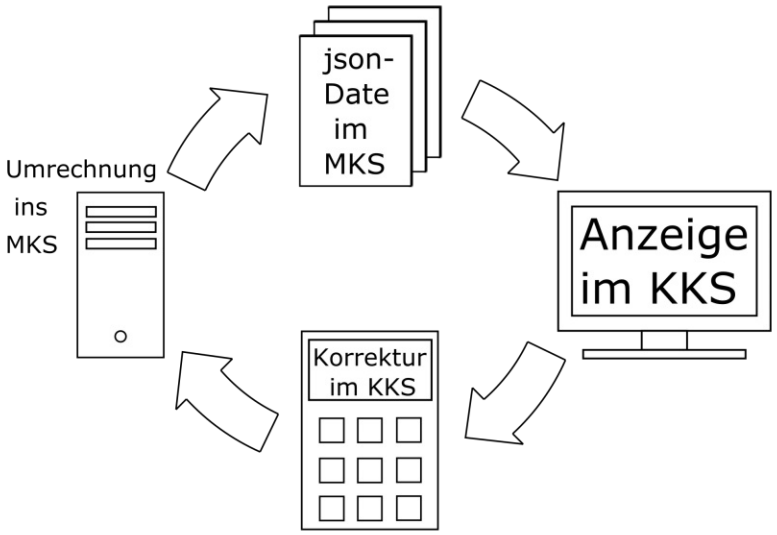


Abbildung 3.5.: Prozess einer Korrektur in der Anwendung

unterschiedliche Screens die selbe Form teilen. Eine Form besteht aus Widgets. Ein Widget kann zum Beispiel ein Eingabefeld oder Ausgabefeld sein. Die Screens und die Formen werden in C++ Klassen programmiert. In einem XML-Format setzt das Programm die Screens und die Formen zusammen und fügt Softkeys hinzu. [17]

3.3.1. Hauptbildschirm

Auf dem Hauptbildschirm bekommt der_die Bediener_in die NPV in unterschiedlichen Bezugssystemen angezeigt. Zum Hauptbildschirm der Anwendung gelangt der_die Bediener_in durch einen Menüpunkt im Hauptmenü der Sinumerik-Steuerung. Der Hauptbildschirm ist Ausgangspunkt für weitere Optionen in der Anwendung. Durch das Klicken eines Softkeys in der unteren Leiste (8) wird dem_der Anwender_in die NPV in einem anderen Bezugssystem angezeigt. Korrekturwerte können im Feld (6) eingegeben werden. Durch Klicken des „Korrigieren“-Softkeys wird die Änderung gespeichert. Im ausklappbaren Menü (2) wählt der_die Bediener_in eine andere NPV. Mit dem Softkey „Aktualisieren“ bestätigt der_die Anwender_in die ausgewählte NPV. Der Softkey „Neu“ ist zum Hinzufügen eines neuen NPV-Unterprogramms. Dazu wechselt die Anwendung zum Screen 3.3.2. Mit dem Softkey „Öffnen“ kann der_die Bediener_in eine vorhandene JSON-Datei öffnen, wobei die Anwen-

dung ebenfalls zum Screen 3.3.2 wechselt. Unter dem Softkey „Einstellung“ befinden sich die Einstellungen für die TRAF-Berechnung. Hier wechselt die Anwendung zum Screen 3.3.3. Mit dem Softkey „Koordinaten“ kann der_die Bediener_in ein neues Bezugssystem definieren oder ein vorhandenes bearbeiten. Dazu wechselt die Anwendung zum Screen 3.3.4.

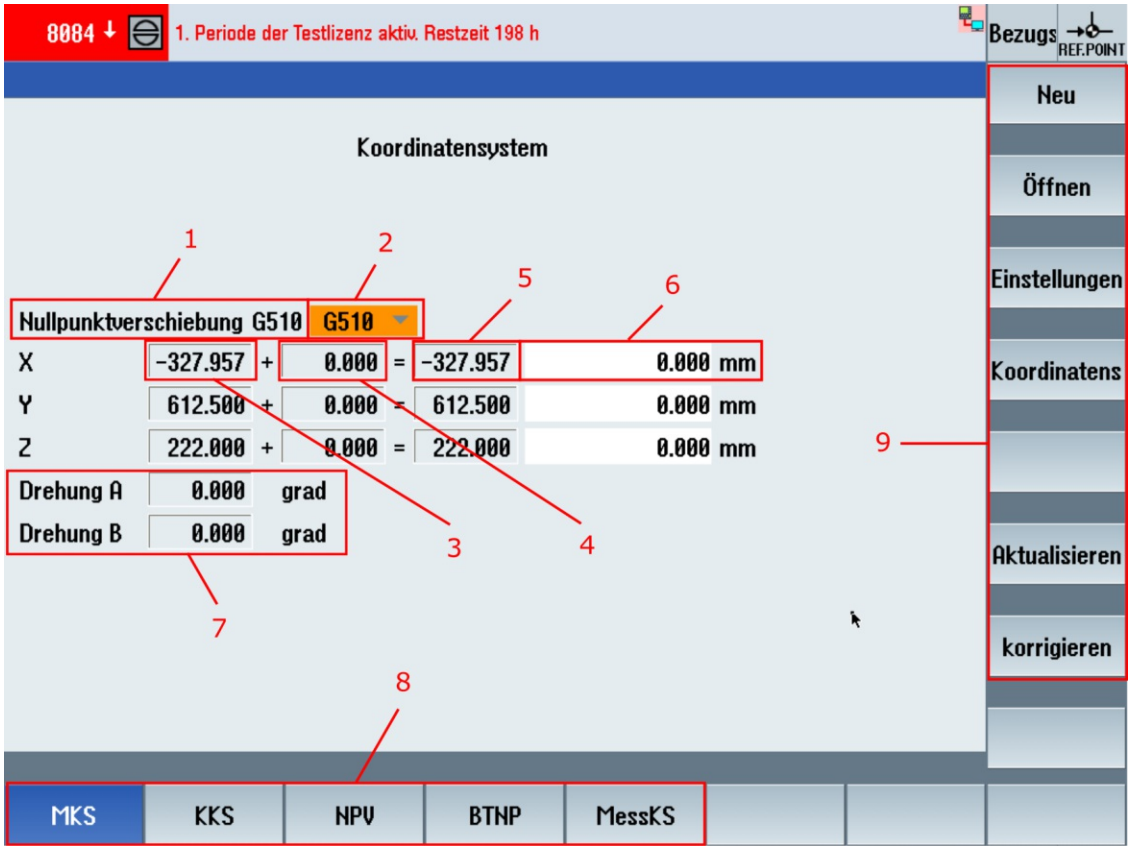


Abbildung 3.6.: Hauptbildschirm der Anwendung

Wie in der Abbildung 3.6 ersichtlich:

1. Anzeige der aktiven NPV
2. Auswahl der NPV
3. Verschiebung in x-Richtung
4. Aktueller Korrekturwert
5. Summe aus ursprünglichem x-Wert und Korrekturwert
6. Eingabe eines Korrekturwertes

7. Verdrehung der NPV in A und B
8. Auszuwählende Bezugssysteme
9. Weitere Softkeys

Aufbau des Hauptbildschirms

Die Abbildungen 3.7 und 3.8 zeigen das Flussdiagramm der Anwendung für den Hauptbildschirm. Dabei sind alle selbst geschriebenen Funktionen in einem Rechteck dargestellt. Mit einer Route werden die wichtigsten `if`-Verzweigungen angedeutet. Die Ovale deuten auf den Wechsel zu anderen Screens hin. Die Softkeys sind als Quadrate dargestellt.

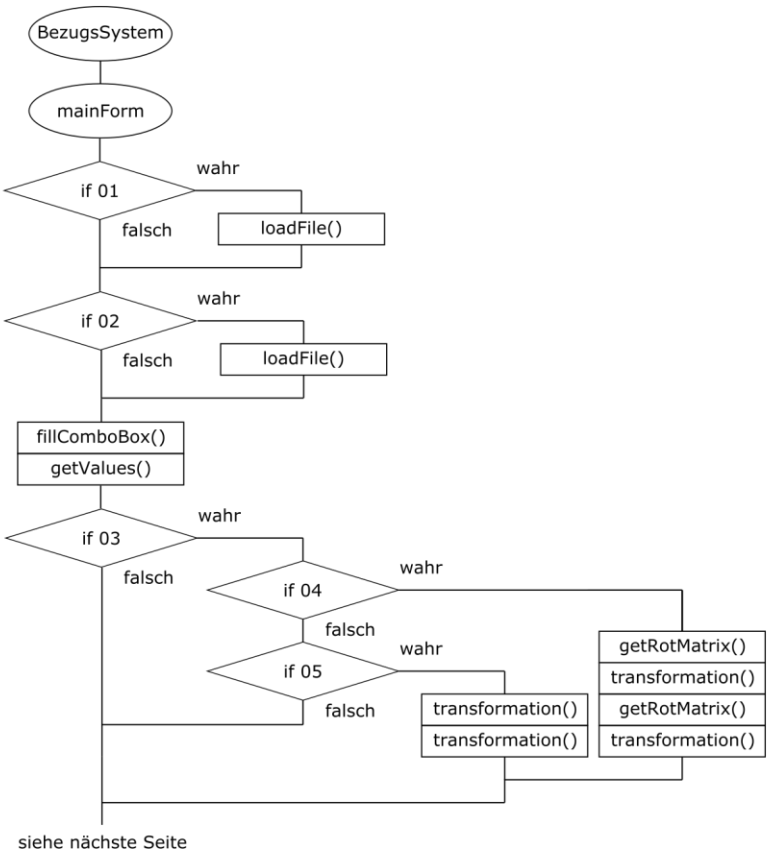


Abbildung 3.7.: Flussdiagramm der Anwendung für den Hauptbildschirm Teil 1

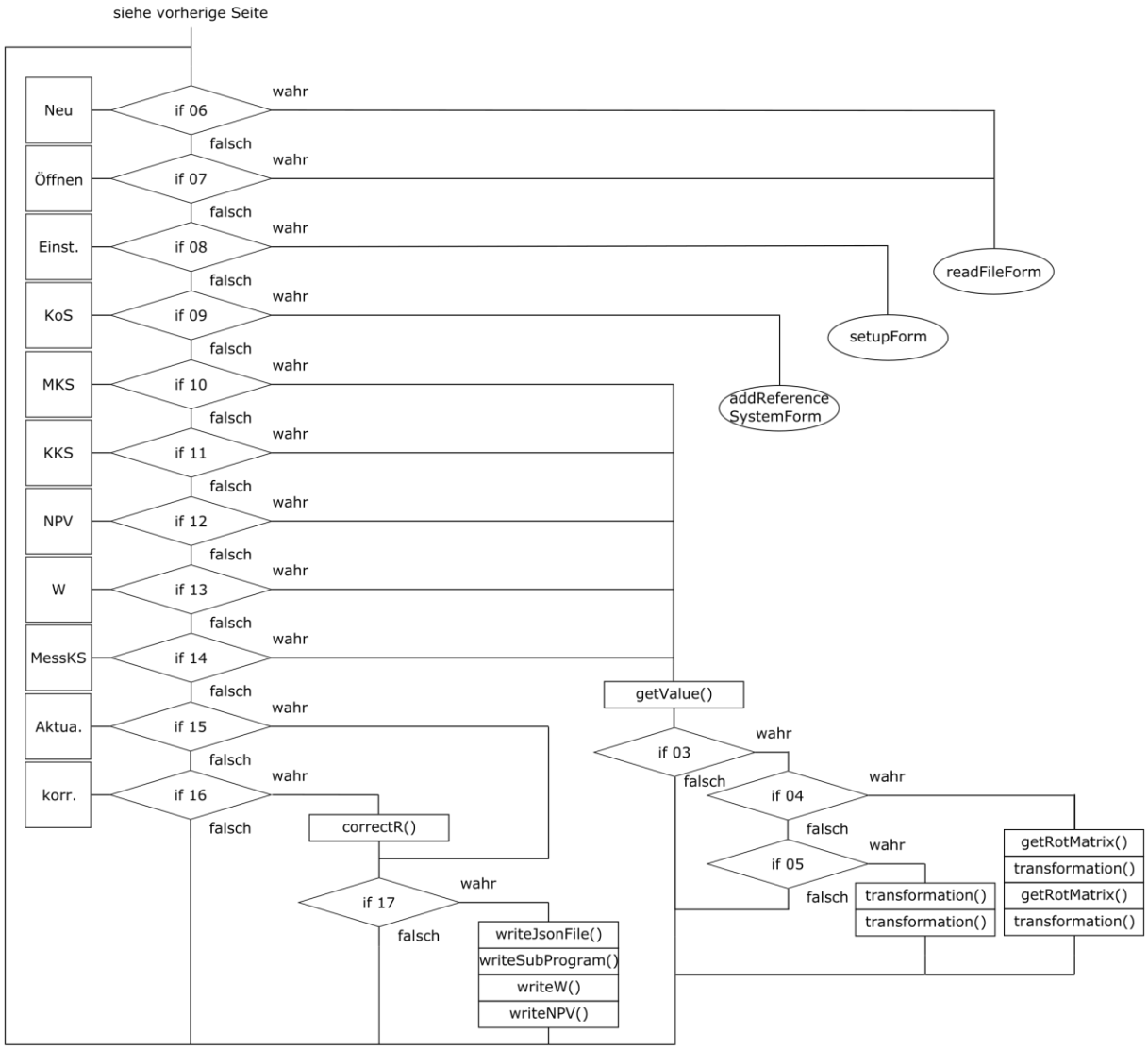


Abbildung 3.8.: Flussdiagramm der Anwendung für den Hauptbildschirm Teil 2

Abkürzung	Bedeutung
mainForm	Form des Hauptbildschirms siehe Kap. 3.3.1
Neu	Softkey-Bezeichnung zum Einlesen einer neuen Datei
Öffnen	Softkey-Bezeichnung zum Öffnen einer Datei
Einst.	Softkey-Bezeichnung für die Einstellungen
KoS	Softkey-Bezeichnung zum Definieren eines Koordinatensystems
MKS	Softkey-Bezeichnung für die Maschinenkoordinatensystem-Ansicht
KKS	Softkey-Bezeichnung für die Konstruktionskoordinatensystem-Ansicht
NPV	Softkey-Bezeichnung für die Nullpunktverschiebungs-Ansicht
W	Softkey-Bezeichnung für die Werkstückkoordinatensystem-Ansicht
MessKS	Softkey-Bezeichnung für die Messmaschinenkoordinatensystem-Ansicht
Aktua.	Softkey-Bezeichnung zum Aktualisieren
Korr.	Softkey-Bezeichnung zum Korrigieren
If 01	Wenn-Abfrage, ob eine NPV in Form einer JSON-Datei vorhanden ist
If 02	Wenn-Abfrage, ob Koordinatensysteme in Form von JSON-Dateien vorhanden sind
If 03	Wenn-Abfrage, ob NPVs vorhanden sind
If 04	Wenn-Abfrage, ob in das NPV-Koordinatensystem umgerechnet wird
If 05	Wenn-Abfrage, ob das ausgewählte Koordinatensystem vorhanden ist
If 06	Wenn-Abfrage, ob „Neu“-Softkey betätigt wurde
If 07	Wenn-Abfrage, ob „Öffnen“-Softkey betätigt wurde
If 08	Wenn-Abfrage, ob „Einst.“-Softkey betätigt wurde
If 09	Wenn-Abfrage, ob „KoS“-Softkey betätigt wurde
If 10	Wenn-Abfrage, ob „MKS“-Softkey betätigt wurde
If 11	Wenn-Abfrage, ob „KKS“-Softkey betätigt wurde

If 12	Wenn-Abfrage, ob „NPV“-Softkey betätigt wurde
If 13	Wenn-Abfrage, ob „W“-Softkey betätigt wurde
If 14	Wenn-Abfrage, ob „MessKS“-Softkey betätigt wurde
If 15	Wenn-Abfrage, ob „Aktua:“-Softkey betätigt wurde
If 16	Wenn-Abfrage, ob „Korr.“-Softkey betätigt wurde
If 17	Wenn-Abfrage, ob Korrekturwerte eingetragen sind
loadFile()	Funktion zum Öffnen und Einlesen einer Datei siehe Kap. 3.4.3
fillComboBox()	Funktion zum Befüllen des ausklappbaren Menüs in Abb. 3.6 (2) siehe Kap. 3.4.4
getValues()	Funktion zum Auslesen der Daten aus einer JSON-Datei siehe Kap. 3.4.5
getRotMatrix()	Funktion zum Bestimmen der Rotationsmatrix siehe Kap. 3.4.10
transformation()	Funktion zur Transformation eines Vektors siehe Kap. 3.4.11
correctR()	Funktion zum Zurückrechnen der Korrekturwerte siehe Kap. 3.4.13
writeJsonFile()	Funktion zum Erzeugen einer JSON-Datei siehe Kap. 3.4.14
writeSubProgram()	Funktion zum Erstellen eines NC-Programms mit NPVs siehe Kap. 3.4.17
writeW()	Funktion, welche den Werkstücknullpunkt in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.15
writeNPV()	Funktion, welche die NPVs in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.16
addReferenceSystemForm	Form des Bildschirms zum Definieren eines Koordinatensystems siehe Kap. 3.3.4
setupForm	Form des Bildschirms für Einstellungen siehe Kap. 3.3.3
readFileForm	Form des Bildschirms zum Öffnen von Dateien bzw. zum Einfügen einer neuen Datei siehe Kap. 3.3.2

Tabelle 3.2.: Beschreibung der Abkürzungen im Flussdiagramm der Anwendung für den Hauptbildschirm

3.3.2. „Öffnen und Einfügen“-Bildschirm

Im Öffnen- und Einfügenbildschirm werden sowohl die Ordner als auch die Bauteil-, Teil- und Unterprogramme angezeigt (1). Die Ordner liegen auf dem NCK. Sie enthalten jedoch nicht alle Dateien. Ein Filter lässt nur bestimmte Dateien anzeigen, z.B. Unterprogramm-Dateien beim Softkey „Einlesen“ oder JSON-Dateien beim Softkey „Öffnen“. Auf der rechten Seite befinden sich die Softkeys (2) zum Einlesen, Öffnen und Abbrechen. Durch den Softkey „Abbruch“ wechselt die Anwendung zum Hauptbildschirm zurück. Mittels des Softkeys „Einlesen“ bzw. des Softkeys „Öffnen“ kehrt die Anwendung auch zum Hauptbildschirm zurück, zudem wird die ausgewählte Datei geladen und angezeigt.

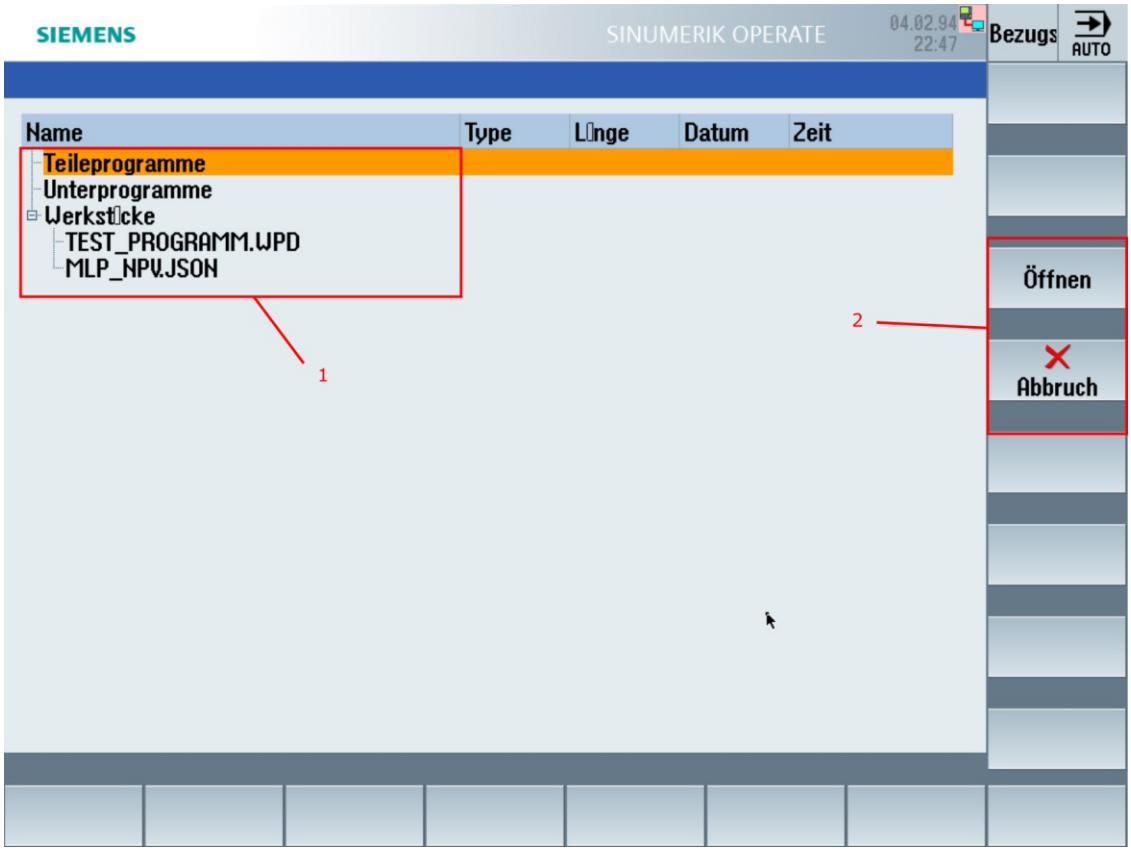


Abbildung 3.9.: Bildschirm zum Öffnen der Datei bzw. einer neuen Datei

Wie in der Abbildung 3.9 ersichtlich:

1. Ordner und Dateien
2. Softkeys

Aufbau des Einfügen- und Öffnenbildschirms

Abbildung 3.10 zeigt das Flussdiagramm. Dabei sind in den Rechtecken alle selbst geschriebenen Funktionen. Die Rauten stellen alle wichtigen `if`-Verzweigungen dar. Die Ovale deuten auf den Wechsel zu anderen Screens hin. Softkeys sind als Quadrate dargestellt.

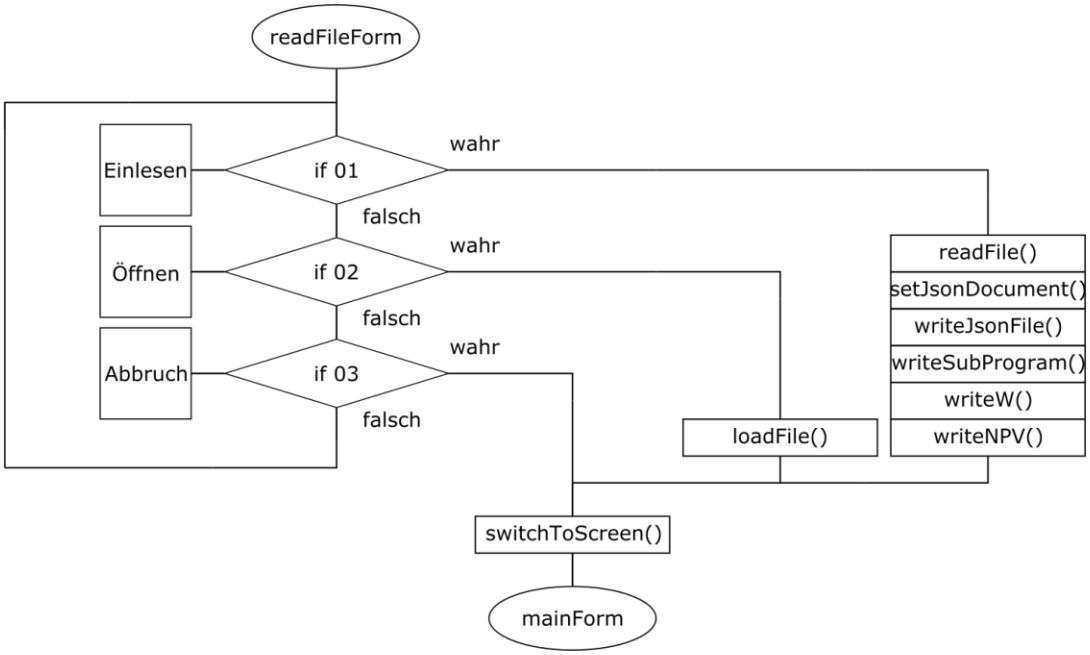


Abbildung 3.10.: Flussdiagramm der Anwendung für den „Einfügen“- und „Öffnen“-Bildschirm

Abkürzung	Bedeutung
readFileForm	Form des Bildschirms zum Öffnen von Dateien bzw. zum Einfügen einer neuen Datei siehe Kap. 3.3.2
Einlesen	Softkey-Bezeichnung zum Einlesen
Öffnen	Softkey-Bezeichnung zum Öffnen
Abbruch	Softkey-Bezeichnung zum Abbrechen
If 01	Wenn-Abfrage, ob „Einlesen“-Softkey betätigt wurde
If 02	Wenn-Abfrage, ob „Öffnen“-Softkey betätigt wurde
If 03	Wenn-Abfrage, ob „Abbruch“-Softkey betätigt wurde
loadFile()	Funktion zum Öffnen und Einlesen einer Datei siehe Kap. 3.4.3

readFile()	Funktion, welche die NPVs aus einem NC-Programm herausfiltert siehe Kap. 3.4.1
setJsonDocument()	Funktion, welche die durch readFile() eingelesenen Daten in eine JSON- Datei umwandelt siehe Kap. 3.4.2
writeJsonFile()	Funktion zum Erzeugen einer JSON-Datei siehe Kap. 3.4.14
writeSubProgram()	Funktion zum Erstellen eines NC-Programmes mit NPVs siehe Kap. 3.4.17
writeW()	Funktion, welche den Werkstücknullpunkt in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.15
writeNPV()	Funktion, welche die NPVs in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.16
switchToScreen()	Standardfunktion von Siemens zum Wechseln des Bildschirms
mainForm	Form des Hauptbildschirms siehe Kap. 3.3.1

Tabelle 3.3.: Beschreibung der Abkürzungen im Flussdiagramm der Anwendung für den „Einfügen“- und „Öffnen“-Bildschirm

3.3.3. Eigenschaftenbildschirm

Mit dem Eigenschaftenbildschirm werden die TRAF-Optionen gesetzt. Bei (1) stellt der_die Bediener_in den zu beachtenden Werkstücknullpunkt ein. Durch (2) berücksichtigt die TRAF-Rechnung die thermische Ausdehnung, vorausgesetzt die Maschine kann die thermische Ausdehnung ermitteln. Ob die Maschine eine oder zwei Spindeln hat, gibt der_die Anwender_in bei (3) an. Mit (4) legt der_die Bediener_in fest, ob das Werkstück steht oder hängt. Auf der rechten Seite befinden sich wieder Softkeys. Mit dem Softkey „Ok“ werden die Einstellungen in der JSON-Datei gespeichert und die Anwendung wechselt zum Hauptbildschirm zurück. Durch den Softkey „Abbruch“ speichert die Anwendung nichts und kehrt zum Hauptbildschirm zurück.

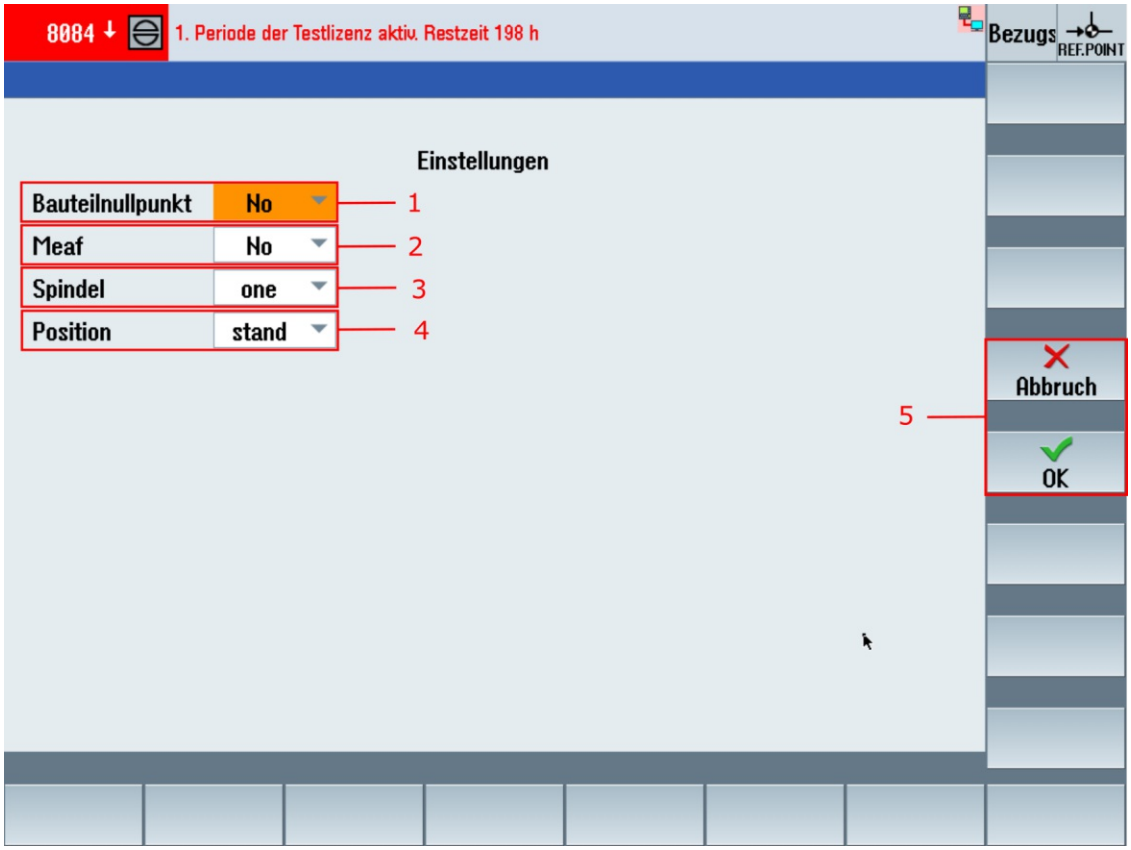


Abbildung 3.11.: Bildschirm für die Einstellungen

Wie in der Abbildung 3.11 ersichtlich:

1. Bauteilnullpunkt (Werkstücknullpunkt)
2. Thermische Ausdehnung
3. Anzahl der Spindeln
4. Lage des Werkstücks

Aufbau des Eigenschaftenbildschirm

Abbildung 3.12 zeigt das Flussdiagramm. Dabei sind in den Rechtecken alle selbst geschriebenen Funktionen. Die Rauten stellen alle wichtigen `if`-Verzweigungen dar. Die Ovale deuten auf den Wechsel zu anderen Screens hin. Softkeys sind wieder als Quadrate dargestellt.

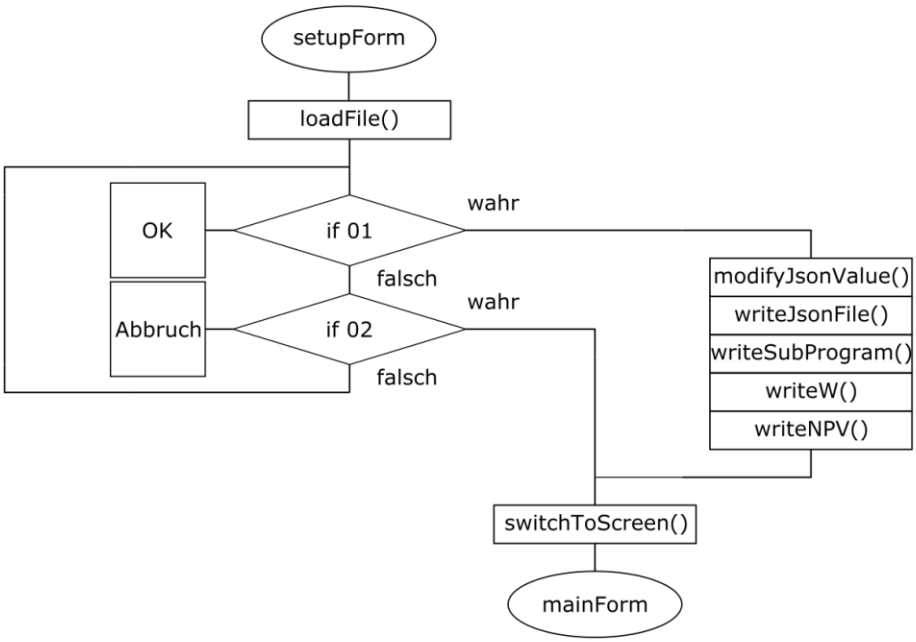


Abbildung 3.12.: Flussdiagramm der Anwendung für die Einstellungen

Abkürzung	**Bedeutung**
setupForm	Form des Bildschirms für Einstellungen siehe Kap. 3.3.3
OK	Softkey-Bezeichnung zum Bestätigen
Abbruch	Softkey-Bezeichnung zum Abbrechen
If 01	Wenn-Abfrage, ob „OK“-Softkey betätigt wurde
If 02	Wenn-Abfrage, ob „Abbruch“-Softkey betätigt wurde
loadFile()	Funktion zum Öffnen und Einlesen einer Datei siehe Kap. 3.4.3
modifyJsonValue()	Funktion für Änderungen in einer JSON-Datei siehe Anhang C
writeJsonFile()	Funktion zum Erzeugen einer JSON-Datei siehe Kap. 3.4.14
writeSubProgram()	Funktion zum Erstellen eines NC-Programmes mit NPVs siehe Kap. 3.4.17
writeW()	Funktion, welche den Werkstücknullpunkt in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.15
writeNPV()	Funktion, welche die NPVs in einen für die Steuerung lesbaren Befehl schreibt siehe Kap. 3.4.16

switchToScreen()	Standardfunktion von Siemens zum Wechseln des Bildschirms
mainForm	Form des Hauptbildschirms siehe Kap. 3.3.1

Tabelle 3.4.: Beschreibung der Abkürzungen im Flussdiagramm der Anwendung für die Einstellungen

3.3.4. Bezugssystem Bearbeitungsbildschirm

Hier kann der_die Anwender_in ein neues Bezugssystem definieren. Außerdem kann ein vorhandenes Bezugssystem bearbeitet werden. Dafür muss der_die Bediener_in die drei Einheitsbasisvektoren (x, y, z) des MKS im neuen Bezugssystem angeben (2-3). Zusätzlich benötigt die Anwendung den Abstand des MKS vom neuen Bezugssystem als Offset (1). Mit dem Softkey „Ok“ (5) wird das Bezugssystem in der JSON-Datei gespeichert. Falls das Bezugssystem schon vorhanden ist, überschreibt das Programm dieses. Danach kommt der_die Anwender_in wieder zurück zum Hauptbildschirm. Durch den Softkey „Abbruch“ (5) werden die Änderungen nicht berücksichtigt und die Anwendung wechselt zum Hauptbildschirm zurück.

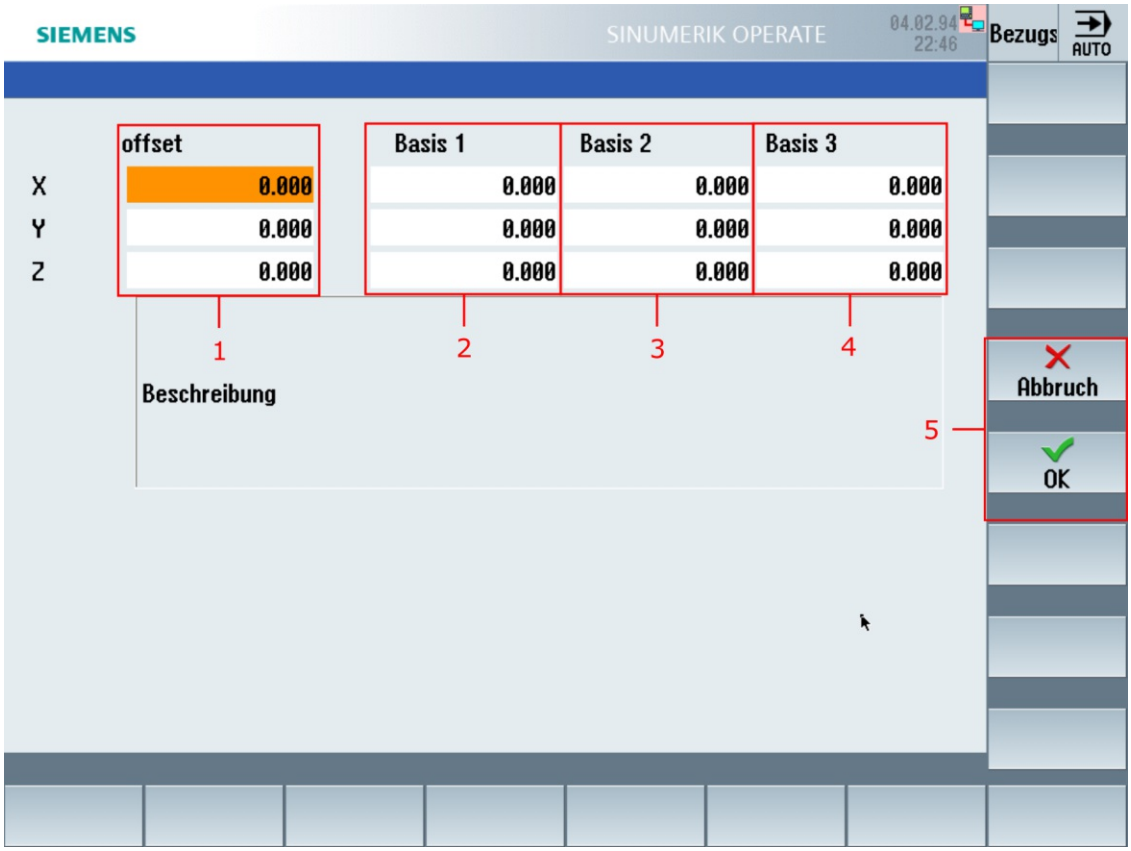


Abbildung 3.13.: Bildschirm zum Bestimmen des Koordinatensystems

Wie in der Abbildung 3.13 ersichtlich:

1. Verschiebung des MKS gegenüber dem Bezugssystem
2. Einheitsbasisvektor x
3. Einheitsbasisvektor y
4. Einheitsbasisvektor z
5. Softkeys

Aufbau des Bezugssystem Bearbeitungsbildschirm

Abbildung 3.14 zeigt das Flussdiagramm. Dabei sind in den Rechtecken alle selbst geschriebenen Funktionen. Die Rauten stellen alle wichtigen `if`-Verzweigungen dar. Die Ovale deuten auf den Wechsel zu anderen Screens hin. Softkeys sind als Quadrate dargestellt.

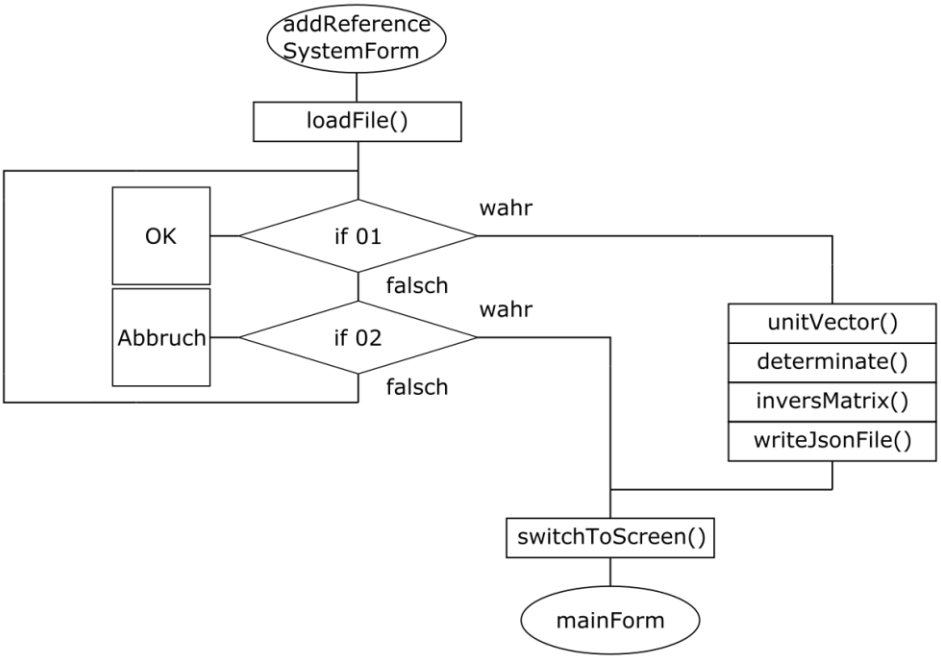


Abbildung 3.14.: Flussdiagramm der Anwendung zum Bearbeiten des Bezugssystems

Abkürzung	**Bedeutung**
addReferenceSystemForm	Form des Bildschirms zum Bestimmen eines Koordinatensystems siehe Kap. 3.3.4
OK	Softkey-Bezeichnung zum Bestätigen
Abbruch	Softkey-Bezeichnung zum Abbrechen
If 01	Wenn-Abfrage, ob „OK“-Softkey betätigt wurde
If 02	Wenn-Abfrage, ob „Abbruch“-Softkey betätigt wurde
loadFile()	Funktion zum Öffnen und Einlesen einer Datei siehe Kap. 3.4.3
unitVector()	Funktion zum Berechnen des Basiseinheitsvektors einer Matrix siehe Kap. 3.4.6
determinate()	Funktion zum Berechnen der Determinante einer Matrix siehe Kap. 3.4.7
inversMatrix()	Funktion zum Berechnen der Inverse einer Matrix siehe Kap. 3.4.8
writeJsonFile()	Funktion zum Erzeugen einer JSON-Datei siehe Kap. 3.4.14
switchToScreen()	Standardfunktion von Siemens zum Wechseln des Bildschirms

mainForm	Form des Hauptbildschirms siehe Kap. 3.3.1

Tabelle 3.5.: Beschreibung der Abkürzungen im Flussdiagramm der Anwendung zum Bearbeiten des Bezugssystems

3.3.5. Aufbau der Screens

Der Bildschirm besteht aus den Softkeys und Formen. Wie schon in 3.3 erwähnt, wird der Aufbau der Anwendung in einer XML-Datei festgelegt. Ein XML-Element besteht aus einem Start-Tag (z.B. `<MENU>`) und einem End-Tag (z.B. `</MENU>`). Einem XML-Element können auch mehrere Attribute mit einem Namen und einem Wert zugewiesen werden. Dort definiert das Programm alle Screens, Formen und Softkeys. Im Listing 3.1 ist ein Softkey beispielhaft erzeugt.

```
<MENU name="vrMain" softkeybar="vr">
  <SOFTKEY position="1" accesslevel="7">
    <PROPERTY name="textID" type="QString">Softkey_name</PROPERTY>
    <FUNCTION name="Softkey_function" args="" />
  </SOFTKEY>
</MENU>
```

Listing 3.1: Softkey erzeugen

Die Bedeutung der einzelnen Schlüsselwörter steht in der Tabelle 3.6.
Als Nächstes wird ein Bildschirm definiert. Dem Bildschirm müssen eine Form und ein Menü zugeordnet werden. In dem Listing 3.2 ist wieder ein Beispiel dafür angegeben.

MENU name	Definiert den Namen des Menüs
MENU softkeybar	Gibt an, wo sich das Menü befindet („v“ Vertikal, „h“ Horizontal, „r“ Rechts, „l“ Links
SOFTKEY position	Gibt den Platz des Softkeys an, dabei geht es von 1 bis 8
SOFTKEY accesslevel	Definiert, welche Berechtigungsstufe benötigt wird, um den Softkey zu sehen
PROPERTY name	Gibt nähere Information über den Typ z.B.: „textID“ angezeigter Text auf dem Softkey, „picture“ angezeigtes Bild auf dem Softkey
PROPERTY type	Typ und sichtbarer Name des Softkeys
FUNCTION name	Variable, welche dem Programm zurückgegeben wird
FUNCTION args	Zusätzliche Argumente, die dem Programm zurückgegeben werden

Tabelle 3.6.: Definition Menü.

```
<SCREEN name="ScreenMain">
  <FORM name="mainForm" formpanel="FullForm" implementation="TRAF.mainForm
   "/>
  <MENU ref ="vrMain"/>
</SCREEN>
```

Listing 3.2: Bildschirm erzeugen

In der Tabelle 3.7 stehen die Schlüsselwörter für den Bildschirm.

SCREEN name	Definition des Screen-Namens
FORM name	Definition des Form-Namens
FORM formpanel	Definition der Größe der Form z.B. „FullForm“
FORM implementation	Definiert, welche Datei hinter der Form steckt
MENU ref	Definiert, welches Menü zu dem Screen gehört

Tabelle 3.7.: Definition Bildschirm.

Hauptbildschirm

Da die Anwendung mehrere Bezugssysteme nützen kann, muss das aktive Bezugssystem angezeigt werden. Dafür hinterlegt die Anwendung den aktiven Softkey mit einem blauen Hintergrund. Für die Umsetzung fasst die Anwendung die Softkeys zu einer Gruppe zusammen. Eine Softkeygruppe fängt mit dem XML-Element `Softkeygroup` an und hat den Attributnamen `name`. Mit dem Attributnamen `selectedsoftkey` und dem Wert „1“ ist immer nur ein Softkey ausgewählt [17]. Am Anfang wird die horizontale untere Softkeybar mit

dem XML-Element `MENU` erstellt. Dieses hat den Attributnamen `name` und den Wert „huMain“. Danach folgt die vertikale rechte Softkeybar mit dem Attributnamen `name` und dem Wert „vrMain“. Das XML-Element `SCREEN` bekommt den Attributnamen `name` mit dem Wert „ScreenMain“. Im XML-Element `MENU` werden dem Menü die Softkeys mit dem Attributnamen `ref` und die Werte `huMain` und `vrMain` zugewiesen. Der Screen erhält das XML-Element `FORM` mit dem Attributnamen `name` und dem Wert `mainForm`, welcher mit dem Attributnamen `implementation` den Wert der Datei „TRAF.mainForm“ zugewiesen bekommt.

```
<MENU name="huMain" softkeybar="hu">
  <SOFTKEYGROUP name="GroupMain" selectedsoftkey="1">

    <SOFTKEY position="1" accesslevel="7">
      <PROPERTY name="textID" type="QString">MKS</PROPERTY>
      <FUNCTION name="MCS" args="" />
    </SOFTKEY>

    <SOFTKEY position="2" accesslevel="7">
      <PROPERTY name="textID" type="QString">KKS</PROPERTY>
      <FUNCTION name="CCS" args="" />
    </SOFTKEY>

    <SOFTKEY position="3" accesslevel="7">
      <PROPERTY name="textID" type="QString">NPV</PROPERTY>
      <FUNCTION name="NPV" args="" />
    </SOFTKEY>

    <SOFTKEY position="4" accesslevel="7">
      <PROPERTY name="textID" type="QString">BTNP</PROPERTY>
      <FUNCTION name="PCS" args="" />
    </SOFTKEY>

    <SOFTKEY position="5" accesslevel="7">
      <PROPERTY name="textID" type="QString">MessKS</PROPERTY>
      <FUNCTION name="MeasCS" args="" />
    </SOFTKEY>

  </SOFTKEYGROUP>
</MENU>

<MENU name="vrMain" softkeybar="vr">

```

```
34   <SOFTKEY position="1" accesslevel="7">
35     <PROPERTY name="textID" type="QString">Neu</PROPERTY>
36     <FUNCTION name="new" args="" />
37     </SOFTKEY>
38
39     <SOFTKEY position="2" accesslevel="7">
40     <PROPERTY name="textID" type="QString">Oeffnen</PROPERTY>
41     <FUNCTION name="open" args="" />
42     </SOFTKEY>
43
44     <SOFTKEY position="3" accesslevel="7">
45     <PROPERTY name="textID" type="QString">Einstellungen</PROPERTY>
46     <FUNCTION name="setup" args="" />
47     </SOFTKEY>
48
49     <SOFTKEY position="4" accesslevel="7">
50     <PROPERTY name="textID" type="QString">Koordinatensystem Bearbeiten</
       PROPERTY>
51     <FUNCTION name="coordination" args="" />
52     </SOFTKEY>
53
54     <SOFTKEY position="6" accesslevel="7">
55     <PROPERTY name="textID" type="QString">Aktualisieren</PROPERTY>
56     <FUNCTION name="refresh" args="" />
57     </SOFTKEY>
58
59     <SOFTKEY position="7" accesslevel="7">
60     <PROPERTY name="textID" type="QString">korrigieren</PROPERTY>
61     <FUNCTION name="correct" args="" />
62     </SOFTKEY>
63
64 </MENU>
65
66 <SCREEN name="ScreenMain">
67   <FORM name="mainForm" formpanel="FullForm" implementation="TRAF.mainForm
       "/>
68     <MENU ref ="huMain"/>
69     <MENU ref ="vrMain"/>
70 </SCREEN>
```

Listing 3.3: Bildschirm erzeugen

Einfügenbildschirm

Auch hier ist zuerst die vertikale Softkeybar auf der rechten Seite definiert. Die XML-Elemente „Softkey_OK“ und „Softkey_CANCEL“ sind von Siemens vordefinierte Softkeys. Die Softkeys haben nicht nur den Namen „Ok“ bzw. „Abbruch“, sondern auch ein kleines Häkchen bzw. Kreuz als Symbol. Außerdem ändert sich der Name mit der ausgewählten Sprache mit.
Der Softkey „cancel_setup“ erhält die Eigenschaft, den Bildschirm zu wechseln, da beim Klicken der Abbruchtaste die Anwendung nur zum Hauptbildschirm wechseln soll. Der Bildschirmwechsel ist mit dem XML-Element `Navigation` und dem Attributnamen `target` und dem Wert „screen“ definiert. Durch das XML-Element `SCREEN`, den Attributnamen `name` und den Screen-Namen „ScreenMain“ als Wert ist der Ziel-Screen definiert. Die Softkeybar wird dem XML-Element `SCREEN` mit dem Attributwert „ScreenNew“ im XML-Element `MENU` mit dem Attributnamen `ref` hinzugefügt. Das XML-Element `FORM` erhält den Attributnamen „newForm“, welcher die Datei „TRAF.readFileForm“ als Wert erhält.

```
<MENU name="vrNew" softkeybar="vr">

    <SOFTKEY_CANCEL position="4">
    <FUNCTION name="cancel_setup" args="" />
    <NAVIGATION target="screen">
      <SCREEN name="ScreenMain" />
    </NAVIGATION>
    </SOFTKEY_CANCEL>

    <SOFTKEY position="3">
    <PROPERTY name="text" type="QString">Einlesen</PROPERTY>
    <PROPERTY name="disabledFace" type="SoftKeyDisabledFace">Grayed</
   PROPERTY>
    <FUNCTION name="import" args="" />
    </SOFTKEY>

</MENU>

<SCREEN name="ScreenNew">
    <FORM name="newForm" implementation="TRAF.readFileForm"
          formpanel="FullForm" terminate="true"/>
    <MENU ref="vrNew"/>
</SCREEN>
```

Listing 3.4: Bildschirm erzeugen

Öffnenbildschirm

Dieser Screen ist gleich aufgebaut wie der Einfügenbildschirm. Die Softkeybar wird dem XML-Element `SCREEN` mit dem Attributwert „ScreenOpen“ und dem Attributnamen `ref` hinzugefügt. Die Form erhält den Attributwert „openForm“. Dabei verwendet die Anwendung die gleiche Datei wie beim Einfügenbildschirm („TRAF.readFileForm“).

```
<MENU name="vrOpen" softkeybar="vr">

    <SOFTKEY_CANCEL position="4">
    <FUNCTION name="cancel_setup" args="" />
    <NAVIGATION target="screen">
      <SCREEN name="ScreenMain" />
    </NAVIGATION>
    </SOFTKEY_CANCEL>

    <SOFTKEY position="3">
    <PROPERTY name="text" type="QString">Oeffnen</PROPERTY>
    <PROPERTY name="disabledFace" type="SoftKeyDisabledFace">Grayed</
   PROPERTY>
    <FUNCTION name="open" args="" />
    </SOFTKEY>

</MENU>

<SCREEN name="ScreenOpen">
    <FORM name="openForm" implementation="TRAF.readFileForm"
          formpanel="FullForm" terminate="true"/>
    <MENU ref="vrOpen"/>
</SCREEN>
```

Listing 3.5: Bildschirm erzeugen

Eigenschaftenbildschirm

Beim Eigenschaftenbildschirm wird zuerst die vertikale Softkeybar auf der rechten Seite definiert. Dann erzeugt das XML-Element `SCREEN` den Screen mit dem Attributwert „Screen-Setup“. Mit dem Attributnamen `ref` bindet die Anwendung die Softkeybar mit dem Attributwert „vrSetup“ ein. Die Form erhält den Attributwert „setupForm“, welcher die Datei „TRAF.setupForm“ als Wert enthält.

```
<MENU name="vrSetup" softkeybar="vr">

    <SOFTKEY_OK position="5" accesslevel="7">
    <FUNCTION name="ok_setup" args="" />
    <NAVIGATION target="screen" >
      <SCREEN name="ScreenMain" />
    </NAVIGATION>
    </SOFTKEY_OK>

    <SOFTKEY_CANCEL position="4">
    <FUNCTION name="cancel_setup" args="" />
    <NAVIGATION target="screen">
      <SCREEN name="ScreenMain" />
    </NAVIGATION>
    </SOFTKEY_CANCEL>

</MENU>

<SCREEN name="ScreenSetup">
    <FORM name="setupForm" implementation="TRAF.setupForm"
          formpanel="FullForm" terminate="true"/>
    <MENU ref="vrSetup" />
</SCREEN>
```

Listing 3.6: Bildschirm erzeugen

Bezugssystem Bearbeitungsbildschirm

Zuerst wird die vertikale Softkeybar mit dem Attributwert „vrAdd“ auf der rechten Seite definiert, die wieder die XML-Elemente „Softkey_OK“ und „Softkey_CANCEL“ besitzt. Mit dem Attributnamen `ref` fügt die Anwendung die Softkeybar dem Screen mit dem Attributwert „ScreenAdd“ hinzu. Der Screen erhält die Form mit dem Attributwert „addReferenceSystemForm“, welche aus der Datei „TRAF.addReferenceSystemForm“ als Wert besteht.

```
<MENU name="vrAdd" softkeybar="vr">

    <SOFTKEY_OK position="5" accesslevel="7">
    <FUNCTION name="ok_setup" args="" />
    <NAVIGATION target="screen">
      <SCREEN name="ScreenMain" />
    </NAVIGATION>
    </SOFTKEY_OK>
```

```
9
10      <SOFTKEY_CANCEL position="4">
11      <FUNCTION name="cancel_setup" args="" />
12      <NAVIGATION target="screen">
13        <SCREEN name="ScreenMain" />
14      </NAVIGATION>
15      </SOFTKEY_CANCEL>
16
17  </MENU>
18
19  <SCREEN name="ScreenAdd">
20      <FORM name="addReferenceSystemForm" implementation="TRAF.
        addReferenceSystemForm"
21            formpanel="FullForm" terminate="false"/>
22      <MENU ref="vrAdd" />
23  </SCREEN>
```

Listing 3.7: Bildschirm erzeugen

3.4. Funktion der Anwendung

3.4.1. Einlesen der Daten

Für die Transformation des MKS auf das KKS benötigt die Anwendung die NPV, da die Informationen im NPV-Unterprogramm stecken. Um die Zahlen nutzbar zu machen, müssen sie in ein JSON-Dateiformat gebracht werden. Dafür ist die Funktion `readFile` zuständig. Zuerst kopiert die Methode `copyFile` die Variable `fileName` mit dem Dateipfad auf die Speicherkarte unter den Pfad von der Variable `cardNpv`. Das Objekt `file` der Klasse `QFile` erzeugt aus der kopierten Datei ein Datei-Objekt. Mit einer `if`-Abfrage kontrolliert die Funktion, ob die Datei geöffnet wurde. Danach liest die Methode `readLine` jede Zeile der Datei und speichert diese in das Objekt `line` der Klasse `QString`. Die Methode `startsWith` überprüft, ob die Zeile mit „TRAF(“ anfängt. Ist das der Fall, dann schreibt die Funktion die Zeile in das Objekt `list` mit der Klasse `QStringList`. Danach übergibt die Funktion die Klasse `list` der Funktion `setJsonDocument`, die die Liste in ein JSON-Dateiformat bringt. Fängt die Zeile mit „$P_UIFR[1]=CTRANS(“ an, schreibt die Funktion die Werte in das Objekt `list` der Klasse `QStringList`. Dieses Objekt wird an die Funktion `setJsonDocument` weitergegeben. Am Ende schließt die Methode `close` die Datei wieder. Die Methode `remove` löscht die Datei von der Speicherkarte, deren Pfad in der Variable „cardPath“ steckt. Danach

speichert die Funktion `QJsonDocument` die NPV unter der Addresse des Objekts `doc`.

void fileManager::readFile	(QString ncPath, QString cardPath, QJsonDocument *doc)
Parameter	**Bedeutung**
QString ncPath	Gibt den Pfad auf der NCK an, wo die Datei zu finden ist.
QString cardPath	Gibt den Pfad auf der Speicherkarte an, wo die Datei zwischengespeichert wird.
QJsonDocument *doc	Übergibt das JSON Objekt der Funktion, in dieses werden die gelesenen Werte hingeschrieben.

Tabelle 3.8.: Funktion readFile().

```
1  void fileManager::readFile(QString ncPath, QString cardPath, QJsonDocument
       *doc)
2  {
3    fileServer.copyFile(ncPath, cardPath, true);
4    QFile file(cardPath);
5
6    if (file.open(QIODevice::ReadOnly))
7    {
8      QJsonObject objTraf;
9      QTextStream in(&file);
10     while (!in.atEnd())
11     {
12       QString line = in.readLine();
13
14       if (line.startsWith("TRAF("))
15       {
16         QStringList list;
17         line = line.remove(0, 5);
18         line = line.remove(-1, 1);
19         list = line.split(",");
20         setJsonDocument(list, &objTraf);
21       }
22       else if (line.startsWith("$P_UIFR[1]=CTRANS("))
23       {
24         QStringList list;
25         line = line.remove(0, line.indexOf(",") + 1);
26         line = line.remove(line.lastIndexOf("+"), line.length());
```

```
27          line = line.replace(line.indexOf("+"), line.indexOf("Y") - line.
    indexOf("+") + 2, " ");
28          line = line.replace(line.indexOf("+"), line.indexOf("Z") - line.
    indexOf("+") + 2, " ");
29          line = line.replace(line.indexOf("+"), line.indexOf("Z") - line.
    indexOf("+") + 3, " ");
30          list = line.split(QRegularExpression("\\s+"));
31          list.push_front("1");
32          setJsonDocument(list, &objTraf);
33        }
34      }
35      file.close();
36      fileServer.remove(cardPath);
37
38      *doc = QJsonDocument(objTraf);
39    }
40  }
```

Listing 3.8: Funktion readFile()

3.4.2. Erstellen eines JSON Files

Ein `QJsonDocument` besteht immer aus der Klasse `QJsonObject` oder einem `QJsonArray`, welche wieder aus einem `QJsonObject` oder einem `QJsonArray` bestehen können usw. (Siehe Anhang A).
Mit der Funktion `setJsonDocument` bringt die Anwendung die Werte in ein JSON-Dateiformat. Dafür übergibt die Funktion eine Liste mit den Werten und ein Objekt zum Befüllen. Durch eine `if`-Verzweigung entscheidet die Funktion, ob eine NPV oder ein Werkstücknullpunkt vorliegt. Die Werte werden von der Funktion in die Klasse `QJsonObject` oder `QJsonArray` umgewandelt. Diese werden dann mit dem jeweiligen Schlüsselwort einem `QJsonObject` übergeben. Die Methode `insert` übergibt dem Zeiger des Objekts `obj` die zurückzugebenden Elemente.

void fileManager::setJsonDocument (QStringList list, QJsonObject *obj)	
Parameter	**Bedeutung**
QStringList list	Liste mit Elementen, die in ein JSON-Format zu bringen sind.
QJsonObject *obj	Dieses Objekt wird zurückgegeben. Darin befinden sich die Elemente in einem JSON-Objekt.

Tabelle 3.9.: Funktion setJsonDocument().

```
void fileManager::setJsonDocument(QStringList list, QJsonObject *obj)
{
  QJsonArray arrVec;
  QJsonObject objNPV;
  QJsonObject objSetup;

  if (list.size() > 12)
  {
    //INT _NPV, INT _W, REAL _Xvor, REAL _Yvor, REAL _Zvor, REAL _A, REAL
    _B, REAL _Xnach, REAL _Ynach, REAL _Znach, BOOL _MEAF, INT _SP, INT
    _BEAz
    objNPV["npv"] = list[0].toInt();
    objSetup["W"] = list[1].toInt();

    arrVec = { list[2].toDouble(), (double) 0.0 };
    objNPV["x"] = arrVec;

    arrVec = { list[3].toDouble(), (double) 0.0 };
    objNPV["y"] = arrVec;

    arrVec = { list[4].toDouble(), (double) 0.0 };
    objNPV["z"] = arrVec;

    objNPV["alpha"] = list[5].toDouble();
    objNPV["beta"] = list[6].toDouble();

    if (list[10] == "1")
      objSetup["meaf"] = true;

    else if (list[10] == "0")
      objSetup["meaf"] = false;

```

```
31      objSetup["spindel"] = list[11].toInt();
32      objSetup["bea"] = list[12].toInt();
33
34      *obj->insert("setup", objSetup);
35      *obj->insert(list[0], objNPV);
36    }
37    else if (list.size() < 6)
38    {
39      arrVec = { list[1].toDouble(), 0.0 };
40      objNPV["X"] = arrVec;
41
42      arrVec = { list[2].toDouble(), 0.0 };
43      objNPV["Y"] = arrVec;
44
45      arrVec = { list[3].toDouble(), 0.0 };
46      objNPV["Z"] = arrVec;
47
48      arrVec = { list[4].toDouble(), 0.0 };
49      objNPV["Z2"] = arrVec;
50
51      *obj->insert(list[0], objNPV);
52    }
53  }
```

Listing 3.9: Funktion setJsonDocument()

3.4.3. Öffnen einer JSON-Datei

Zum Öffnen nützt die Anwendung die Funktion `loadFile`. Zuerst kopiert die Methode `copyFile` die Datei hinter der Variable `propNcPath` von der NCK auf die Speicherkarte unter der Variable `propPath`. Der Wahrheitswert `true` gibt an, dass die Funktion eine Datei mit gleichem Namen im Zielverzeichnis überschreibt. Das Objekt `file` ist von der Klasse `QFile`. Falls dieses nicht geöffnet werden kann, endet die Initiierung mit einer Fehlermeldung. Den Inhalt der Datei schreibt die Funktion in eine Klasse `QByteArray`. Danach schließt die Methode `close` die Datei und die Methode `remove` löscht die Kopie wieder vom Zielordner. Dann wandelt die Methode `fromJson` das Objekt `byteArray` der Klasse `QByteArray` in ein JSON-Dateiformat um und speichert die Datei unter der Adresse des Objekts `doc`. [18]

bool fileManager::loadFile	(QString ncPath, QString cardPath, QJsonDocument* doc)
Parameter	**Bedeutung**
QString ncPath	Gibt den Pfad auf der NCK an.
QString cardPath	Gibt den Pfad auf der Speicherkarte an.
QJsonDocument* doc	Diese Variable enthält die geöffnete Datei im JSON-Format.

Tabelle 3.10.: Funktion loadFile().

```
bool fileManager::loadFile(QString ncPath, QString cardPath, QJsonDocument
    * doc)
{
  qDebug() << "Read Json File:";

  fileServer.copyFile(ncPath, cardPath, true);

  //1. Open the QFile and write it to a byteArray and close the file
  QFile file(cardPath);
  if (!file.open(QIODevice::ReadOnly)) {
    qDebug() << "Json filef couldn't be opened/found";

    return false;
  }
  QByteArray byteArray;
  byteArray = file.readAll();
  file.close();

  fileServer.remove(cardPath);

  //2. Format the content of the byteArray as QJsonDocument
  //and check on parse Errors
  QJsonParseError parseError;

  *doc = QJsonDocument::fromJson(byteArray, &parseError);

  if (parseError.error != QJsonParseError::NoError)
  {
    qWarning() << "Parse error at " << parseError.offset << ":" <<
   parseError.errorString();

    return false;
```

```
31    }
32    return true;
33 }
```

Listing 3.10: Funktion loadFile()

3.4.4. Feststellen der Nullpunktverschiebung

Mit der Funktion `fillComboBox` durchsucht die Anwendung das Objekt `docNPV` der Klasse `QJsonDocument` nach allen möglichen NPV. Danach fügt die Methode `addItem` die NPVs in ein ausklappbares Eingabefeld mit dem Objekt `comboBox` hinzu. Nun stellt die Methode `setCurrentIndex` das ausklappbare Eingabefeld auf sichtbar. Am Ende beschreibt die Methode `setText` ein Textfeld mit dem Objekt `label_30` und der aktuellen NPV.

```
1  void mainForm::fillComboBox()
2  {
3    comboBox->clear();
4    for (size_t i = 1; i < 10; i++)
5    {
6      if (docNPV.object().contains(QString::number(i)))
7      {
8        comboBox->addItem("G5" + QString::number(i + 3));
9      }
10   }
11   for (size_t i = 10; i < 100; i++)
12   {
13     if (docNPV.object().contains(QString::number(i)))
14     {
15       comboBox->addItem("G5" + QString::number(i));
16     }
17   }
18   comboBox->setCurrentIndex(1);
19   label_30->setText(comboBox->currentText());
20 }
```

Listing 3.11: Funktion fillComboBox()

3.4.5. Ermitteln der Werte für die Nullpunktverschiebung

Die Funktion `getValues` liest die Werte der NPVs von dem Objekt `docNPV` aus. Dazu fragt die `if`-Verzweigung ab, ob eine NPV ausgewählt ist. Die aktuelle NPV zeigt die Anwendung

mit dem Textfeld und dem Objekt `label_30` an. Alle x-, y-, z-Werte, Korrekturwerte und den Versatz schreibt die Funktion in ein Array. Die Verdrehungen sind als Variablen gespeichert.

```
void mainForm::getValues()
{
  if (comboBox->count() > 0)
  {
    npv = label_30->text();
    npv = npv.remove(0, 2);

    vector[0] = docNPV.object().value(npv).toObject().value("x").toArray()
    [0].toDouble();
    vector[1] = docNPV.object().value(npv).toObject().value("y").toArray()
    [0].toDouble();
    vector[2] = docNPV.object().value(npv).toObject().value("z").toArray()
    [0].toDouble();
    vector[3] = 1;

    vectorR[0] = docNPV.object().value(npv).toObject().value("x").toArray
    ()[1].toDouble();
    vectorR[1] = docNPV.object().value(npv).toObject().value("y").toArray
    ()[1].toDouble();
    vectorR[2] = docNPV.object().value(npv).toObject().value("z").toArray
    ()[1].toDouble();
    vectorR[3] = 1;

    alpha = docNPV.object().value(npv).toObject().value("alpha").toDouble
    ();
    beta = docNPV.object().value(npv).toObject().value("beta").toDouble();
    offsetVector[0] = double(-1.0) * vector[0];
    offsetVector[1] = double(-1.0) * vector[1];
    offsetVector[2] = double(-1.0) * vector[2];
    offsetVector[3] = 1;
  }
}
```

Listing 3.12: Funktion getValues()

3.4.6. Ermitteln der Einheitsbasisvektoren

Die Funktion `unitVector` berechnet nach Gl. 2.19 die Basiseinheitsvektoren der Matrix. Zuerst errechnet die `for`-Schleife die Norm mit der Variable `length` des Spaltenvektors.

Danach wird der Spaltenvektor mit der `for`-Schleife normiert.

void matCal::unitVector	(double inputMatrix[4][4], double (*outputMatrix)[4])
Parameter	**Bedeutung**
double inputMatrix[4][4]	Übergibt die zu berechnende Matrix.
double (*outputMatrix)[4]	Gibt den Spaltenvektor an, wo die Ergebnisse hin gespeichert werden.

Tabelle 3.11.: Funktion unitVector().

```
1  void matCal::unitVector(double inputMatrix[4][4], double (*outputMatrix)
       [4])
2  {
3    for (size_t i = 0; i < 3; i++)
4    {
5      double length = 0.0;
6      for (size_t j = 0; j < 3; j++)
7      {
8        length += inputMatrix[j][i] * inputMatrix[j][i];
9      }
10
11     length = sqrt(length);
12     for (size_t j = 0; j < 3; j++)
13     {
14       outputMatrix[j][i] = inputMatrix[j][i] / length;
15     }
16   }
17
18
19   for (size_t i = 0; i < 4; i++)
20   {
21     outputMatrix[i][3] = inputMatrix[i][3];
22   }
23 }
```

Listing 3.13: Funktion unitVector()

3.4.7. Determinante bestimmen

Durch die Funktion `determinate` wird die Determinante nach Gl. 2.9 berechnet. Die `for`-Schleife läuft dreimal durch und nützt die Regel von Sarrus. Dabei gibt der Operator `%` den

Rest zurück.

double matCal::determinant (double matrix[4][4])	
Parameter	**Bedeutung**
double matrix[4][4]	Übergibt die zu berechnende Matrix.
return	Gibt den Wert der Determinante zurück.

Tabelle 3.12.: Funktion determinant().

```
double matCal::determinant(double matrix[4][4])
{
  double determinat = 0;

  for (size_t i = 0; i <= 2; i++)
  {
    determinat += matrix[i % 3][0] * matrix[(i + 1) % 3][1] * matrix[(i + 2) % 3][2];
    determinat -= matrix[0][(i + 2) % 3] * matrix[1][(i + 1) % 3] * matrix[2][i % 3];
  }

  return determinat;
}
```

Listing 3.14: Funktion determinate()

3.4.8. Invertieren

Mit der Funktion `inversMatrix` berechnet die Anwendung die Inverse der Matrix nach Gl. 2.12 . Die Anwendung übergibt der Funktion die Determinante, die zu invertierende Matrix und die neue Matrix. Dabei nützt die Funktion die Adjunkte der Matrix und transponiert sie. Außerdem dividiert die Funktion die transponierte Matrix durch die Determinante.

void matCal::inversMatrix	(double determinant, double inputMatrix[4][4], double (*outputMatrix)[4])
Parameter	**Bedeutung**
double determinant	Gibt die Determinante der Matrix an.
double inputMatrix[4][4]	Die zu berechnende Matrix.
double (*outputMatrix)[4]	Übergibt die inverse Matrix zurück.

Tabelle 3.13.: Funktion inversMatrix().

```
1  void matCal::inversMatrix(double determinant, double inputMatrix[4][4],
       double (*outputMatrix)[4])
2  {
3    for (int i = 0; i <= 2; i++)
4    {
5      for (int j = 0; j <= 2; j++)
6      {
7        outputMatrix[i][j] = 0;
8        outputMatrix[i][j] = inputMatrix[(j + 1) % 3][(i + 1) % 3] *
       inputMatrix[(j + 2) % 3][(i + 2) % 3] - inputMatrix[(j + 2) % 3][(i +
       1) % 3] * inputMatrix[(j + 1) % 3][(i + 2) % 3];
9        outputMatrix[i][j] = outputMatrix[i][j] / determinant;
10     }
11   }
12   outputMatrix[3][3] = 1;
13 }
```

Listing 3.15: Funktion inversMatrix()

3.4.9. Eine neue JSON-Datei erstellen

Die Funktion `writeJsonFile` schreibt das JSON-Dateiformat in eine Datei. Mit der Methode `toJson` beschreibt die Funktion das Objekt `byteArray` der Klasse `QByteArray`. Danach erzeugt die Methode `createFile` eine neue Datei auf der Speicherkarte, die von der Methode `open` geöffnet wird. Die Methode `write` beschreibt die Datei mit dem Objekt `byteArray`. Danach schließt die Methode `close` die Datei wieder. Die Methode `moveFile` verschiebt die neue Datei von der Speicherkarte auf die NCK.

void fileManager::writeJsonFile	(QJsonDocument doc, QString cardPath, QString ncPath)
Parameter	**Bedeutung**
QJsonDocument doc	Gibt die Variable mit dem JSON-Format an.
QString cardPath	Der Pfad zum Zwischenspeichern auf der Speicherkarte.
QString ncPath	Der Pfad zur Datei auf der NCK.

Tabelle 3.14.: Funktion writeJsonFile().

```
1  void fileManager::writeJsonFile(QJsonDocument doc, QString cardPath,
       QString ncPath)
2  {
3    QByteArray byteArray;
4
5    byteArray = QJsonDocument(doc).toJson();
6
7    fileServer.createFile(cardPath, true);
8
9    QFile file(cardPath);
10   file.open(QIODevice::WriteOnly | QIODevice::Append);
11   file.write(byteArray);
12   file.close();
13
14   fileServer.moveFile(cardPath, ncPath, true);
15 }
```

Listing 3.16: Funktion writeJsonFile()

3.4.10. Ermitteln der Rotationsmatrix

Die Funktion `getRotMatrix` erstellt die NPV-spezifische Rotationsmatrix. Durch die `if`-Abfrage kontrolliert die Funktion, ob die NPV möglich ist. Danach erstellt die Funktion eine neue Rotationsmatrix mit den Alpha und Beta Werten der NPV.

void matCal::getRotMatrix	(QString npv, double alpha, double beta, double(*outputMatrix)[4])
Parameter	**Bedeutung**
QString npv	Gibt die Nullpunktverschiebung an.
double alpha	Der Winkel Alpha um die x-Achse.
double beta	Der Winkel Beta um die y-Achse.
double(*outputMatrix)[4]	Übergibt das Objekt, in das die Rotationsmatrix gespeichert wird.

Tabelle 3.15.: Funktion getRotMatrix().

```
1  void matCal::getRotMatrix(QString npv, double alpha, double beta, double(*
       outputMatrix)[4])
2  {
3    if (npv > 10 && npv < 99)
4    {
5      alpha = alpha * PI / 180;
6      beta = beta * PI / 180;
7
8      outputMatrix[0][0] = cos(beta);
9      outputMatrix[1][0] = -sin(alpha) * sin(beta);
10     outputMatrix[2][0] = -cos(alpha) * sin(beta);
11     outputMatrix[3][0] = 0;
12
13     outputMatrix[0][1] = 0;
14     outputMatrix[1][1] = cos(alpha);
15     outputMatrix[2][1] = -sin(alpha);
16     outputMatrix[3][1] = 0;
17
18     outputMatrix[0][2] = sin(beta);
19     outputMatrix[1][2] = sin(alpha) * cos(beta);
20     outputMatrix[2][2] = cos(alpha) * cos(beta);
21     outputMatrix[3][2] = 0;
22
23     outputMatrix[0][3] = 0;
24     outputMatrix[1][3] = 0;
25     outputMatrix[2][3] = 0;
26     outputMatrix[3][3] = 1;
27   }
28 }
```

Listing 3.17: Funktion getRotMatrix()

Die gleiche Funktion wird auch für die Rotationsmatrix zur Rückrechnung der Korrekturwerte verwendet. Dazu ruft die Funktion die andere Variante auf. Danach berechnet die Funktion `transformation` die transponierte Matrix. Diese erweitert die Funktion noch mit dem Offset der NPV.

void matCal::getRotMatrix	(QString npv, double alpha, double beta, double offsetVector[4], double(*outputMatrix)[4])
Parameter	**Bedeutung**
QString npv	Gibt die Nullpunktverschiebung an.
double alpha	Der Winkel Alpha um die x-Achse.
double beta	Der Winkel Beta um die y-Achse.
double offsetVector[4]	Gibt den Spaltenvektor mit der Differenz der beiden Nullpunkte an.
double(*outputMatrix)[4]	Übergibt das Objekt, in das die Rotationsmatrix gespeichert wird.

Tabelle 3.16.: Funktion getRotMatrix().

```
1  void matCal::getRotMatrix(QString npv, double alpha, double beta, double
       offsetVector[4], double(*outputMatrix)[4])
2  {
3    double outputVector[4] = { 0 };
4
5    getRotMatrix(npv, alpha, beta, outputMatrix);
6
7    transformation(offsetVector, outputMatrix, outputVector);
8
9    for (size_t i = 0; i < 4; i++)
10   {
11     outputMatrix[i][3] = outputVector[i];
12   }
13 }
```

Listing 3.18: Funktion getRotMatrix()

3.4.11. Transformation der Nullpunktverschiebung

Mit der Funktion `transformation` erfolgt die Transformation eines Vektors. Die Funktion führt eine Matrixmultiplikation mit einem Vektor durch. Dafür nützt die Funktion zwei `for`-Schleifen, die erste `for`-Schleife für die Spalten der Matrix und die zweite für die Zeilen.

Für das Bezugssystem der NPV gilt:

void matCal::transformation	(double vector[4], double matrix[4][4], double *outputVector)
Parameter	**Bedeutung**
double vector[4]	Gibt den zu transformierenden Vektor an.
double matrix[4][4]	Gibt die Transformationsmatrix an.
double *outputVector	In diesem Objekt wird der transformierte Vektor gespeichert.

Tabelle 3.17.: Funktion transformation().

```
1  void matCal::transformation(double vector[4], double matrix[4][4], double
       *outputVector)
2  {
3    for (size_t i = 0; i < 4; i++)
4    {
5      outputVector[i] = 0;
6      vector;
7      for (size_t j = 0; j < 4; j++)
8      {
9        outputVector[i] += matrix[i][j] * vector[j];
10     }
11   }
12 }
```

Listing 3.19: Funktion transformation()

Für ein definiertes Bezugssystem:

void matCal::transformation	(double vector[4], double *outputVector, QJsonDocument doc, QString cos, QString matrix)
Parameter	**Bedeutung**
double vector[4]	Gibt den zu transformierenden Vektor an.
double *outputVector	In diesem Objekt wird der transformierte Vektor gespeichert.
QJsonDocument doc	Übergibt das Objekt mit allen Koordinatensystemen ins JSON-Format.
QString cos	Gibt das aktive Koordinatensystem an.
QString matrix	Gibt an, welche Matrix berücksichtigt werden soll (Normale, Invertierte).

Tabelle 3.18.: Funktion transformation().

```
void matCal::transformation(double vector[4], double *outputVector,
    QJsonDocument doc, QString cos, QString matrix)
{
  for (size_t i = 0; i < 4; i++)
  {
    outputVector[i] = 0;
    for (size_t j = 0; j < 4; j++)
    {
      if (doc.object().value(cos).toObject().value(matrix).toArray()[i].
    toArray()[j].isDouble())
      {
        outputVector[i] += doc.object().value(cos).toObject().value(matrix
    ).toArray()[i].toArray()[j].toDouble() * vector[j];
      }
      else
      {
        outputVector[i] += 0;
      }
    }
  }
}
```

Listing 3.20: Funktion transformation()

3.4.12. Berechnen der Verschiebungen im beliebigen Koordinatensystem

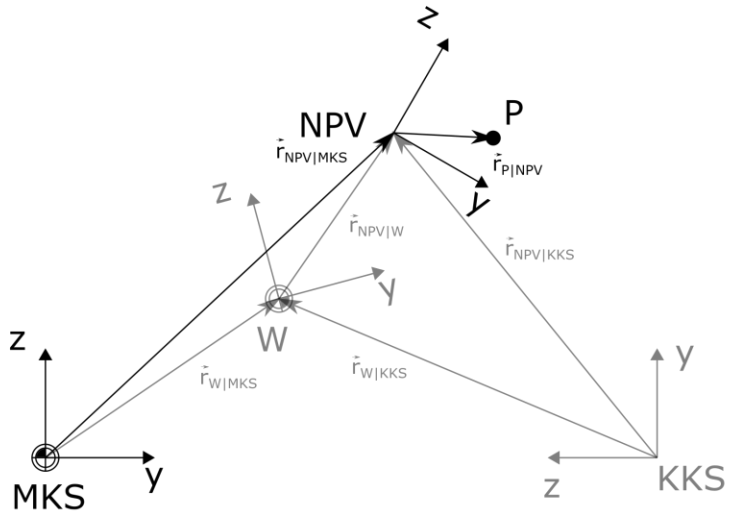


Abbildung 3.15.: NPV im MKS und einem ReferenzPointSystem

Der Ortsvektor vom KKS zum NPV lautet:

$$\vec{r}_{NPV\_KKS} = \vec{r}_{W\_KKS} + \vec{r}_{NPV\_W} \tag{3.12}$$

Dabei ist der Vektor $\vec{r}_{NPV\_W}$ in der Gl. (3.12) der Wert zur NPV vor der Drehung *NPV-vor*. Die Gl. (3.12) wird mit Matrizen angeschrieben. Dafür nützt die Matrix das KKS als Bezugssystem:

$$\underline{r}_{NPV\_KKS|KKS} = \underline{\underline{T}}^{KKS}_{MKS} \cdot \left(\underline{r}_{NPV\_MKS|MKS} - \underline{r}_{W\_MKS|MKS}\right) + \underline{r}_{W\_KKS|KKS} \tag{3.13}$$

$$\begin{bmatrix} x_{NPS\_KKS} \\ y_{NPS\_KKS} \\ z_{NPS\_KKS} \end{bmatrix}_{KKS} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}^{MKS}_{KKS} \cdot \left(\begin{bmatrix} x_{NPS\_MKS} \\ x_{NPS\_MKS} \\ x_{NPS\_MKS} \end{bmatrix}_{MKS} - \begin{bmatrix} x_{W\_MKS} \\ y_{W\_MKS} \\ z_{W\_MKS} \end{bmatrix}_{MKS} \right) + \begin{bmatrix} x_{W\_KKS} \\ y_{W\_KKS} \\ z_{W\_KKS} \end{bmatrix}_{KKS} \tag{3.14}$$

Da Matrix-Multiplikationen sehr gut zu programmieren sind, wird aus der 3×3 Matrix eine 4×4 Matrix gemacht. Alle Vektoren erhalten eine vierte Zeile mit dem Wert Eins. Die translatorische Verschiebung von $\underline{r}_{W\_KKS}$ kommt als vierte Spalte hinzu. Die restlichen Werte der vierten Zeilen sind Null:

$$\begin{bmatrix} x_{NPS\_KKS} \\ y_{NPS\_KKS} \\ z_{NPS\_KKS} \\ 1 \end{bmatrix}_{KKS} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & x_{W\_KKS} \\ b_{21} & b_{22} & b_{23} & y_{W\_KKS} \\ b_{31} & b_{32} & b_{33} & z_{W\_KKS} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{MKS}_{KKS} \cdot \left(\begin{bmatrix} x_{NPS\_MKS} - x_{W\_MKS} \\ x_{NPS\_MKS} - y_{W\_MKS} \\ x_{NPS\_MKS} - z_{W\_MKS} \\ 1 \end{bmatrix}_{KKS} \right) \tag{3.15}$$

Die Transformationsmatrix beschreibt das MKS im KKS. Damit transformiert die Matrix einen Vektor vom MKS ins KKS. Die Funktion `calculateValue` rechnet die ursprüngliche NPV in ein beliebiges Koordinatensystem um. Dafür nützt die Funktion `calculateValue` die Funktion `getRotMatrix` und die Funktion `transformation`. Diese führt die Funktion einmal für den Ortsvektor und einmal für die Korrekturwerte durch.

bool mainForm::calculateTransformation	(QString cos)
Parameter	**Bedeutung**
QString cos	Gibt das ausgewählte Koordinatensystem an.
return	Gibt den Erfolg der Funktion zurück. True: Die Funktion konnte ohne Problem durchgeführt werden. False: Die Funktion konnte NICHT durchgeführt werden.

Tabelle 3.19.: Funktion calculateTransformation().

```
1 bool mainForm::calculateTransformation(QString cos)
2 {
```

```
  setEnabled();

  if (comboBox->count() > 0)
  {
    if (cos == "NPV")
    {
      mat.getRotMatrix(npv, alpha, beta, offsetVector, matrix);
      mat.transformation(vector, matrix, transformedVector);

      mat.getRotMatrix(npv, alpha, beta, matrix);
      mat.transformation(vectorR, matrix, transformedVectorR);
      return true;
    }
    else if (docCoS.object().contains(cos))
    {
      mat.transformation(vector, transformedVector, docCoS, cos, "trans");
      mat.transformation(vectorR, transformedVectorR, docCoS, cos, "trans"
    );
      return true;
    }
  }
  setEmpty();
  return false;
}
```

Listing 3.21: Funktion calculateValue()

3.4.13. Korrekturen zurückrechnen

Die Korrekturen müssen vom KKS ins MKS zurück gerechnet werden. Dafür berechnet die Anwendung die Korrekturen mit der invertierten Transformationsmatrix:

$$\underline{r}_{Korr\_MKS|MKS} = \underline{\underline{T}}^{MKS}_{KKS} \cdot \underline{r}_{Korr\_KKS|KKS} \tag{3.16}$$

Die inverse und die transponierte Transformationsmatrix sind gleich, da die Transformationsmatrix drei Spaltenvektoren besitzt, die orthogonal zueinander sind:

$$\underline{\underline{T}}^{MKS}_{KKS} = \underline{\underline{T}}^{KKS\,-1}_{MKS} \tag{3.17}$$

Die Berechnung der Inverse erfolgt mit Hilfe der Gl. 2.11 für die Transformationsmatrix:

$$adj(\underline{\underline{T}}_{KKS}^{MKS}) = \begin{bmatrix} det\begin{vmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{vmatrix} & -det\begin{vmatrix} b_{21} & b_{23} \\ b_{31} & b_{33} \end{vmatrix} & det\begin{vmatrix} b_{21} & b_{22} \\ b_{31} & b_{32} \end{vmatrix} \\ -det\begin{vmatrix} b_{12} & b_{13} \\ b_{32} & b_{33} \end{vmatrix} & det\begin{vmatrix} b_{11} & b_{13} \\ b_{31} & b_{33} \end{vmatrix} & -det\begin{vmatrix} b_{11} & b_{12} \\ b_{31} & b_{32} \end{vmatrix} \\ det\begin{vmatrix} b_{12} & b_{13} \\ b_{22} & b_{23} \end{vmatrix} & -det\begin{vmatrix} b_{11} & b_{13} \\ b_{21} & b_{23} \end{vmatrix} & det\begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} \end{bmatrix}^T \tag{3.18}$$

Die Berechnung der Determinante erfolgt nach der Regel von Sarrus nach Gl. 2.9:

$$\begin{aligned} det(\underline{\underline{T}}_{KKS}^{MKS}) = \; & b_{11} \cdot b_{22} \cdot b_{33} + b_{12} \cdot b_{23} \cdot b_{31} + b_{13} \cdot b_{21} \cdot b_{32} - \\ & b_{13} \cdot b_{22} \cdot b_{31} - b_{11} \cdot b_{23} \cdot b_{32} - b_{12} \cdot b_{21} \cdot b_{33} \end{aligned} \tag{3.19}$$

$$\underline{\underline{T}}_{KKS}^{MKS^{-1}} = \frac{1}{det(\underline{\underline{T}}_{KKS}^{MKS})} \cdot adj(\underline{\underline{T}}_{KKS}^{MKS}) \tag{3.20}$$

Die Funktion `calculateCorrectionVectorR` rechnet die Korrektur in das MKS zurück. Dabei fragt die Methode `displayText` die eingegebenen Werte ab. Existiert noch keine Transformationsmatrix, erstellt die Methode `determinant` die Determinante. Die Methode `inversMatrix` berechnet die Inverse der Matrix und danach wird mit der Methode `transformation` die Transformationsmatrix berechnet. Die Methode `modifyJsonValue` ändert die Werte im JSON-Objekt.(Siehe Anhang C). [19]

void mainForm::calculateCorrectionVectorR	(QString cos)
Parameter	**Bedeutung**
QString cos	Gibt das ausgewählte Koordinatensystem an.

Tabelle 3.20.: Funktion calculateCorrectionVectorR().

```
1 void mainForm::calculateCorrectionVectorR(QString cos)
2 {
3   if (comboBox->count() > 0)
4   {
5     correctionVector[0] = lineEdit->displayText().toDouble();
```

```
    correctionVector[1] = lineEdit_2->displayText().toDouble();
    correctionVector[2] = lineEdit_3->displayText().toDouble();
    correctionVector[3] = 1;

    if (docCoS.object().contains(cos))
    {
      mat.transformation(correctionVector, correctionVectorNew, docCoS,
   cos, "invers");
    }
    else if (cos == "NPV")
    {
      determinant = mat.determinant(matrix);
      if (determinant != 0)
      {
        mat.inversMatrix(determinant, matrix, inverseMatrix);
        mat.transformation(correctionVector, inverseMatrix,
   correctionVectorNew);
      }
    }

    correctionVectorNew[0] += docNPV.object().value(npv).toObject().value(
   "x").toArray()[1].toDouble();
    correctionVectorNew[1] += docNPV.object().value(npv).toObject().value(
   "y").toArray()[1].toDouble();
    correctionVectorNew[2] += docNPV.object().value(npv).toObject().value(
   "z").toArray()[1].toDouble();

    docPath = jsonPath.arg(npv).arg("x").arg(1);
    write.modifyJsonValue(docNPV, docPath, correctionVectorNew[0]);

    docPath = jsonPath.arg(npv).arg("y").arg(1);
    write.modifyJsonValue(docNPV, docPath, correctionVectorNew[1]);

    docPath = jsonPath.arg(npv).arg("z").arg(1);
    write.modifyJsonValue(docNPV, docPath, correctionVectorNew[2]);
  }
}
```

Listing 3.22: Funktion correctR()

3.4.14. Schreiben einer JSON-Datei

Die Funktion `writeJsonFile` schreibt ein JSON-Objekt in eine Datei. Durch die Methode `toJson` definiert die Funktion ein Objekt `byteArray` von der Klasse `QByteArray` mit dem JSON-Objekt. Die Methode `createFile` erstellt eine Datei auf der Speicherkarte. Mit der Methode `open` öffnet die Funktion die davor erstellte Datei. Durch die Methode `write` wird die Datei mit dem JSON-Objekt beschrieben. Die Methode `close` schließt die Datei wieder. Mit der Methode `moveFile` verschiebt die Funktion die Datei von der Speicherkarte auf die NCK.

void fileManager::writeJsonFile	(QJsonDocument doc, QString cardPath, QString ncPath)
Parameter	**Bedeutung**
QJsonDocument doc	Gibt die zu beschreibende Variable im JSON-Format an.
QString cardPath	Gibt den Pfad zum Zwischenspeicher auf der Speicherkarte an.
QString ncPath	Gibt den Pfad der Datei auf der NCK an.

Tabelle 3.21.: Funktion writeJsonFile().

```
1  void fileManager::writeJsonFile(QJsonDocument doc, QString cardPath,
       QString ncPath)
2  {
3    QByteArray byteArray;
4
5    byteArray = QJsonDocument(doc).toJson();
6
7    fileServer.createFile(cardPath, true);
8
9    QFile file(cardPath);
10   file.open(QIODevice::WriteOnly | QIODevice::Append);
11   file.write(byteArray);
12   file.close();
13
14   fileServer.moveFile(cardPath, ncPath, true);
15 }
```

Listing 3.23: Funktion writeJsonFile()

3.4.15. Schreiben eines Werkstücknullpunkts

Die Funktion `writeW` schreibt den Werkstücknullpunkt in eine Datei. Mit der Methode `open` öffnet die Funktion die Datei. Durch die Klasse `QTextStream` und das Objekt `out` beschreibt die Funktion die Datei. Zuerst wird ein Kommentar mit der Bauteilverschiebung erstellt, dann fragt die Funktion die Werte für den Werkstücknullpunkt ab. Danach schreibt die Funktion den Befehl zur Nullpunktsetzung. Am Ende schließt die Methode `close` die Datei wieder.

void fileManager::writeW	(QString cardPath, QJsonDocument *doc)
Parameter	**Bedeutung**
QString cardPath	Gibt den Pfad zum Zwischenspeicher auf der Speicherkarte an.
QJsonDocument *doc	Übergibt der Funktion das Objekt, in der das Ergebnis im JSON-Objekt gespeichert wird.

Tabelle 3.22.: Funktion writeW().

```
1  void fileManager::writeW(QString cardPath, QJsonDocument *doc)
2  {
3    QString header = ";----------------------------------G5
       %1----------------------------------";
4
5    int W = doc->object().value("setup").toObject().value("W").toInt();
6    double vector[4] = { 0 };
7
8    QFile file(cardPath);
9    file.open(QIODevice::WriteOnly | QIODevice::Append);
10   QTextStream out(&file);
11   out << header.arg(W + 3) << endl;
12
13   QString text = "$P_UIFR[%1] =  CTRANS(X, %2 + R1000, Y, %3 + R1001, Z,
       %4 + R1002)";
14
15   vector[0] = doc->object().value(QString::number(W)).toObject().value("X"
       ).toArray()[0].toDouble();
16   vector[0] += doc->object().value(QString::number(W)).toObject().value("X
       ").toArray()[1].toDouble();
17   vector[1] = doc->object().value(QString::number(W)).toObject().value("Y"
       ).toArray()[0].toDouble();
```

```
18    vector[1] += doc->object().value(QString::number(W)).toObject().value("Y
       ").toArray()[1].toDouble();
19    vector[2] = doc->object().value(QString::number(W)).toObject().value("Z"
       ).toArray()[0].toDouble();
20    vector[2] += doc->object().value(QString::number(W)).toObject().value("Z
       ").toArray()[1].toDouble();
21
22    out << text.arg(W).arg(vector[0]).arg(vector[1]).arg(vector[2]) << "\n"
       << endl;
23
24    file.close();
25  }
```

Listing 3.24: Funktion writeW()

3.4.16. Schreiben der Nullpunktverschiebungen

Mit der Funktion `writeNPV` werden die NPVs in eine Datei geschrieben. Dabei öffnet die Methode `open` die Datei. Durch eine `for`-Schleife und eine `if`-Abfrage ermittelt die Funktion alle NPVs. Mit Hilfe von der Klasse `QTextStream` und dem Objekt `out` beschreibt die Funktion die Datei. Vor jedem NPV-Eintrag steht ein Kommentar mit der Nummer. Danach werden die NPV-Befehle geschrieben. Am Ende schließt die Methode `close` die Datei wieder.

void fileManager::writeNPV	(QString cardPath, QJsonDocument *doc)
Parameter	**Bedeutung**
QString cardPath	Gibt den Pfad zum Zwischenspeicher auf der Speicherkarte an.
QJsonDocument *doc	Übergibt der Funktion das Objekt, in der das Ergebnis im JSON-Objekt gespeichert wird.

Tabelle 3.23.: Funktion writeNPV().

```
1  void fileManager::writeNPV(QString cardPath, QJsonDocument *doc)
2  {
3    int W = doc->object().value("setup").toObject().value("W").toInt();
4    double alpha;
5    double beta;
6    int meaf;
7    int sp;
```

```
  int bea;
  /* X, Y, Z, Z2, A, B1, B2, X2, U, V, W, SP1, SP2*/
  double array[13] = { 0 };
  QString header = ";---------------------------------G5
   %1---------------------------------";
  QString text = "$P_UIFR[%1] =  CTRANS(X, %2, Y, %3, Z, %4)";

  QFile file(cardPath);
  file.open(QIODevice::WriteOnly | QIODevice::Append);

  QTextStream out(&file);

  for (size_t npv = 10; npv < 100; npv++)
  {
    if (doc->object().contains(QString::number(npv)))
    {
      getValues(doc, npv, array, &alpha, &beta, &meaf, &sp, &bea);
      TRAF traf(npv, W, array, alpha, beta, meaf, sp, bea, doc);
      out << header.arg(QString::number(npv)) << endl;
      out << text.arg(QString::number(npv)).arg(array[0]).arg(array[1]).
   arg(array[2]);

      if (W > 0)
      {
        out << ((QString) " : $P_UIFR[%5]").arg(W);
      }
      out << "\n" << endl;
    }
  }
  out << "M30";
  file.close();
}
```

Listing 3.25: Funktion writeNPV()

3.4.17. Schreiben eines Unterprogramms(SPF) für die Steuerung

Die Funktion `writeSubProgram` erstellt das neue Unterprogramm mit den NPVs. Mit der Methode `createFile` wird die Unterprogrammdatei auf der Speicherkarte erstellt. Wenn das NC-Programm einen Werkstücknullpunkt besitzt, schreibt die Methode `writeW` diesen in das Unterprogramm. Danach befüllt die Methode `writeNPV` das Unterprogramm mit den

NPVs. Am Ende verschiebt die Methode `moveFile` die Datei von der Speicherkarte auf die NCK.

void fileManager::writeSubProgram	(QJsonDocument doc, QString cardPath, QString ncPath)
Parameter	**Bedeutung**
QJsonDocument doc	Übergibt die Variable mit den Werten in das JSON-Format.
QString cardPath	Gibt den Pfad zum Zwischenspeicher auf der Speicherkarte an.
QString ncPath	Gibt den Pfad an, wohin die Dateien auf der NCK gespeichert werden sollen.

Tabelle 3.24.: Funktion writeSubProgram().

```
1  void fileManager::writeSubProgram(QJsonDocument doc, QString cardPath,
       QString ncPath)
2  {
3    fileServer.createFile(cardPath, true);
4
5    if (doc.object().value("setup").toObject().value("W").toInt() > 0)
6    {
7      writeW(cardPath, &doc);
8    }
9    writeNPV(cardPath, &doc);
10   fileServer.moveFile(cardPath, ncPath, true);
11 }
```

Listing 3.26: Funktion writeSubProgram()

3.5. Prüfen der Anwendung

3.5.1. Prüfverfahren

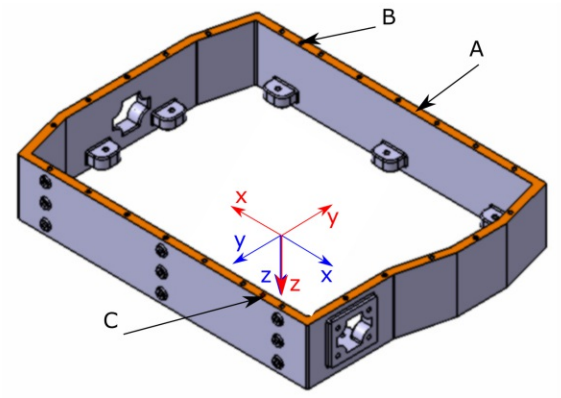
Mit der Abnahmeuntersuchung der Werkzeugmaschine bewertet der_die Kunde_in die Realisierung der Bauteilqualität oder der Zykluszeit zur Fertigung eines Bauteils. Eine wichtige Methode ist die Fertigung von Prüfwerkstücken und deren Vermessung. Für die erfolgreiche Maschinenabnahme definieren der Maschinenhersteller und der_die Kunde_in das Prüfwerkstück und die Beurteilungskriterien. Dabei können Einmal-, Kurzzeit-, Langzeit-,

oder periodisch wiederholende Prüfungen durchgeführt werden. In der Serienfertigung nützt der_die Kunde_in Kurzzeit- und Langzeitfähigkeits-Messungen zur Beurteilung von Maschinen. Neben dem zeitlichen Aspekt umfassen diese auch die Einflüsse von Prozess- und Umweltbedingungen. Im Gegensatz zur direkten Messung ist die Fertigung von Prüfwerkstücken schnell und ohne aufwendige Messtechnik an der Maschine durchführbar. Außerdem kann die Fertigung von Prüfwerkstücken anschauliche Aussagen über die Fertigungsgenauigkeit geben. Bei der Fertigung von Prüfwerkstücken werden auch die realen Umgebungsbedingungen berücksichtigt. Prüfwerkstücke eignen sich jedoch nicht, um Werkzeugmaschinen untereinander zu vergleichen, da sich maschinenbedingte und nicht-maschinenbedingte sowie wiederholbare und nicht wiederholbare Einflüsse summarisch auswirken. Genauso kann der Werkstoff des Werkstücks, die Aufspannung, das Werkzeug, die Prozessbedingungen, die Umweltbedingungen und der_die Maschinenbediener_in die Ergebnisse beeinflussen. [20]

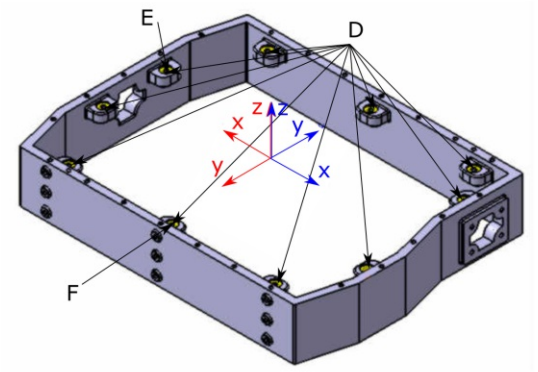
Prüfwerkstück

Zum Überprüfen des Programms wurde die Fertigung eines Prüfwerkstücks begleitet. Dafür fertigt die Maschine das Prüfwerkstück zum ersten Mal. Später sollte die Maschine 500 Stück vom Bauteil fertigen. Für die Produktion muss ein_e Mitarbeiter_in das NC-Programm noch an die Maschine und deren Aufspannung anpassen. Dafür fertigt diese_r ein Bauteil, vermisst dieses und übernimmt die Anpassung im NC-Code.
Als Bauteil steht ein modulares Akkugehäuse eines LKW-Herstellers zur Verfügung. Abb. 3.16 zeigt eine auf die wichtigsten Merkmale reduzierte Nachbildung dieses Bauteils. Das Gehäuse ist ein rechteckiger Rahmen, in dem die Akkuzellen liegen. Dabei kann der Akku durch die modulare Bauart unterschiedlich groß gestaltet werden.
In der ersten Aufspannung bearbeitet die Maschine das Bauteil von vier Seiten, die vier außenliegenden Seiten sind in Abb. 3.17(a) mit den Ebenen der NPV dargestellt. Danach dreht der_die Maschinenbediener_in das Bauteil. In der zweiten Aufspannung fertigt die Maschine die restlichen zwei Seiten, welche in Abb. 3.17(b) dargestellt sind. Damit besitzt das Programm sechs NPVs.

Das Bauteil wird mit einer Spindel bearbeitet. Dabei berücksichtigt die Maschine die Wärmeausdehnung nicht, da immer nur ein Prüfwerkstück zum Vermessen gefertigt wird. Die erste Messung zur Berücksichtigung der Wärmeausdehnung findet in der Maschine nach 30 Minuten statt. Das Fertigen eines Prüfwerkstücks benötigt für die erste Seite 12 Minuten 58 Sekunden und für die zweite Seite 9 Minuten 16 Sekunden. Somit reicht die Zeit nicht eine Messung der Wärmeausdehnung durchzuführen.



(a) Erste Bearbeitungsseite



(b) Zweite Bearbeitungsseite

Abbildung 3.16.: ISO Ansicht Akkugehäuse

Tabelle 3.25 zeigt die NPVs von den Listings 3.27 und 3.28 zusammengefasst und wie sie der

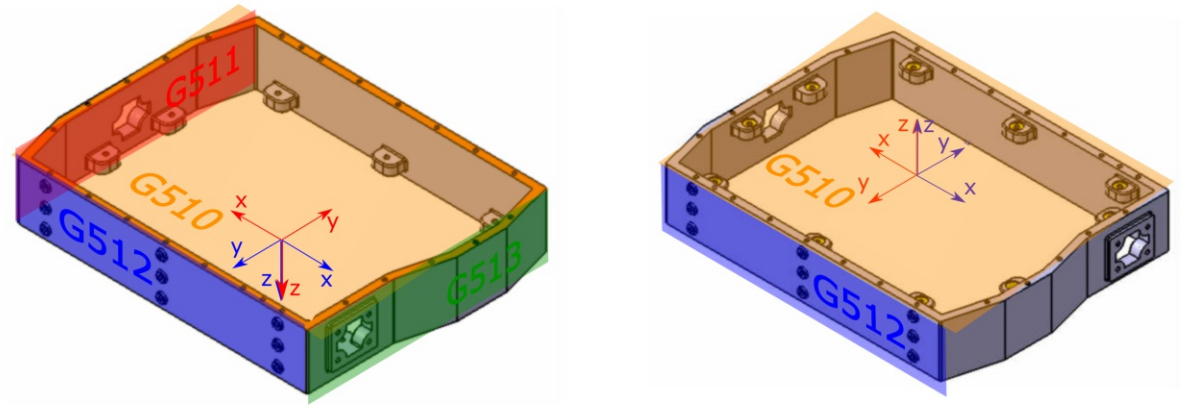
Die Ausgangswerte:

erste Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-327,957 mm	612,5 mm	222,0 mm	0°	0°
G511	-327,957 mm	612,5 mm	222,0 mm	0°	270°
G512	-327,957 mm	612,5 mm	222,0 mm	−90°	180°
G513	-327,957 mm	612,5 mm	222,0 mm	0°	90°
zweite Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	287,5 mm	-40,5 mm	-60,0 mm	0°	180°
G512	287,5 mm	-40,5 mm	-60,0 mm	−90°	0°

Tabelle 3.25.: Nullpunktverschiebungen.

Steuerung mitgeteilt werden. Dabei sind Listing 3.27 und 3.28 vom CAM-System erstellte Bearbeitungsprogramme mit den NPVs.

Abb. 3.17(a) zeigt die NPVs G510, G511, G512 und G513 im Modell eingezeichnet. In Abb. 3.17(b) ist die zweite Aufspannung mit den NPVs G510 und G512 dargestellt.



(a) Erste Aufspannung mit den NPVs

(b) Zweite Aufspannung mit den NPVs

Abbildung 3.17.: ISO Ansicht der NPV

```
$P_UIFR[10]=CFINE(X,0.2,Y,0,Z,0,A,0,B,0)
TRAF(10,0,-327.957,612.5,222,0,0,0,0,0,0,1,1)
;;;***************************************************

;;;***************************************************
$P_UIFR[11]=CFINE(X,0,Y,0,Z,-0.2,A,0,B,0)
TRAF(11,0,-327.957,612.5,222,0,270,0,0,0,0,1,1)
;;;***************************************************

;;;***************************************************
$P_UIFR[12]=CFINE(X,-0.2,Y,-0.2,Z,0,A,0,B,0)
TRAF(12,0,-327.957,612.5,222,-90,180,0,0,0,0,1,1)
;;;***************************************************

;;;***************************************************
$P_UIFR[13]=CFINE(X,0,Y,0,Z,0.2,A,0,B,0)
TRAF(13,0,-327.957,612.5,222,0,90,0,0,0,0,1,1)
```

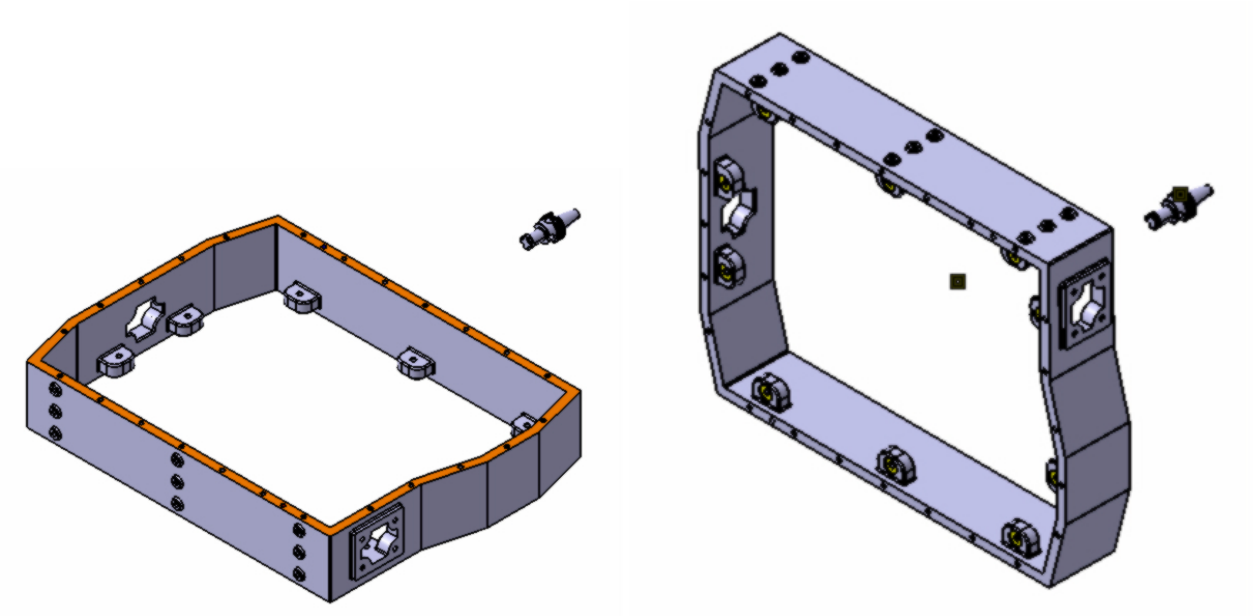
Listing 3.27: NPV erste Aufspannung

```
$P_UIFR[10]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
TRAF(10,0,287.5,-40.5,-60,0,180,0,0,0,0,1,1)
;;;***************************************************

;;;***************************************************
$P_UIFR[12]=CFINE(X,0,Y,-0.2,Z,0,A,0,B,0)
```

```
7 TRAF(12,0,287.5,-40.5,-60,-90,0,0,0,0,0,1,1)
```

Listing 3.28: NPV zweite Aufspannung



(a) Bauteil vor der Drehung (b) Bauteil nach der Drehung in der Position G510

Abbildung 3.18.: Drehung des Bauteils in der Maschine

Abb. 3.18 zeigt, wie das Bauteil gegenüber dem Fräser gedreht wird. Dadurch befinden sich die NPVs für Bearbeitungen an neuen Positionen. Deshalb rechnet der TRAF-Befehl die NPV um.

Die Tabelle 3.26 zeigt die umgerechneten NPVs im MKS nach der Rotation, da die NPVs des Bauteils nach der Rotation um die A- und B-Achse an anderen Positionen im MKS liegen als vor der Rotation. Mit diesen NPVs arbeitet die Maschine das NC-Programm ab. Diese Umrechnung erledigt die Steuerung mit dem schon vorhandenen TRAF-Befehl.

Nach der Rotation durch den TRAF-Befehl befinden sich die NPVs an folgenden Positionen im MKS:

erste Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-327,957 mm	612,5 mm	222,0 mm	0°	0°
G511	222,0 mm	612,5 mm	327.957 mm	0°	270°
G512	327,957 mm	222,0 mm	612,5 mm	−90°	180°
G513	-222,0 mm	612,5 mm	-327,957 mm	0°	90°
zweite Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-287,5 mm	-40,5 mm	60,0 mm	0°	180°
G512	287,5 mm	60,0 mm	-40,5 mm	−90°	0°

Tabelle 3.26.: NPV nach der Rotation im MKS.

3.5.2. Messmittel

Zur Beurteilung der Bauteilqualität wird oft die 3D-Koordinatenmessmaschine herangezogen. Die Koordinatenmessmaschine bezieht Punkt von der Bauteiloberfläche auf ein definiertes Koordinatensystem. Die zu bestimmenden Größen bekommt man durch eine computergestützte Ausgleichsrechnung. Dadurch können nicht nur Längen und Winkel bestimmt werden, sondern auch die Lage von Bauteilmerkmalen und die Lage zueinander. Die Koordinatenmessmaschine vergleicht auch die gemessene Ist-Geometrie mit der Soll-Geometrie. Die Soll-Geometrie stellt die CAD-Datei bereit. [20]

Vermessung

Auf der Messmaschine vermisst der_die Messtechniker_in zuerst die Auflagepunkte des Werkstücks in der Werkzeugmaschine. Dadurch ergibt sich das Bezugssystem des Rohteils. Von diesem Bezugssystem vermisst der_die Messtechniker_in weitere Bezugsflächen, -achsen und -punkte. Diese gibt der_die Konstrukteur_in schon in der Fertigungszeichnung an. Das Werkstück besitzt zwei Bezugssysteme. Das erste Bezugssystem ist in Abb. 3.16(a) dargestellt und besteht aus einer Bezugsebene A und den zwei diagonal gegenüberliegenden Zentrierbohrungen als Bezugsachse B und C. Der Winkel zwischen der xz-Ebene und der Linie zwischen den Zentrierbohrungen ist bekannt. Somit ergibt sich ein Bezugssystem. Auf dieses Bezugssystem bezieht die Messmaschine die Bohrungen auf der Bezugsebene A bzw. der Ebene von G510.

Das zweite Bezugssystem ist in Abb. 3.16(b) dargestellt und besteht aus der Bezugsebene D, welche parallel zur Bezugsebene A liegt und den zwei Bohrungen mit der Bezugsachse E und F.
Die vermessenen Merkmale beziehen sich immer auf eines der beiden Bezugssysteme. Im Messprogramm sind die Soll-Werte der Merkmale hinterlegt. Mit den gemessenen Ist-Werten kann die Maschine die Abweichung ermitteln. Im Messbericht steht die Abweichung jedes Merkmals zum Soll-Wert. Außerdem wird die zulässige Toleranz angegeben. Haben die Merkmale mit derselben NPV ähnlich große Abweichungen in dieselbe Richtung, lohnt sich eine Korrektur der NPV.

3.5.3. Korrektur

Das KKS ist in Abb. 3.16 blau dargestellt und ist 180° um die z-Achse verdreht gegenüber dem rot dargestellten MKS. Der x-Vektor des MKS zeigt in die negative x-Achse des KKS. Der y-Vektor des MKS zeigt in die negative y-Achse des KKS. Der z-Vektor des MKS zeigt in die positive z-Achse des KKS. Damit ergibt sich die Transformationsmatrix:

$$\underline{\underline{T}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

Diese Transformationsmatrix gibt der_die Mitarbeiter_in in den Screen für die Bearbeitung des Bezugssystems ein. Dies ist in Abb. 3.19 dargestellt.

Im gemessenen KKS kommt man zu folgenden Ergebnissen:
Die Merkmale passen bezogen auf die NPV der ersten Aufspannung in x-Richtung um $0,2$ mm nicht. Deswegen werden diese im KKS um $-0,2$ mm in x-Richtung korrigiert. Alternativ erhält man dasselbe Ergebnis, wenn man den Werkstücknullpunkt um 0.2 mm im MKS verschiebt. Die NPV G512 muss in beiden Aufspannungen im KKS um $-0,2$ mm in z-Richtung korrigieren. Die NPV G510 in der zweiten Aufspannung liegt in der Toleranz.

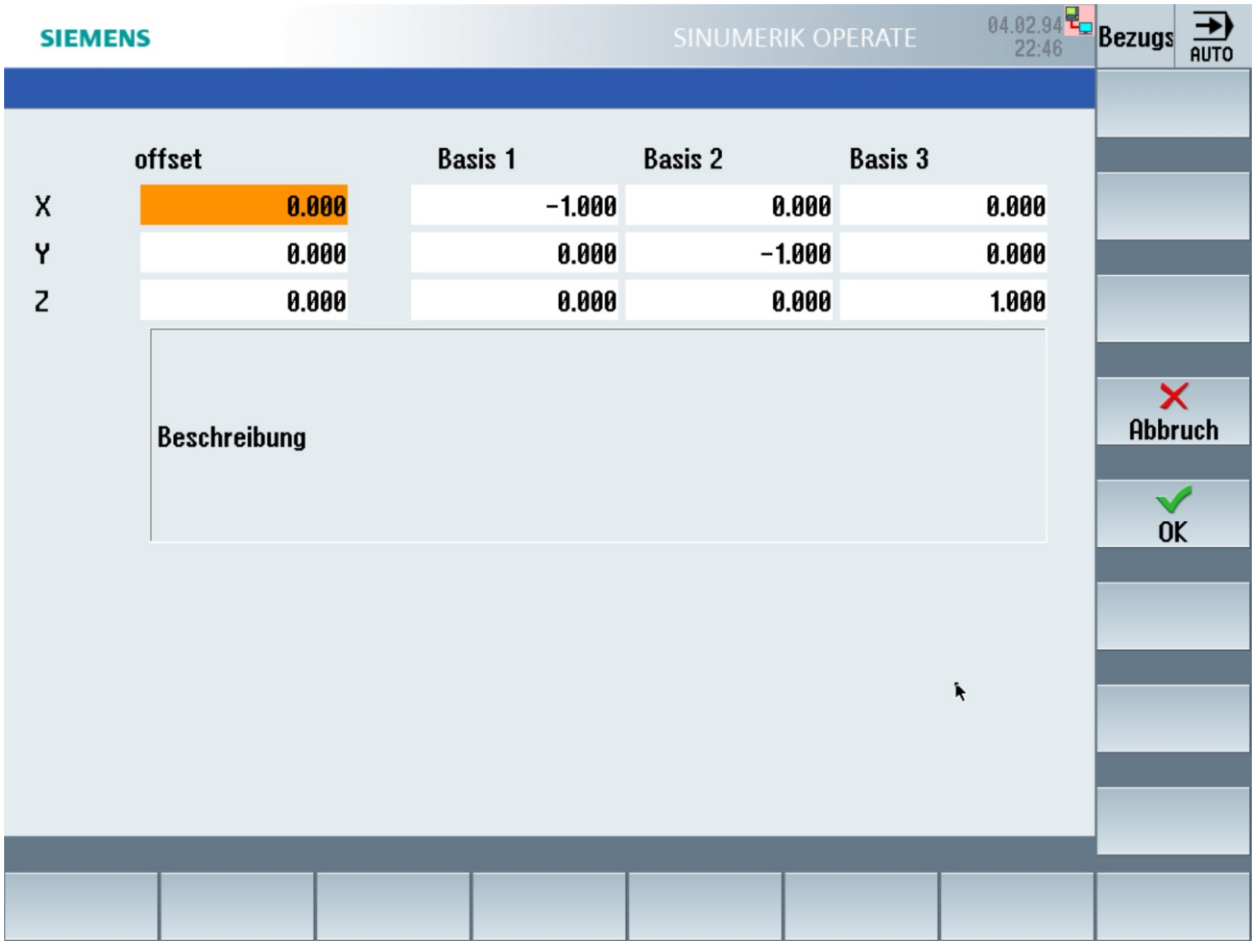


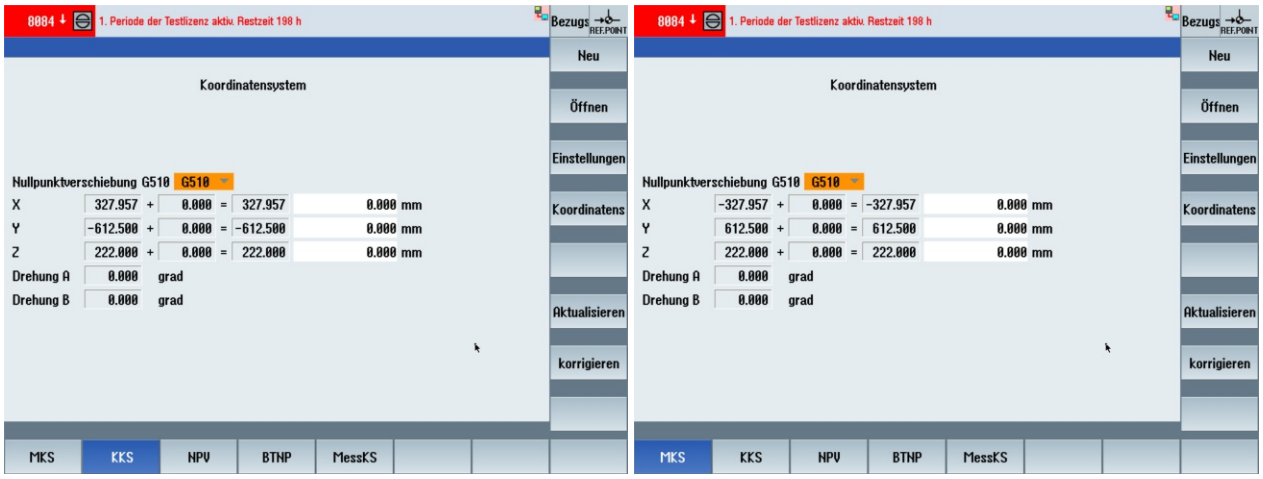
Abbildung 3.19.: Eingabebildschirm Transformationsmatrix

Die Tabelle 3.27 zeigt die NPV nach der Rotation um die A- und B-Achse im MKS, dabei hat der Maschinenbediener diese Werte schon korrigiert. Die Korrekturen wurden vom Maschinenbediener im Bearbeitungsprogramm vom CAM-System eingegeben. In den Listings 3.27 und 3.28 erkennt man die Korrekturen jeweils in den Befehlen `$P_UIFR[]`.

Die vom Maschinenbediener korrigierten Werte nach der Rotation im MKS:

erste Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-327,757 mm	612,5 mm	222,0 mm	0°	0°
G511	222,0 mm	612,5 mm	327.757 mm	0°	270°
G512	327,757 mm	221,8 mm	612,5 mm	−90°	180°
G513	-222,0 mm	612,5 mm	-327,757 mm	0°	90°
zweite Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-287,5 mm	-40,5 mm	60,0 mm	0°	180°
G512	287,5 mm	59,8 mm	-40,5 mm	−90°	0°

Tabelle 3.27.: NPVs korrigiert nach der Rotation im MKS.



(a) NPV G510 im KKS (b) NPV G510 im MKS

Abbildung 3.20.: Bildschirm für Korrekturen mit den Beispielwerten

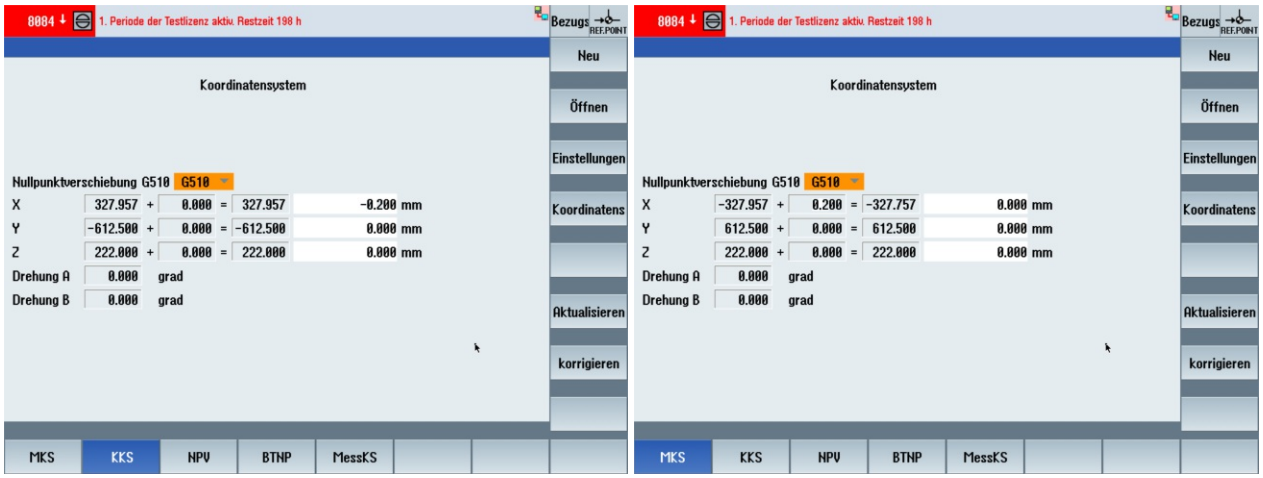
In Abb. 3.20 wird der Hauptbildschirm mit den Werten des G510 vom Bauteil gezeigt. Im Bild (a) sind die Werte im KKS dargestellt und im Bild (b) sind die Werte in das MKS zurück gerechnet. Da das Koordinatensystem um 180° um die z-Achse zueinander gedreht ist, ändert sich nur das Vorzeichen vom x- und y-Wert.

Die NPVs von mir nach dem ersten Einlesen in die Anwendung:

erste Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-327,957 mm	612,5 mm	222,0 mm	0°	0°
G511	222,0 mm	612,5 mm	327.957 mm	0°	270°
G512	327,957 mm	222,0 mm	612,5 mm	−90°	180°
G513	-222,0 mm	612,5 mm	-327,957 mm	0°	90°
zweite Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-287,5 mm	-40,5 mm	60,0 mm	0°	180°
G512	287,5 mm	60,0 mm	-40,5 mm	−90°	0°

Tabelle 3.28.: NPVs eingelesen in die Anwendung.

Dabei zeigt die Tabelle 3.28 die NPV nach der Rotation um die A- und B-Achse im MKS. Diese Rotation der NPV führte die von mir programmierte Anwendung durch. Die Ergebnisse von Tabelle 3.28 sind die gleichen, wie die Ergebnisse vom TRAF-Befehl in der Tabelle 3.26.



(a) Korrektur im KKS (b) Korrektur im MKS

Abbildung 3.21.: Eingabe der Korrekturen

In Abb. 3.21 zeigt Bild (a) die Werte im ausgewählten KKS. In den Feldern mit weißem Hintergrund ist es möglich, Werte einzugeben. Hier gebe ich einen Korrekturwert von $-0,200$ in der x-Achse ein. Im Bild (b) ist das MKS aktiviert. Beim Umrechnen vom KKS in das MKS ändert der Korrekturwert das Vorzeichen $(0,200)$. Der Hauptbildschirm zeigt nun die

Korrektur in der zweiten Spalte an, der Wert ist nun grau hinterlegt und kann in diesem Feld nicht geändert werden, weitere Änderungen trägt man in die weiß hinterlegten Felder ein. In Bild (b) zeigt der Hauptbildschirm für die x-Achse den ursprünglichen Wert $(-327,957)$, den Korrekturwert $(0,200)$ und das Ergebnis, den korrigierten Wert, $(-327,757)$ an.

Die NPVs nach dem Korrigieren in der Anwendung:

erste Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-327,757 mm	612,5 mm	222,0 mm	0°	0°
G511	222,0 mm	612,5 mm	327.757 mm	0°	270°
G512	327,757 mm	221,8 mm	612,5 mm	−90°	180°
G513	-222,0 mm	612,5 mm	-327,757 mm	0°	90°
zweite Aufspannung					
NPV	X	Y	Z	A	B
G54	0,0 mm	0,0 mm	0,0 mm	0°	0°
G510	-287,5 mm	-40,5 mm	60,0 mm	0°	180°
G512	287,5 mm	59,8 mm	-40,5 mm	−90°	0°

Tabelle 3.29.: NPVs eingelesen und korrigiert in der Anwendung.

Die Tabelle 3.29 zeigt die Werte, welche ich mit der Anwendung korrigiert habe und die der Steuerung zum Abarbeiten des NC-Programms übergeben werden. Die Anwendung liefert die erwarteten Ergebnisse. Die Ergebnisse in Tabelle 3.27 sind gleich wie die Ergebnisse in Tabelle 3.29, welche der Maschinenbediener auf die herkömmliche Art korrigiert hat.

4. Schluss

Das von mir verfasste Programm ermöglicht die NPV in einem beliebigen Koordinatensystem anzuzeigen. Zusätzlich können durch die Anwendung Korrekturen in das Maschinenkoordinatensystem (MKS) zurückgerechnet werden. Dafür kann der_die Anwender_in bis zu fünf unterschiedliche Bezugssysteme anlegen, die er später auch ändern kann. Durch ein Verzeichnis lassen sich neue oder bereits vorhandene Bauteile einlesen. Dabei muss das zu lesende NPV-Programm ein spezielles Format haben. Die einzulesende Datei darf maximal 20 Zeichen lang sein. Außerdem muss der TRAF-Befehl in folgender Form vorliegen:
„$TRAF(10, 1, 76.206, 258.187, -66.366, -52.074, 143.24, 0, 0, 0, 1, 2, 1)$“.
Der Werkstücknullpunkt muss wie folgt definiert werden:
„$\$P\_UIFR[1] = CTRANS(X, 0 + R201, Y, 0 + R202, Z, 0 + R203, Z2, 0 + R203)$“.
Falsche Eingaben werden nicht überprüft und würden zu einem Absturz der Steuerung führen. In den Einstellungen können die TRAF-Berechnungsoptionen eingesehen und geändert werden.

Das angewendete Programm zeigt eine Möglichkeit zur NC-Programmierung an der Werkzeugmaschine in einem beliebigen Koordinatensystem. Jedoch bleibt die Frage offen, wie der_die Anwender_in die Transformationsmatrix ermitteln kann.

Dies ist auch zugleich die größte Schwäche der Anwendung. Die Genauigkeit der Anwendung liegt bei der Transformationsmatrix. Deswegen müssen die Ergebnisse immer kritisch hinterfragt werden. Außerdem kann die Anwendung nur NPVs programmieren. Das ist nur effektiv, wenn jedes Merkmal (z.B. Bohrung, Gewinde oder Tasche) im Zentrum eine NPV hat. Bei Mustern von Merkmalen (z.B. Bohrungen für einen Flansch) muss eine Änderung im NC-Programm an den Merkmalen stattfinden. Dafür ist die Anwendung jedoch nicht gemacht. Außerdem ist die Anwendung mit der Annahme, dass ein Schwenktisch vorliegt, erstellt worden. Das Verhalten der Anwendung bei einem Schwenkkopf ist unbekannt.

Das NC-Programmieren an der Steuerung gestalten die Hersteller immer einfacher z.B. durch

Bohrzyklen. Jedoch erschwert sich durch immer kompliziertere Baugruppen die Korrektur der einzelnen Bauteile, speziell wenn das Bauteil die Toleranzen nicht einhält. Darum benötigen die Betreiber von Werkzeugmaschinen ein Tool, das NC-Programme unabhängig vom Bezugssystem interpretieren kann. Diese Anwendung macht die ersten Schritte dorthin.

Im Weiteren muss eine Anwendung geschaffen werden, die nicht nur NPVs sondern auch Merkmale programmieren kann. Möglich ist auch eine Anwendung, die das NC-Programm von einem Bezugssystem in ein anderes übersetzt. Zudem könnte die Messmaschine ein Protokoll mit Messdaten rückführen, damit könnte die Anwendung die Abweichungen automatisch korrigieren. Dafür benötigt die Anwendung ein standardisiertes Protokoll. Schon jetzt erstellen Messmaschinen Protokolle mit einem Soll-Ist Vergleich. Statt den Soll-Werten aus der CAD-Zeichnung würde der Vergleich mit den Soll-Werten aus dem NC-Programm stattfinden.

Für eine Verbesserung der Anwendung müsste der Fokus mehr auf dem_der Anwender_in liegen. Einzelne Tools zum Korrigieren gehören aufgegriffen und in einer Anwendung vereint. Außerdem muss der Aspekt der Messtechnik mehr in die Anwendung eingebunden werden. Zur Unterstützung könnte auch noch eine Abbildung des Werkstücks helfen.

Literatur

[1] *Werkzeugmaschinen für die Metallbearbeitung*, Norm, DIN 69651, Okt. 1974.

[2] A. Hellmich, A. Hirsch, M. Michael, J. Regel, M. Richter und V. Wittstock, *Werkzeugmaschinen*. 2012.

[3] W. Bahmann, *Werkzeugmaschinen kompakt*. 2013.

[4] A. Böge, *Handbuch Maschinenbau: Grundlagen und Anwendungen der Maschinenbau-Technik - Auflage 21.* 2013.

[5] H.-K. Tönshoff, *Werkzeugmaschinen Grundlagen*. Springer Verlag, 1995.

[6] A. Hirsch, *Werkzeugmaschinen: Anforderungen, Auslegung, Ausführungsbeispiele*. 2016.

[7] P. Hehenberger, *Computerunterstützte Fertigung*. 2011.

[8] S. Vajna, C. Weber, H. Bley, K. Zeman und P. Hehenberger, *CAx für Ingenieure – Eine praxisbezogene Einführung - Auflage 2*. Springer Verlag, 2009.

[9] *CAD-Benutzungsfunktionen*, Norm, VDI 2249, Sep. 2003.

[10] *Koordinatenachsen und Bewegungsrichtungen für numerisch gesteuer- te Arbeitsmaschinen*, Norm, DIN 66217, Dez. 1975.

[11] A. Stadler und M. Tholen, *Das C++ Tutorial*. 2018.

[12] M. Battisti, *C ++ im Unterricht*. 2008.

[13] V. Plenk, *Angewandte Netzwerktechnik kompakt*. 2017.

[14] S. Ziya, *Mathematik für Ingenieure*. 2020.

[15] Quartl, Sanitiy, PhilologistLaGr und S. Birkner. „Inverse Matrix.“ (8. Feb. 2021), Adresse: `https://de.wikipedia.org/wiki/Inverse_Matrix` (besucht am 26. 04. 2021).

[16] S. Birkner, LutzL, Zemke, Digamma und 87.145.76.154. „Adjunkte.“ (11. Dez. 2008), Adresse: `https://de.wikipedia.org/wiki/Adjunkte` (besucht am 26. 04. 2021).

[17] Sinumerik Dokumentation, *Programmierhandbuch SINUMERIK Integrate Crete My-HMI/3GL (Qt Api)*. 2019.

[18] Ivanovic. „Minimal Example on how to read, write and print QJson code (with QJsonDocument, QJsonArray, QJsonObject, QFile).“ (18. Apr. 2020), Adresse: https://stackoverflow.com/questions/61075951/minimal-example-on-how-to-read-write-and-print-qjson-code-with-qjsondocument (besucht am 26. 04. 2021).

[19] Rrozek. „QtJsonModifyValue.“ (13. Jan. 2018), Adresse: https://github.com/rrozek/QtJsonModifyValue/blob/653fc8f0c10f976bd7cf9cb8a825c08c950524c8/main.cpp (besucht am 26. 04. 2021).

[20] C. Brecher und M. Weck, *Werkzeugmaschinen Fertigungssysteme 2 - Konstruktion, Berechnung und messtechnische Beurteilung*. 2017.

A. Json-Datei


```
{
    "10":
  {
    "alpha": 0,
    "beta": 0,
    "npv": 10,
    "x": [-327.957, 0.2],
    "y": [612.5, 0],
    "z": [222, 0]
  },
    "11":
  {
    "alpha": 0,
    "beta": 270,
    "npv": 11,
    "x": [-327.957, 0.2],
    "y": [612.5, 0],
    "z": [222, -7.347880794884119e-17]
  },
    "12":
  {
    "alpha": -90,
    "beta": 180,
    "npv": 12,
    "x": [-327.957, 0.2],
    "y": [612.5, 0],
    "z": [222, -0.2]
  },
    "13":
  {
    "alpha": 0,
    "beta": 90,
    "npv": 13,
    "x": [-327.957, 0.2],
```

```
    "y": [612.5, 0],
    "z": [222, 0]
  },
  "setup":
  {
    "bea": 1,
    "W": 0,
    "meaf": false,
    "spindel": 1
  }
}
```

Listing A.1: JSON-Datei für die erste Aufspannung

```
{
    "10":
  {
        "alpha": 0,
        "beta": 180,
        "npv": 10,
        "x": [287.5, 0],
        "y": [-40.5, 0],
        "z": [-60, 0]
    },
    "12":
  {
        "alpha": -90,
        "beta": 0,
        "npv": 12,
        "x": [287.5, 0],
        "y": [-40.5, 0],
        "z": [-60, 0.2]
    },
    "setup":
  {
        "bea": 1,
        "W": 0,
        "meaf": false,
        "spindel": 1
    }
}
```

Listing A.2: JSON-Datei für die zweite Aufspannung

B. Koordinatensysteme

```
{
    "CCS":
  {
        "input":
    [
            [-1, 0, 0, 0],
            [0, -1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
        ],
        "invers":
    [
            [-1, 0, 0, 0],
            [0, -1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
        ],
        "trans":
    [
            [-1, 0, 0, 0],
            [0, -1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
        ]
    },
    "MCS":
  {
        "input":
    [
            [1, 0, 0, 0],
            [0, 1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
```

```
34          ],
35          "invers":
36      [
37                  [1, 0, 0, 0],
38                  [0, 1, 0, 0],
39                  [0, 0, 1, 0],
40                  [0, 0, 0, 1]
41          ],
42          "trans":
43      [
44                  [1, 0, 0, 0],
45                  [0, 1, 0, 0],
46                  [0, 0, 1, 0],
47                  [0, 0, 0, 1]
48          ]
49      }
50  }
```

Listing B.1: JSON-Datei mit den Koordinatensystemen

C. modifyJsonValue

```
void fileManager::modifyJsonValue(QJsonValue& destValue, const QString&
    path, const QJsonValue& newValue)
{
  const int indexOfDot = path.indexOf('.');
  const QString dotPropertyName = path.left(indexOfDot);
  const QString dotSubPath = indexOfDot > 0 ? path.mid(indexOfDot + 1) :
    QString();

  const int indexOfSquareBracketOpen = path.indexOf('[');
  const int indexOfSquareBracketClose = path.indexOf(']');

  const int arrayIndex = path.mid(indexOfSquareBracketOpen + 1,
    indexOfSquareBracketClose - indexOfSquareBracketOpen - 1).toInt();

  const QString squareBracketPropertyName = path.left(
    indexOfSquareBracketOpen);
  const QString squareBracketSubPath = indexOfSquareBracketClose > 0 ? (
    path.mid(indexOfSquareBracketClose + 1)[0] == '.' ? path.mid(
    indexOfSquareBracketClose + 2) : path.mid(indexOfSquareBracketClose +
    1)) : QString();

  // determine what is first in path. dot or bracket
  bool useDot = true;
  if (indexOfDot >= 0) // there is a dot in path
  {
    if (indexOfSquareBracketOpen >= 0) // there is squarebracket in path
    {
      if (indexOfDot > indexOfSquareBracketOpen)
        useDot = false;
      else
        useDot = true;
    }
    else
```

```
27      useDot = true;
28    }
29    else
30    {
31      if (indexOfSquareBracketOpen >= 0)
32        useDot = false;
33      else
34        useDot = true; // acutally, id doesn't matter, both dot and square
       bracket don't exist
35    }
36
37    QString usedPropertyName = useDot ? dotPropertyName :
       squareBracketPropertyName;
38    QString usedSubPath = useDot ? dotSubPath : squareBracketSubPath;
39
40    QJsonValue subValue;
41    if (destValue.isArray())
42      subValue = destValue.toArray()[arrayIndex];
43    else if (destValue.isObject())
44      subValue = destValue.toObject()[usedPropertyName];
45    else
46      qDebug() << "oh, what should i do now with the following value?! " <<
       destValue;
47
48    if (usedSubPath.isEmpty())
49    {
50      subValue = newValue;
51    }
52    else
53    {
54      if (subValue.isArray())
55      {
56        QJsonArray arr = subValue.toArray();
57        QJsonValue arrEntry = arr[arrayIndex];
58        modifyJsonValue(arrEntry, usedSubPath, newValue);
59        arr[arrayIndex] = arrEntry;
60        subValue = arr;
61      }
62      else if (subValue.isObject())
63        modifyJsonValue(subValue, usedSubPath, newValue);
64      else
65        subValue = newValue;
```

```
66    }
67    if (destValue.isArray())
68    {
69      QJsonArray arr = destValue.toArray();
70      if (subValue.isNull())
71        arr.removeAt(arrayIndex);
72      else
73        arr[arrayIndex] = subValue;
74      destValue = arr;
75    }
76    else if (destValue.isObject())
77    {
78      QJsonObject obj = destValue.toObject();
79      if (subValue.isNull())
80        obj.remove(usedPropertyName);
81      else
82        obj[usedPropertyName] = subValue;
83      destValue = obj;
84    }
85    else
86      destValue = newValue;
87  }
```

Listing C.1: Die Funktion modifyJsonValue() Teil1

```
1  void fileManager::modifyJsonValue(QJsonDocument& doc, const QString& path,
       const QJsonValue& newValue)
2  {
3    QJsonValue val;
4    if (doc.isArray())
5      val = doc.array();
6    else
7      val = doc.object();
8
9    modifyJsonValue(val, path, newValue);
10
11   if (val.isArray())
12     doc = QJsonDocument(val.toArray());
13   else
14     doc = QJsonDocument(val.toObject());
15 }
```

Listing C.2: Die Funktion modifyJsonValue() Teil2

D. NPV-Programm

```
;*****************************************************************
;* Unterprogramm zur Berechnung aller Verschiebungen
;* des gesamten Werkstueckes
;*
;******************* INFO ZU PROGRAMM ********************
; PROGRAMMIERER:  MBe_
; TEILE NAME:  Batteryhousing FF4 Layer 1
; SPANNUNGS NUMMER:  OP10
; KUNDE:
;*************************************************************
;*****************************************************

;Verschiebung Maschinennullpunkt --> Bauteilnullpunkt
;G54
$P_UIFR[1]=CTRANS(X,0,Y,0,Z,0):CROT(X,0,Y,0,Z,0)

;;; NULLPUNKTE UND VERRECHNUNGEN DER DREHUNGEN!!!!!!!!
;;;*****************************************************
;;;*****************************************************
;;; INFO
;Bearbeitung A=0 / B=0 [ G510 ]
;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
$P_UIFR[10]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
TRAF(10,0,-327.957,612.5,222,0,0,0,0,0,0,1,1)
;;;*****************************************************

;;;*****************************************************
;;; INFO
;Bearbeitung A=0 / B=270 [ G511 ]
;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
$P_UIFR[11]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
TRAF(11,0,-327.957,612.5,222,0,270,0,0,0,0,1,1)
;;;*****************************************************
```

```
34
35 ;;;*************************************************
36 ;;; INFO
37 ;Bearbeitung A=-90 / B=180 [ G512 ]
38 ;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
39 $P_UIFR[12]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
40 TRAF(12,0,-327.957,612.5,222,-90,180,0,0,0,0,1,1)
41 ;;;*************************************************
42
43 ;;;*************************************************
44 ;;; INFO
45 ;Bearbeitung A=0 / B=90 [ G513 ]
46 ;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
47 $P_UIFR[13]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
48 TRAF(13,0,-327.957,612.5,222,0,90,0,0,0,0,1,1)
49 ;;;*************************************************
50
51 RET
```

Listing D.1: NPV-Programm erste Aufspannung

```
1  ;*****************************************************************
2  ;* Unterprogramm zur Berechnung aller Verschiebungen
3  ;* des gesamten Werkstueckes
4  ;*
5  ;******************* INFO ZU PROGRAMM ********************
6  ; PROGRAMMIERER:  MBe_
7  ; TEILE NAME:  Batteryhousing FF4 Layer 1
8  ; SPANNUNGS NUMMER:  OP20
9  ; KUNDE:
10 ;*************************************************************
11 ;*************************************************
12
13 ;Verschiebung Maschinennullpunkt --> Bauteilnullpunkt
14 ;G54
15 $P_UIFR[1]=CTRANS(X,0,Y,0,Z,0):CROT(X,0,Y,0,Z,0)
16
17 ;;; NULLPUNKTE UND VERRECHNUNGEN DER DREHUNGEN!!!!!!!!
18 ;;;*************************************************
19 ;;;*************************************************
20 ;;; INFO
21 ;Bearbeitung A=0 / B=180 [ G510 ]
22 ;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
```

```
$P_UIFR[10]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
TRAF(10,0,287.5,-40.5,-60,0,180,0,0,0,0,1,1)
;;;***************************************************

;;;***************************************************
;;; INFO
;Bearbeitung A=-90 / B=0 [ G512 ]
;Verschiebung Bauteilnullpunkt --> Bearbeitungsnullpunkt"
$P_UIFR[12]=CFINE(X,0,Y,0,Z,0,A,0,B,0)
TRAF(12,0,287.5,-40.5,-60,-90,0,0,0,0,0,1,1)
;;;***************************************************

RET
```

Listing D.2: NPV-Programm zweite Aufspannung