Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Denis Sivkov

Entwicklung und Simulation der Steuerung eines dreiachsigen Positioniertisches für den Genauigkeitsnachweis von Laserscannern im dynamischen Betrieb

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Denis Sivkov

Entwicklung und Simulation der Steuerung eines dreiachsigen Positioniertisches für den Genauigkeitsnachweis von Laserscannern im dynamischen Betrieb

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Dipl.-Kfm. Jörg Dahlkemper
Zweitgutachter : Prof. Dr.-Ing. Alfred Ebberg

Abgegeben am

Denis Sivkov

Thema der Masterarbeit

Entwicklung und Simulation der Steuerung eines dreiachsigen Positioniertisches für den Genauigkeitsnachweis von Laserscannern im dynamischen Betrieb

Stichworte

Laserscanner, Messung der Flugzeit, Portalroboter, V-Modell, CoDeSys, SoftMotion, Steuerung, Trajektorie, Man-Maschine-Interface, Python, Blender, Keyframe, Laser-View

Kurzzusammenfassung

Die Masterarbeit umfasst eine Einführung in die existierenden Methoden für den Genauigkeitsnachweis von Laserscannern. Eine nach dem V-Modell aufgebaute Entwicklung einer Steuerung des Positioniertisches wird in der Arbeit auf einer Simulationsebene realisiert. Die Bewegung des Laserscanners über die vorgegebene Trajektorie und Simulation der Scandaten über den 3D-Simulator sind in der Arbeit detailliert beschrieben. Der mechanische Aufbau mit einer linearen Achse ermöglicht eine reale Aufnahme der Laserscans im dynamischen Betrieb sowie den weiteren Vergleich von Scandaten mit der Referenzposition

Denis Sivkov

Title of the master thesis

Development and Simulation of a Controller for a gantry robot with three degree of freedom for the purpose of accuracy of laser scanner while dynamic operation

Keywords

Laser scanner, measurement of time of flight, gantry robot, CoDeSys, SoftMotion, controller, trajectory, human-machine-interface, python, Blender, keyframes, LaserView

Abstract

The master thesis describes the existing methods for investigation of laser scanner accuracy. Development of a program for programmable logic unit is described. The programed PLC controls a gantry robot with three degree of freedom. The control of robot is realised as simulation. The motion of laser scanner over a designed trajectory and simulation with 3D-Simulator are in the thesis detailed described. Mechanical setup with a linear axis makes possible the recording of the laser scans while dynamic operation and the comparison the scan data with reference position

Inhaltsverzeichnis:

Abbildungsverzeichnis

1. Einleitung

Laserscanner finden Anwendung in unterschiedlichen Bereichen, wie z.B. in der Fertigungsindustrie, Logistik, Gebäudeüberwachung oder in autonomen Fahrzeugen [1]. Dabei ist es mit Laserscannern möglich, in einem fest definierten Sektor Tiefeninformationen zu messen und diesen Bereich folglich zu überwachen. Ein beispielhaftes Anwendungsszenario stellt die Überwachung eines Geländes mit Laserscannern, statt mit Kameras und Sicherheitspersonal, dar.

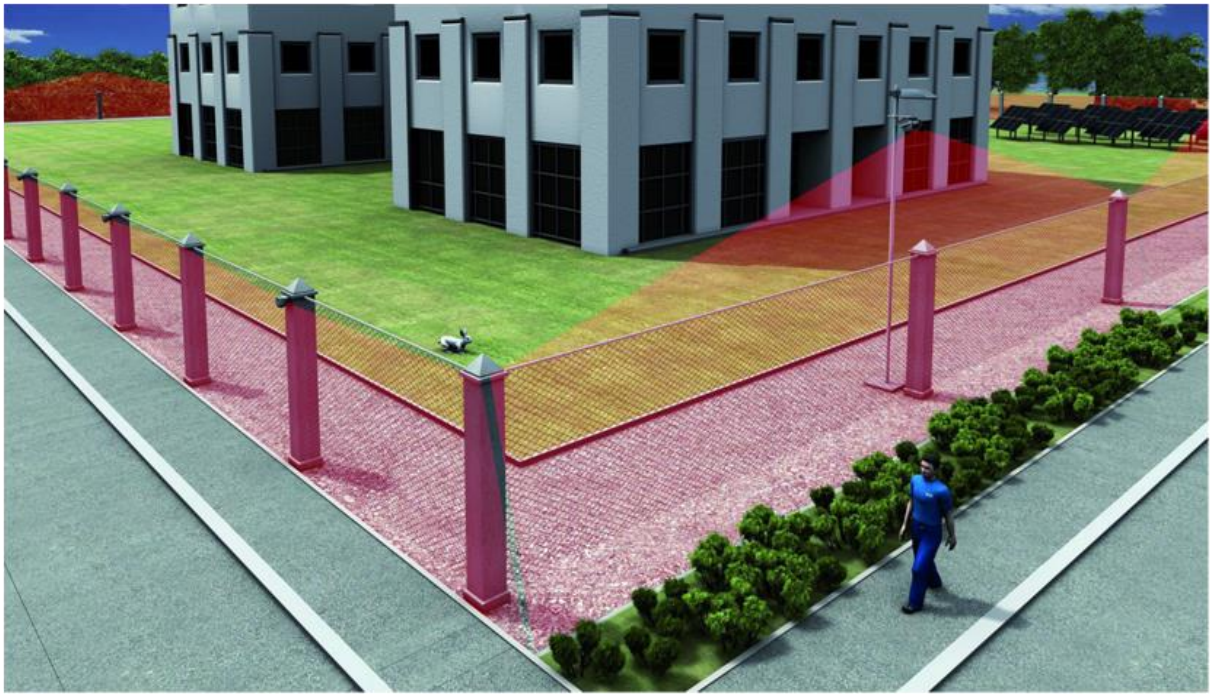


Abbildung 1-1 Laserscanner zur Freigeländeüberwachung

Aufgrund der hohen Scanfrequenz (100 Hz und mehr) können Laserscanner ebenso an Fahrzeugen eingesetzt werden. Dadurch lassen sich Objekte im Umkreis verfolgen und so prognostizieren, ob es beim aktuellen Fahrtpfad zu einer Kollision kommen kann. In der Logistik werden sogenannte AGVs (Automatic Guided Vehicle), wie z.B. ein autonomer Gabelstapler, bereits eingesetzt. Diese können sich mithilfe von Laserscannern in einer Lagerhalle lokalisieren und sich dadurch autonom bewegen.

Ein Hersteller von Laserscannern ist die Sick AG, die marktführend im Bereich der optischen Sensorik ist und eine reiche Produktvielfalt an Lichtschranken und Laserscannern anbietet [2].

Der Sick Laserscanner NAV350 (Abbildung 1-2) gelangt zurzeit in der Navigation zum Einsatz. Dieser Laserscanner erreicht laut Spezifikation im stationären Betrieb eine Genauigkeit kleiner als 1cm. Eine verwertbare Aussage über die Genauigkeit im dynamischen Betrieb existiert bislang nicht. Da während der Bewegung des Fahrzeuges Kräfte auf den Laserscanner wirken, können die Kräfte eine Abweichung der spezifizierten Genauigkeit auslösen.

Abbildung 1-2 Sick Laserscanner NAV350

Die Messdaten der aktuellen Algorithmen zur Lokalisierung weichen momentan von der bestehenden Referenz ab. Die Referenz ist allerdings nicht absolut. Abbildung 1-3 stellt ein Diagramm einer gewöhnlichen AGV-Kurve dar. Es ist auf der Abbildung 1-3 ebenso zu ersehen, dass die Abweichung außerhalb der vorgegebenen Genauigkeit (<1cm) liegt.

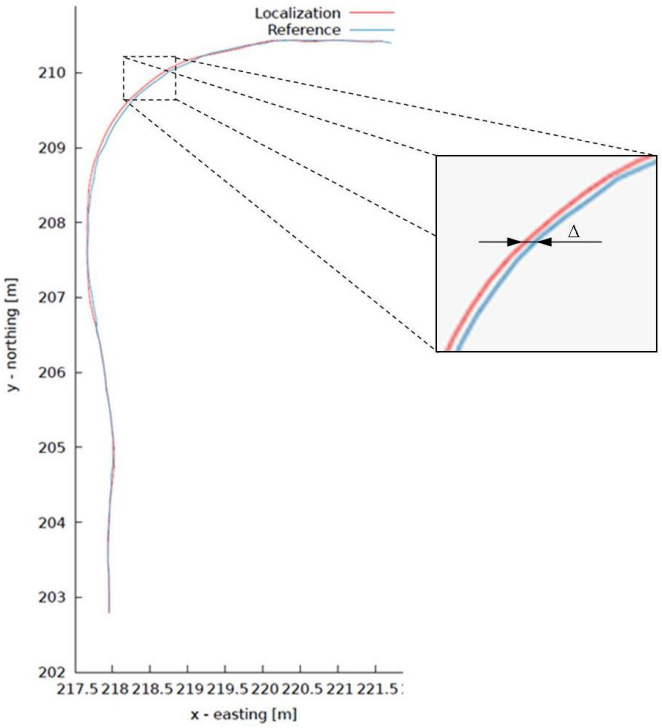


Abbildung 1-3 Vergleich Lokalisierung – Referenz

Aufgabenstellung

Die Masterarbeit soll eine Lücke schließen: Es bestehen zwar viele wissenschaftliche Arbeiten, die den Nachweis der Genauigkeit von Laserscanner im statischen Betrieb beschreiben – bislang gibt es allerdings keine Arbeit, die den Nachweis im dynamischen Betrieb des Laserscanners möglich macht.

Eine der ersten Aufgaben der Arbeit ist die Entwicklung eines Konzeptes einer Anlage, die gestellte Forderungen erfüllt. Des Weiteren sollen die Planung und Definition der notwendigen Trajektorien erfolgen, um die Messgenauigkeit des Laserscanners im dynamischen Betrieb nachweisen zu können.

Wenn die Systemanforderungen sowie das Architekturdesign des Positioniertisches definiert sind, soll die Programmierung der Steuerung realisiert werden. Eine Man-Maschine-Schnittstelle wird ein essenzieller Teil der Programmierung, da das Handhaben der Anlage dem zukünftigen Bediener möglichst übersichtlich und einfach gemacht werden soll.

Als Nächstes sollen die vordefinierten Trajektorien mithilfe der entwickelten Steuerung verfahren werden. Die IST-Position soll mit einer vorgegebenen Frequenz abgetastet werden. Die Modellierung des Positioniertischs und der Testhalle, wo die Anlage stehen wird, soll mittels Blende erfolgen. Weiterhin werden die gespeicherten Abtastpunkte der Trajektorie als Keyframes in Blender-Model hinzugefügt. Somit entsteht eine Szenerie mit einer entsprechenden Bewegung der Achsen.

Sobald eine Szenerie erstellt ist, können mittels eines 3D-Simulators die Scanpunkte simuliert werden. Weiterhin werden die simulierten Scandaten ausgewertet und eine Aussage über die Genauigkeit des Laserscanner im dynamischen Betreib getätigt.

2. Stand der Technik

Um differenziert darzustellen, welche Arbeiten bzw. technische Umsetzungen im Bereich der Forschung und Entwicklung bislang durchgeführt wurden, werden im Kapitel 2 die wissenschaftlichen Arbeiten vorgestellt, die für den Genauigkeitsnachweis bzw. Klassifizierung von Laserscannern existieren.

2.1. Charakterisierung von 2-D Laserscannern zum Ausweichen der Hindernisse

Mobile Roboter werden mit verschiedenen Laserscannern ausgerüstet, um Hindernissen auszuweichen. Die Charakterisierung des eingesetzten Sensors spielt eine entscheidende Rolle bei der Auswahl des Geräts für eine bestimmte Applikation. Die Arbeit von Cang Ye und Johann Borenstein (University of Michigan)[4] beschreibt die Untersuchung von jenen Parametern des Laserscanners LMS 200 (siehe Abbildung 2-1), die zur Erstellung der 3-D Karten relevant sind.

Abbildung 2-1 Laserscanner LMS 200

Der Versuchsaufbau besteht aus einer computergesteuerten Linearachse zur genauen Positionierung des Hindernisses im bestimmten Abstand vom Laserscanner. Der Laserscanner bleibt während des Versuchs in einer konstanten Position und Orientierung. Insbesondere die Orientierung des Laserscanners soll genau mit der Bewegungsachse ausgerichtet sein. Der Aufbau ist in Abbildung 2-2 detailliert dargestellt.

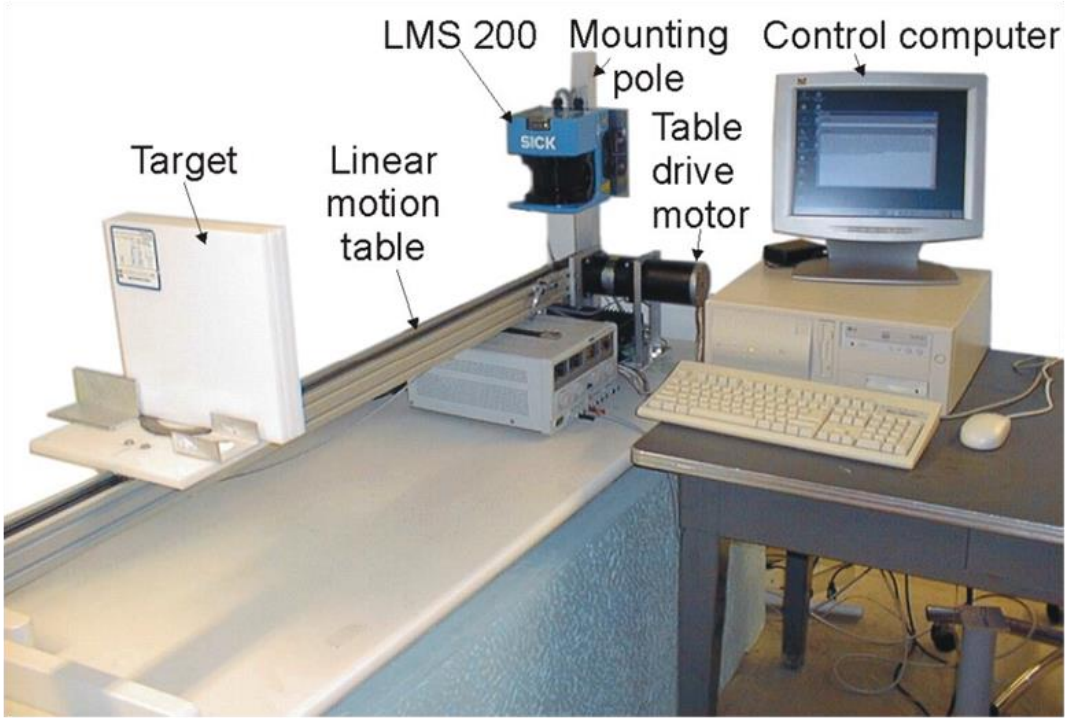


Abbildung 2-2 Versuchsaufbau

Der Öffnungswinkel vom LMS 200 beträgt 180 Grad (Messbereich 0-180°) und die Auflösung 1°. Die Mitte des Öffnungswinkels (91. Strahl) ist parallel zur Linearachse. Bevor man mit dem Versuch anfangen kann, ist die Kalibrierung des Versuchsausbaus notwendig. Zur Kalibrierung wurden folgende Prozeduren durchgeführt:

1. das Hindernis im Abstand von 330 mm vom Laserscanner positionieren und den Messwert d_{m1} ablesen,
2. das Hindernis aus den Abstand von 3800 mm bewegen und den Messwert d_{m2} ablesen,
3. wenn die Differenz $d_{m2} - d_{m1}$ größer als 3800 mm ausfällt, den Kippwinkel des Laserscanners ändern und die Schritte 1. und 2. wiederholen, bis die Differenz den Minimumwert erreicht,
4. den Strahl identifizieren, an dem die Differenz $d_{m2} - d_{m1}$ minimal ist. Der Strahl, der die minimale Differenz liefert, ist am nächsten zur Bewegungslinie der Achse

In dieser Arbeit werden die Einflüsse von Parametern des Laserscanners auf die Messergebnisse dargestellt und verglichen.

Die untersuchten Parameter des Laserscanners LMS 200 sind

- Datenübertragungsrate
- Drift
- Optische Eigenschaften
- Einfallswinkel des Laserstrahls

Einflüsse der Datenübertragungsrate

Eine schnelle Datenübertragungsrate ist für den schnellen Roboter äußerst wichtig. Es ist vor allem wichtig, dass keine Daten während des Betriebs verloren gehen. Die Untersuchungen von Cang Ye und Johann Borenstein haben erwiesen, dass der LMS 200 auch bei der schnellsten Übertragungsrate von 500Kbaud fehlerfrei funktioniert.

Einflüsse des Driftes

Der Abstand vom Hindernis zum Laserscanner wurde 600.000 Mal gemessen. Die Entfernung blieb während des Versuches konstant auf 2 Meter. Der gemessene Abstand ist in der Abbildung 2-3 dargestellt.

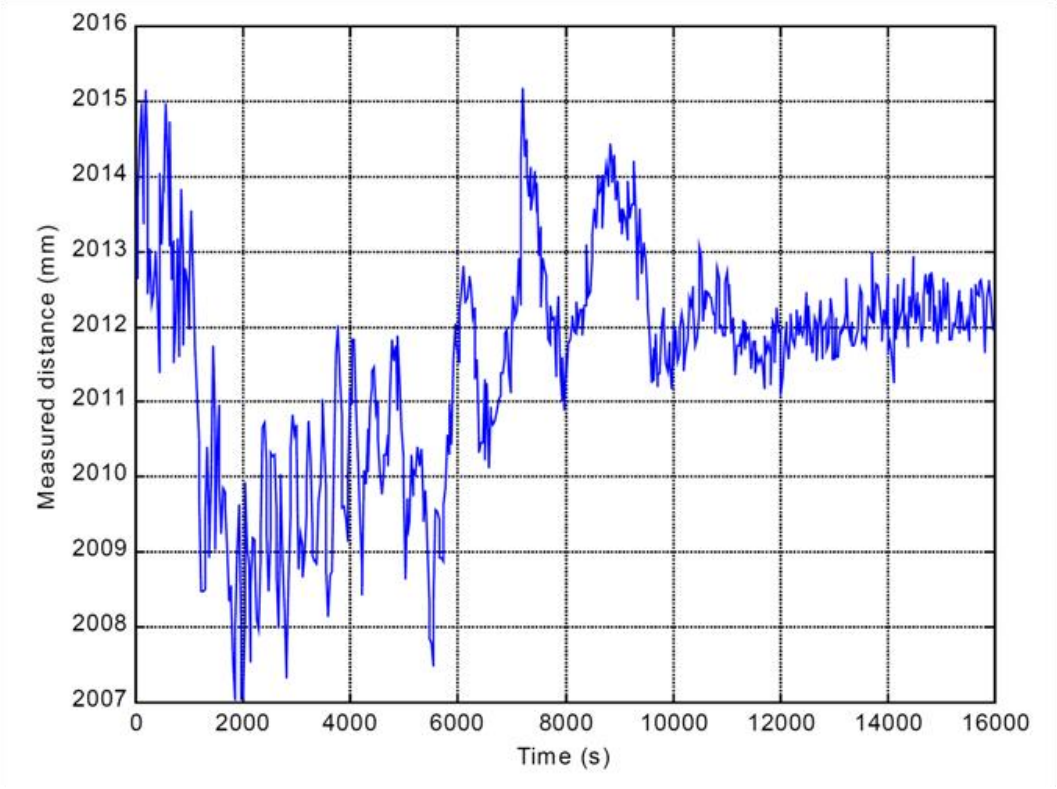


Abbildung 2-3 Der gemessene Abstand während der Versuchsdauer

Man kann eine Aufwärmphase von ca. 10.000 Sekunden identifizieren. Die Messergebnisse schwingen während der Phase von 7 mm bis 15 mm. Der Wert des gemessenen Abstandes stabilisiert sich nach dem Aufwärmen und beläuft sich auf 12 mm.

Einflüsse der Hindernisoberfläche

Um die Charakterisierung des Laserscanner LMS 200 vollständig zu realisieren, wurden im Versuch 13 Hindernisse mit verschiedenen Oberflächen analysiert:

Glänzende Farben:	goldfarbene, silberfarbene Oberflächen
	Oberfläche aus poliertem Aluminium
Matte Farben:	Blau, Purpur, Rot, Grün, Gelb
Graustufen:	RBG: 255, 191, 127, 63, 0

Das Hindernis wurde vor dem Beginn des Versuches in einer Entfernung von 192 mm vom Laserscanner positioniert und nach dem Start in 500 mm Schritten auf die Entfernung von 3692 mm bewegt. Der Abstand zum Hindernis wurde in jeder Position 10.000 gemessen. Die Ergebnisse sind auf Abbildung 2-4, Abbildung 2-5 und Abbildung 2-6 ersichtlich.

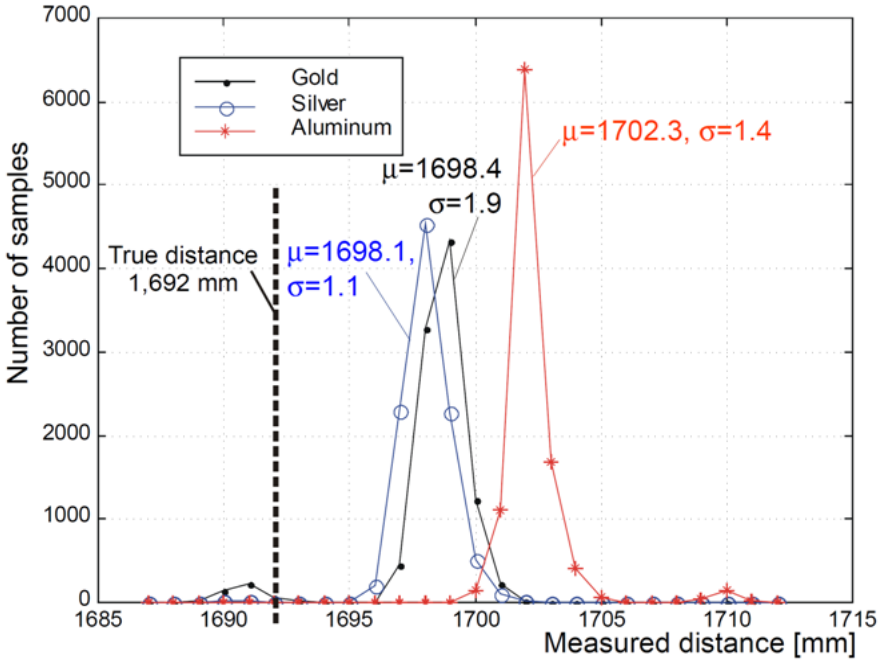


Abbildung 2-4 Messergebnisse mit der glänzenden Oberflächen

In Abbildung 2-4 sind die Ergebnisse der Messungen mit den glänzenden Oberflächen dargestellt. Die Messwerte der drei Farben weisen eine Normalverteilung auf. Die am größten reflektierende Oberfläche verfügt entsprechend über die höchste Spitze. Dies ist eine Folge der Lichtreflektion: Je glänzender die Oberfläche ist, desto geringer ist die Intensität des reflektierten Lichtes.

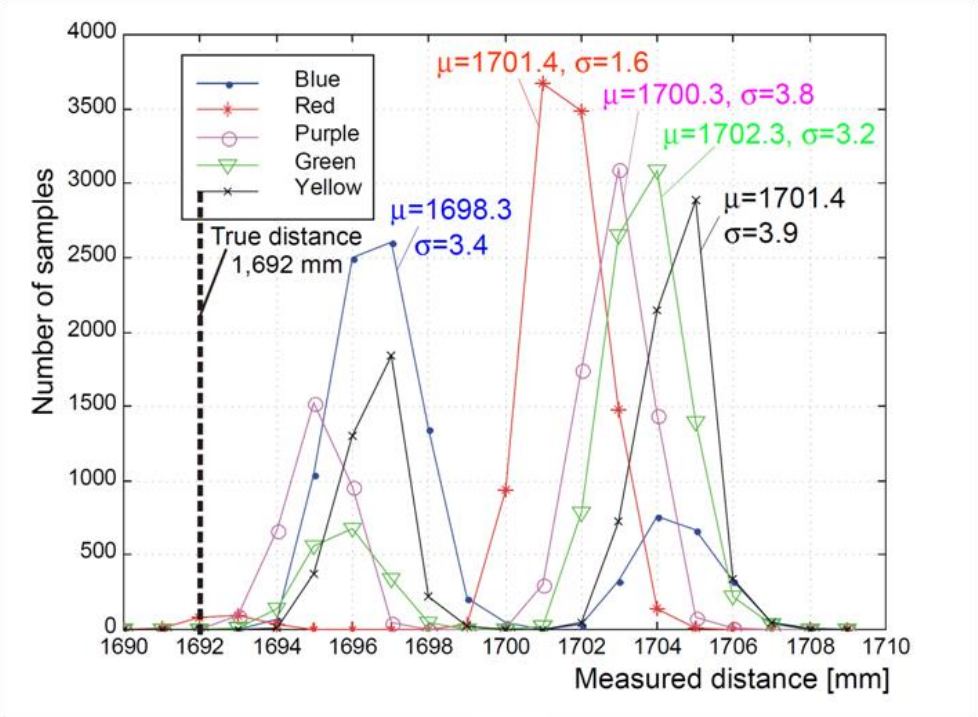


Abbildung 2-5 Messergebnisse mit den matten Oberflächen

Die Kurven der Ergebnisse mit den matten Oberflächen haben einen spezifischen Verlauf und haben die Form von zwei Wellen. Die Wellen sind normal verteilt. Die Diagramme sind vergleichbar mit den Ergebnissen der glänzenden Oberflächen. Das heißt, dass die Farbe der Oberfläche keine signifikante Auswirkung auf die Messergebnisse aufweist.

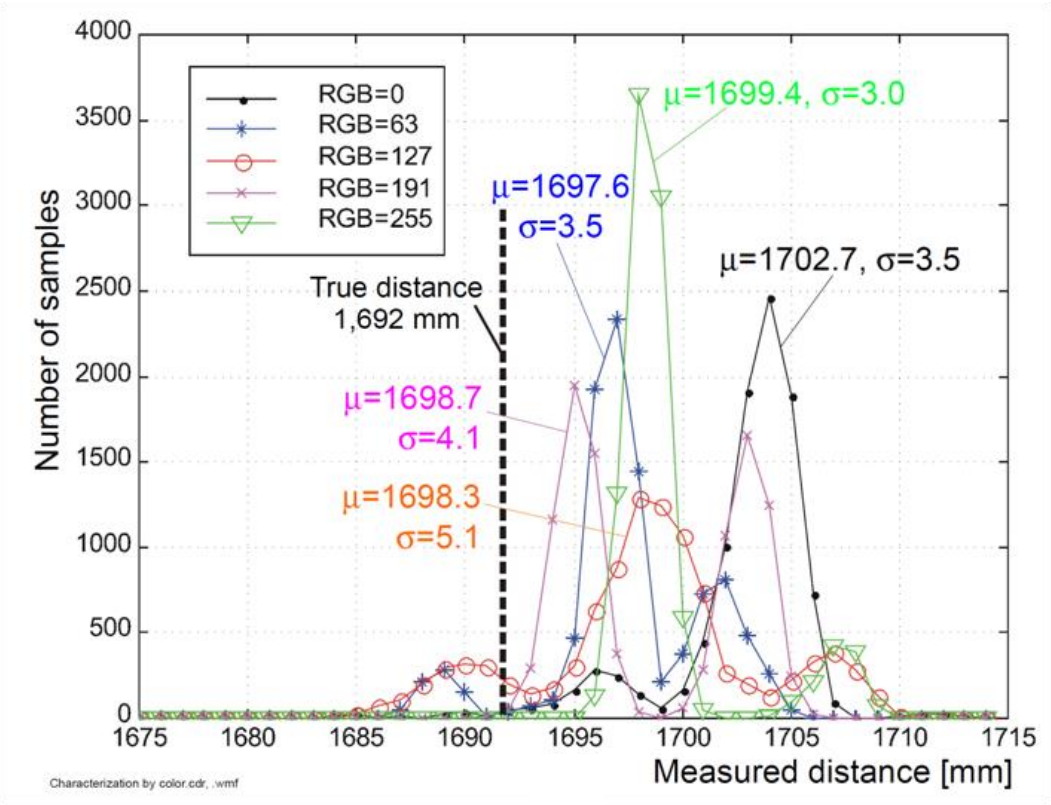


Abbildung 2-6 Messergebnisse mit den grauen Oberflächen

Die Messwerte der Ergebnisse mit den Graustufen bilden die normal verteilen Kurven mit einer hohen Zentralspitze sowie zwei kleinen Seitenwellen. Die Werte sind allgemein kleiner als die Werte der Messungen mit den glänzenden und den matten Oberflächen. Das bedeutet, dass die Graustufen im Vergleich zu anderen Messungen eine signifikante Auswirkung auf die Verteilung der Messergebnisse aufweisen.

Einflüsse des Einfallswinkels

Das Hindernis mit RGB=127 Oberfläche wurde mit einem Abstand von 2 m zum Laserscanner positioniert und als Nächstes wurden die Messungen mit 13 verschiedenen Winkeln durchgeführt. Die eingestellten Winkel sind 0°, ±10°, ±20°, ±30°, ±40°, ±50° und ±60°.

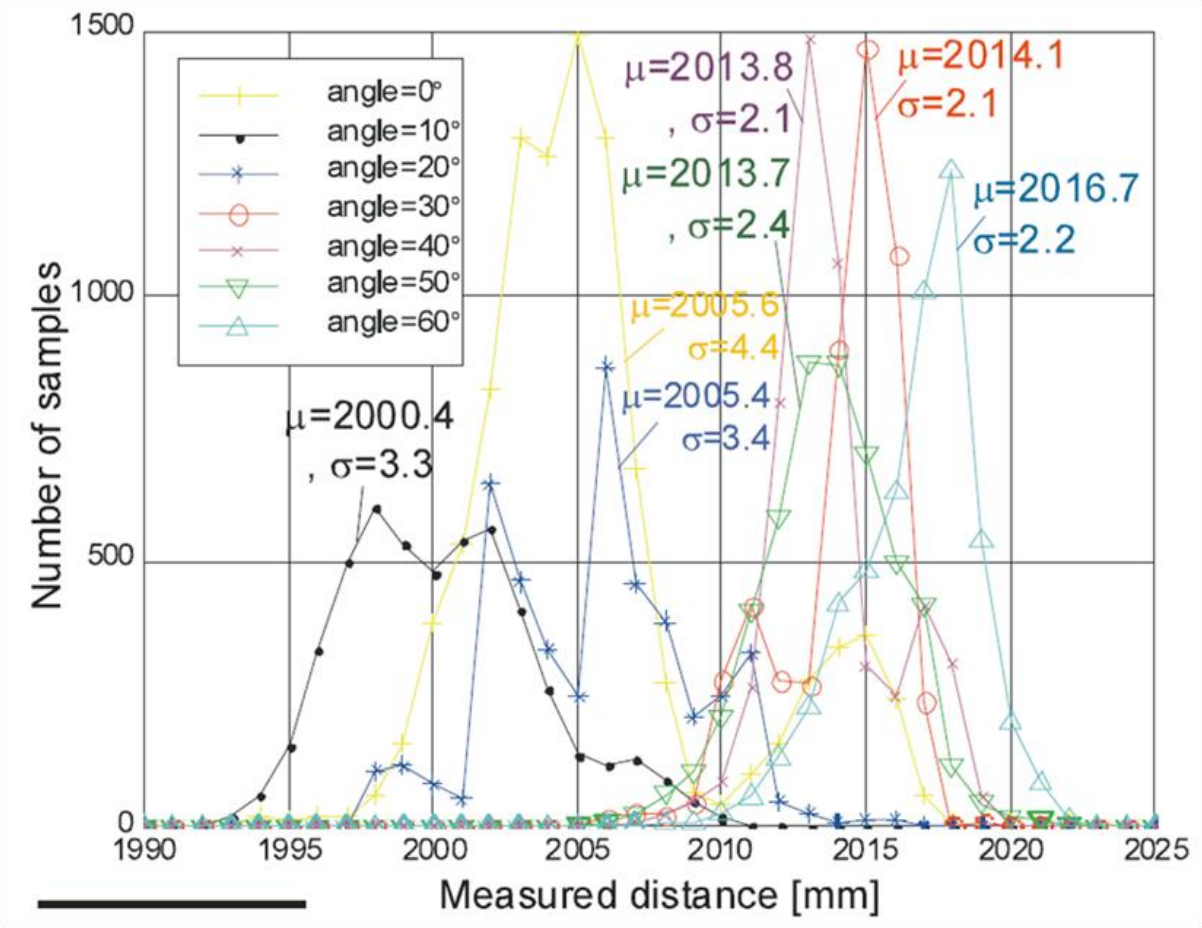


Abbildung 2-7 Messergebnisse mit verschiedenen Einfallswinkeln des Laserstrahls

In Abbildung 2-7 zu erkennen, dass die Kurven für 0°, ±10° und ±20° voneinander nicht weit entfernt sind. Der größte Fehler findet sich bei einer Orientierung von ±30 und er wird kleiner, wenn die Orientierung größer wird. Der Unterschied zwischen dem kleinsten und dem größten mittleren Fehler beträgt etwa 16 mm.

Kalibrierung von Laserscannern

Nachdem die empirischen Daten ermittelt wurden, konnte ein mathematisches Modell zur Kalibrierung sowie zur Reduzierung des Messfehlers erstellt werden. Der Kalibrierungsablauf sah dann folgendermaßen aus:

- das Hindernis auf dem Abstand von 212 mm vom Laserscanner platzieren,
- den Abstand im 20 mm Schritten ändern und pro Position 4.000 Messung durchführen,
- mittleren Fehler berechnen.

Die 13 Hindernisse mit verschiedenen Oberflächeneigenschaften wurden verwendet, um die Kalibrierung vom Laserscanner vorzunehmen. Die Ergebnisse haben erwiesen, dass sich der mittlere Fehler in einer linearen Abhängigkeit zum reellen Abstand befindet. Eine Funktion der Geraden konnte aus diesem Grund definiert werden, um den mittleren Fehler auszugleichen. Die Funktion hat die Form:

$$y = k\mu + b \tag{2-1}$$

mit

y die Abweichung vom reellen Abstand

μ der mittlere Fehler

k und b sind Konstanten, die die Summe der kleinsten Quadrate minimieren

Summer der kleinsten Quadrate:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - k * \mu_i - b)^2 \tag{2-2}$$

Aus der Formel (2-1)und (2-2) lassen sich die Konstanten k und b berechnen:

Die Mittelwerte von y und μ in die Formel(2-1) einsetzen und nach b umrechnen:

$b = \hat{y} - k\hat{\mu}$

b in die Formel (2-2) einsetzen. Anschließend nach k umrechnen:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - k * \mu_i - \hat{y} + k\hat{\mu})^2$$

$$\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\sum_{i=1}^{n}(y_i-\hat{y})^2-2k(y_i-\hat{y})(\mu_i-\hat{\mu})+k^2(\mu_i-\hat{\mu})^2$$

$$\sum_{i=1}^{n}(y_i-\hat{y}_i)^2-\sum_{i=1}^{n}(y_i-\hat{y})^2=\sum_{i=1}^{n}-2k(y_i-\hat{y})(\mu_i-\hat{\mu})+k^2(\mu_i-\hat{\mu})^2$$

$$0=\sum_{i=1}^{n}-2k(y_i-\hat{y})(\mu_i-\hat{\mu})+k^2(\mu_i-\hat{\mu})^2$$

Somit ergeben sich die Formeln für die Konstanten k und b:

$$k=\sum_{i=1}^{n}\frac{2(y_i-\hat{y})(\mu_i-\hat{\mu})}{(\mu_i-\hat{\mu})^2} \tag{2-3}$$

$$b=\hat{y}-k\hat{\mu} \tag{2-4}$$

mit

$\hat{y}$ und $\hat{\mu}$ Mittelwerte von y_i und μ_i

Die Kalibrierung sollte idealerweise zum Ausgleich des Fehlers bei verschiedener Orientierung und Oberflächeneigenschaft des Hindernisses beisteuern, da die Umgebung, in welcher der mobile Roboter eingesetzt wird, normalerweise unbekannt ist. Die Herren Cang Ye und Johann Borenstein haben die Messdaten vom Hindernis mit RGB=127 und Orientierung 0° verwendet, um die Konstanten zur Kalibrierung vom Laserscanner auszurechnen (k=1.0002 und b=-3.6). Die Messergebnisse nach der Kalibrierung sind in Abbildung 2-8 sowie Abbildung 2-9 zu sehen.

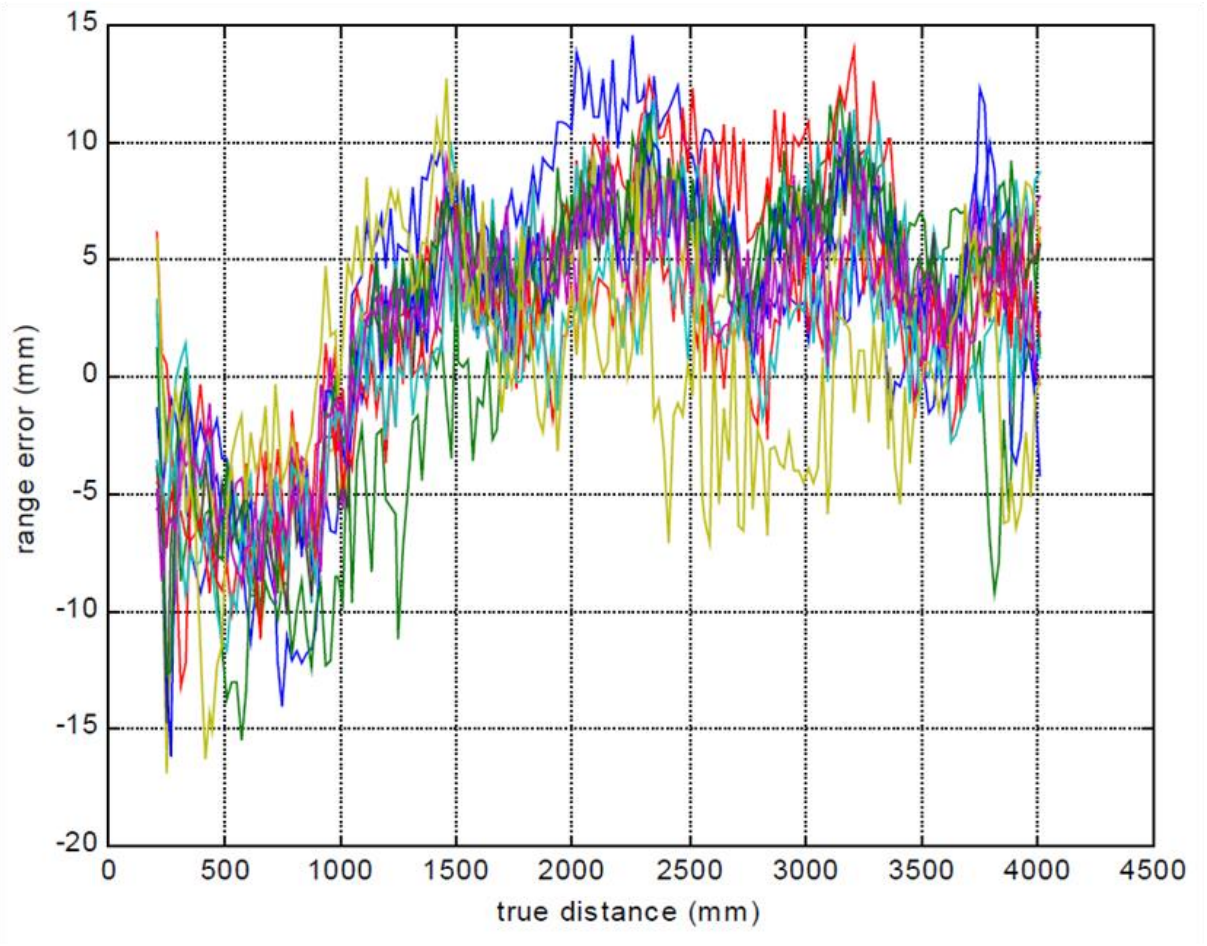


Abbildung 2-8 Messergebnisse nach der Kalibrierung: Fehler in Abhängigkeit vom realen Abstand

In Abbildung 2-8 sind die Messergebnisse mit 13 Hindernisse aufgezeichnet. Es ist zu sehen, dass der Fehler die Werte von -17 bis 14.5 mm annimmt. Der Fehler ist kleiner als der Fehler aus der Herstellerspezifikation.

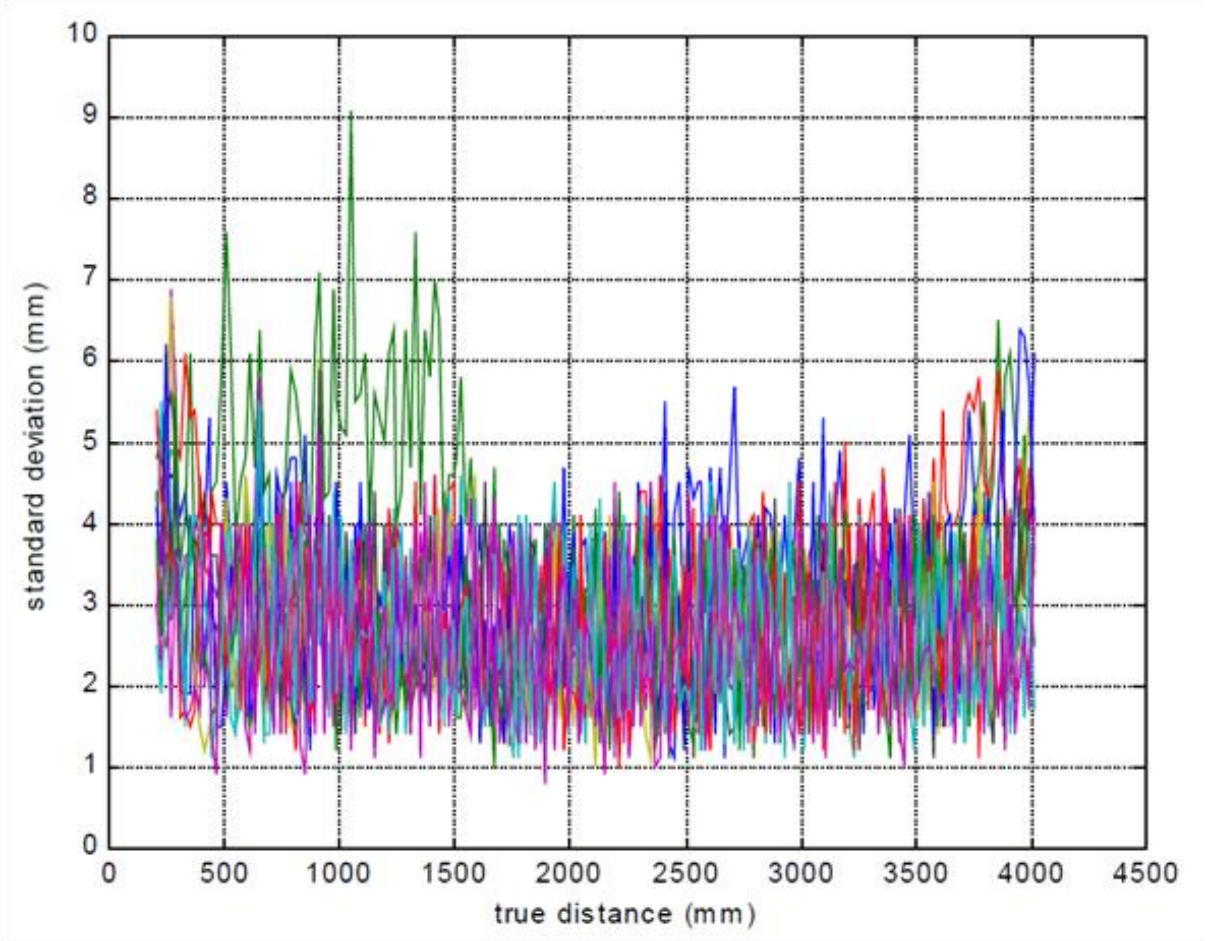


Abbildung 2-9 Messergebnisse nach der Kalibrierung: Standardabweichung in Abhängigkeit vom realen Abstand

In Abbildung 2-9 sind die Messergebnisse mit 13 Hindernisse dargestellt. Es ist zu sehen, dass die Standardabweichung die Werte von 0.8 bis 9.1 mm annimmt. Der Abweichung ist ebenfalls kleiner als die Abweichung aus der Herstellerspezifikation.

Fazit

Der Laserscanner LMS200 wurde in der Arbeit von Cang Ye und Johann Borenstein durch eine intensive sowie detaillierte Messreihe charakterisiert. Im Dauermessversuch stellte man fest, dass das Gerät eine Aufwärmphase von ungefähr drei Stunden hat und der Drift sich nach der Phase stabilisiert.

Das Fehlerverhalten bei verschiedenen Oberflächeneigenschaften und Orientierung des Laserstrahls ist ausgewertet. Somit konnte eine Aussage über den maximalen Fehler von 17 mm gemacht werden. Eine Untersuchung des Fehlerverhaltens im dynamischen Betrieb war nicht Teil der Arbeit.

2.2. Ermittlung der Genauigkeit von Laserscannern

Eine Ermittlung der Genauigkeit von Laserscannern mit verschiedenen Hindernissen ist in einer Forschungsarbeit von Wolfgang Boehler und Andreas Marbs detailliert beschrieben. Es erfolgten standardisierte Messungen mit unterschiedlichen Geräten, um den Vergleich zwischen den Geräten zu ermöglichen. Die Hindernisse mit unterschiedlichen Eigenschaften wurden wie bei der Arbeit aus dem Kapitel 2.1 verwendet, um die Genauigkeit der Geräte zu analysieren. Da die untersuchten Geräte unterschiedliche Scanwinkelauflösung haben, sind nicht alle Laserscanner in der Lage, kleine Objekte zu detektieren. Außerdem produzieren alle Laserscanner die Fehler beim Messen an den Ecken, das ist der sogenannte Edge Effect.

Für die vorliegende Masterarbeit sind die Testversuche von Interesse, die in der Forschungsarbeit von Wolfgang Boehler und Andreas Marbs aufgeführt sind. Die Nachweise von folgenden Genauigkeiten des Laserscanners werden in der Arbeit behandelt:

- Winkelgenauigkeit
- Messgenauigkeit
- Auflösung
- Edge Effect
- Einfluss der Oberflächeneigenschaften auf die Messergebnisse

Alle Nachweise wurden im statischen Betrieb des Laserscanners durchgeführt. Die Arbeit beinhaltet keine Information über die Möglichkeit des Genauigkeitsnachweises im dynamischen Betrieb.

3. Theoretische Grundlagen

3.1. Vorgehensmodelle

Aus der Informatik sind vielfältige Softwarevorgehensmodelle bekannt, die unterschiedlich gut auf die Automatisierungstechnik anwendbar sind. Im Folgenden soll das V-Modell als sehr bekanntes und einfaches Modell aus der Softwareentwicklung vorgestellt werden, so das darauf aufbauende 3-Ebenen-Vorgehensmodell für mechatronische Systeme von Bender [7].

3.1.1. V-Modell

Das V-Modell zählt zu den klassischen Modellen der Softwareentwicklung. Der linke Ast soll die Frage beantworten, was zu entwickeln ist und soll dies spezifizieren. In der Wurzel des V-Modells erfolgt die eigentliche Entwicklung von Komponenten, die anschließend einzeln getestet und auf dem rechten Ast an den Anforderungen verifiziert werden. Das V-Modell beginnt mit der Anforderungsanalyse. Anschließend erfolgt das Architekturdesign, bei dem eine grobe Idee der Architektur des Systems bestimmt werden soll. Die Komponenten des Gesamtsystems werden in der Komponentenentwicklung einzeln entwickelt, implementiert und getestet.

Im nächsten Schritt finden die Integration sowie die Verifikation des Gesamtsystems statt. Die Installation des Gesamtsystems erlaubt die Validierung der Systemanforderungen. Die Inbetriebnahme ermöglicht es, die Anforderungen der Nutzer zu überprüfen. In der Phase des Komponententestes werden die Programmierfehler gefunden, in der Phase der Integration werden Entwurfsfehler festgestellt und in der Phase der Installation und Validierung die Fehler in den erstellten Anforderungen.

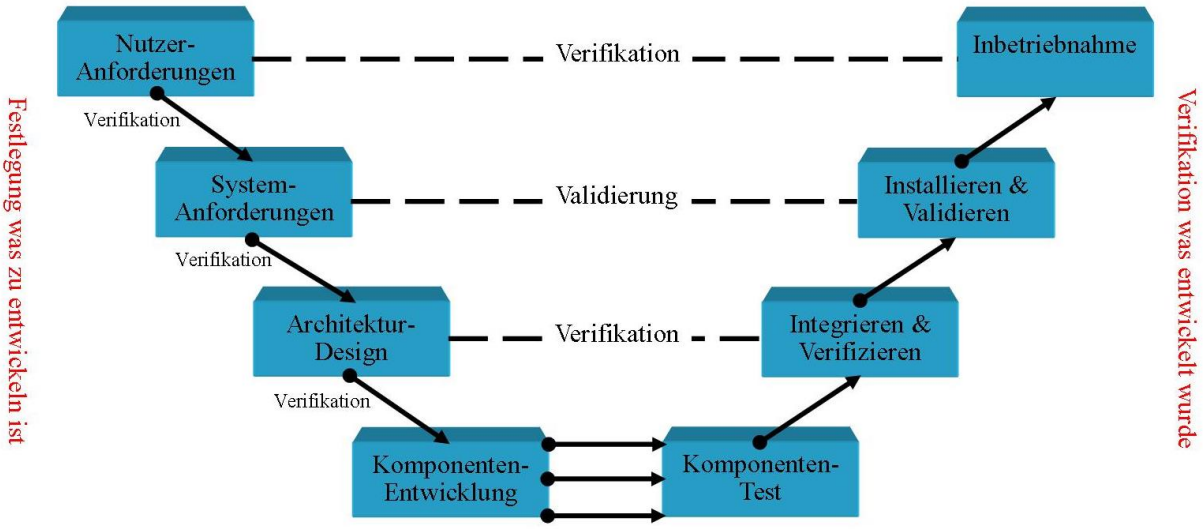


Abbildung 3-1 V-Modell

Entgegen der Abbildung 3-1 endet der Systemlebenszyklus nicht mit der Inbetriebnahme bzw. der Abnahme, sondern umfasst den Betrieb mit Wartung. Die Phase wird nicht in den klassischen V-Modellen berücksichtigt.

3.1.2. 3-Ebenen Vorgehensmodell

Die 3-Ebenen-Vorgehensweise basiert auf dem V-Modell und berücksichtigt die drei Disziplinen: Software, Hardware und Mechanik. Alle drei Disziplinen sind sehr wichtig bei der Entwicklung einer mechatronischen Maschine.

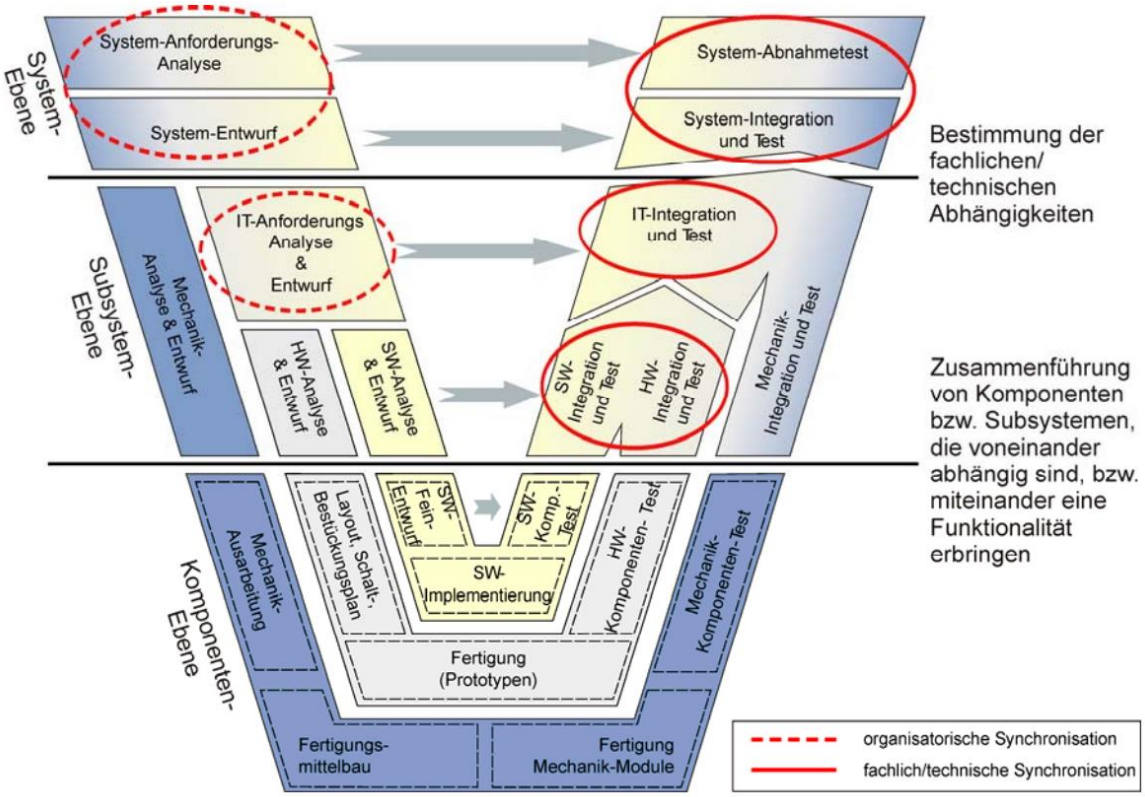


Abbildung 3-2 3-Ebenen Vorgehensmodell für mechatronische Systeme

Wie man in Abbildung 3-2 sehen kann, ist der gesamte Entwicklungsprozess in drei Ebenen unterteilt: System, Subsystem und Komponenten-Ebene. Die Einteilung reduziert die Komplexität. Auf der Systemebene werden das Gesamtsystem betreffende Fragestellungen bearbeitet. Ausgehend von den Anforderungen, wird zunächst die logische Systemarchitektur entworfen. Auf der Subsystemebene werden die Anforderungen an die Teilsysteme erneut analysiert. Die Mechanik kann an dieser Stelle bereits eigenständig arbeiten. Auf der untersten Ebene, der Komponenten-Ebene, erfolgt die eigentliche Realisierung der Maschine. Im aufsteigenden Ast des 3-Ebenen-Vorgehensmodells werden die entwickelten Komponenten schrittweise integriert sowie getestet.

3.2. Kinematisches Robotermodell

Ein Roboterarm ist nichts anders als ein Mehrkörpersystem mit einer bestimmten Anzahl von Körpern, die in Reihe bzw. parallel (Kapitel1) mithilfe der Gelenke verbunden sind. Anders ausgedrückt: Eine Roboterarm ist eine offene kinematische Kette. In Abbildung 3-3 ist diese Kette grafisch dargestellt.

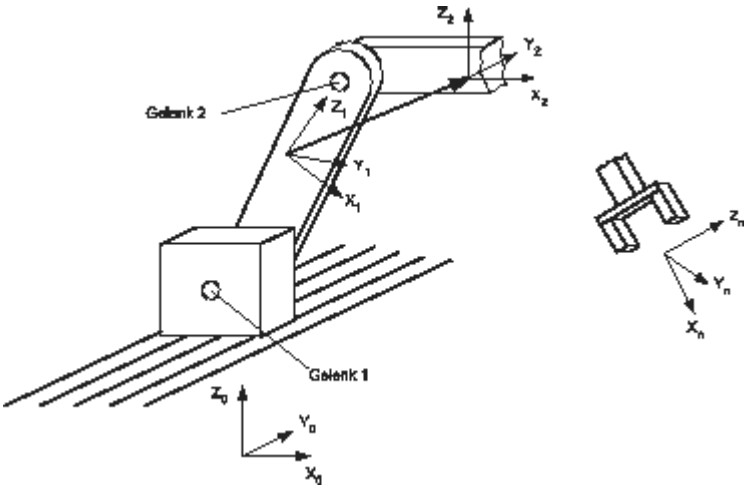


Abbildung 3-3 Offene, unverzweigte kinematische Kette

Der Bestand der Kinematik ist die Ermittlung von Lage, Geschwindigkeit und Beschleunigung der einzelnen Körper eines Mehrkörpersystems. Die Lage eines Körpers wird durch seine Position (z. B. seine Schwerpunktkoordinaten) sowie seine Orientierung (z. B. die Richtung seiner Hauptträgheitsachsen) erfasst. Aus der Lage werden anschließend durch eine Ableitung die Geschwindigkeit und Beschleunigung errechnet.

Um die Bewegung im Raum zu beschreiben, fällt die Wahl auf ein Referenzkoordinatensystem.

3.2.1. Rotationsmatrizen (Drehmatrizen)

Um die Lage eines Körpers im Raum zu bestimmen, verwendet die Wissenschaftler Rotationsmatrizen (Drehmatrizen). Es existieren in der Welt der Kinematik drei Drehmatrizen, die eine Drehung um X-, Y- bzw. Z-Achse repräsentieren. Nach Abbildung 3-4 werden positive Drehrichtungen als Rechtsschrauben um die jeweiligen Koordinatenachsen definiert.

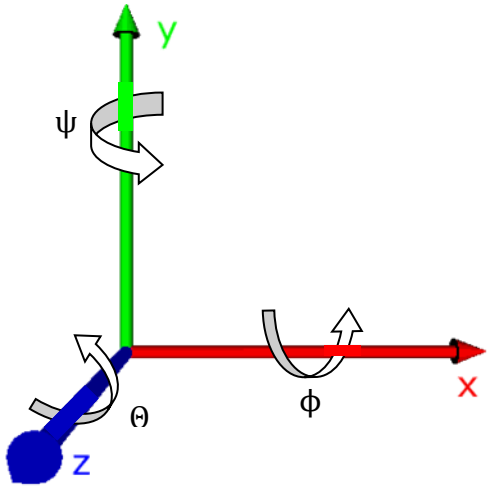


Abbildung 3-4 Positive Drehrichtung

Elementardrehungen

Das gedrehte Koordinatensystem wird mit dem Index R gekennzeichnet.

Koordinaten im nicht gedrehten System: $[x, y, z]^T$

Koordinaten im gedrehten System: $[u, v, w]^T$

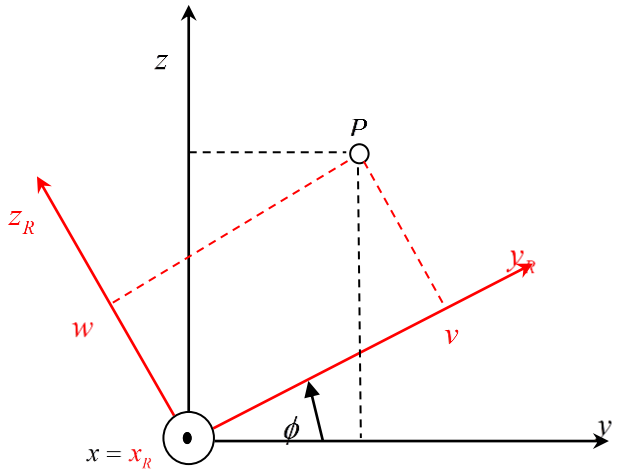


Abbildung 3-5 Elementardrehung um X-Achse

Drehung um die X-Achse:

In Abbildung 3-5 ist eine Elementardrehung um die X-Achse grafisch dargestellt. Der Zusammenhang zwischen den Koordinaten des Punktes P im Basiskoordinatensystem sowie im gedrehten Koordinatensystem lautet:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3-1}$$

Mit der Drehmatrix:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad \text{Drehung um die X-Achse} \tag{3-2}$$

$$R_y(\psi) = \begin{bmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{bmatrix} \quad \text{Drehung um die Y-Achse} \tag{3-3}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Drehung um die Z-Achse} \tag{3-4}$$

Zusammengesetzte Drehungen

Die häufigsten Algorithmen zur Beschreibung zusammengesetzter Drehungen stellen der Euler- und Kardan-Winkel dar.

Beide Algorithmen sind durch die Ausführung von drei Elementardrehungen in einer Reihe beschrieben.

Kardan-Winkel:		*Euler-Winkel:*	
Rotation um die X-Achse:	$R_x(\phi)$	Rotation um die Z-Achse:	$R_z(\phi)$
Rotation um die neue Y-Achse:	$R_y(\psi)$	Rotation um die neue X-Achse:	$R_x(\psi)$
Rotation um die neue Z-Achse:	$R_z(\theta)$	Rotation um die neue Z-Achse:	$R_z(\theta)$

3.2.2. Homogene Koordinaten und Transformationen

Homogene Koordinaten:

$$x = \begin{bmatrix} r \\ 1 \end{bmatrix} \in \mathbb{R}^4 \qquad x_0 = \begin{bmatrix} r_0 \\ 1 \end{bmatrix} \in \mathbb{R}^4$$

mit

einem Ortsvektor im Initialsystem $r = [x, y, z]^T$ und

einem Ortsvektor im Koordinatensystem $r_0 = [x_0, y_0, z_0]^T$ (Translationsvektor)

Allgemein gilt für die homogenen Koordinaten $x = \begin{bmatrix} r \\ \lambda \end{bmatrix} \in \mathbb{R}^4$ mit einem skalaren Maßstabsfaktor λ.

Homogene Transformationsmatrix:

$$T = \begin{bmatrix} \textit{Rotationsmatrix} & \textit{Translationsvektor} \\ 0 \quad 0 \quad 0 & \textit{Maßstabsvektor} \end{bmatrix} = \begin{bmatrix} R & r_0 \\ 0^T & 1 \end{bmatrix} \in \mathbb{R}^4$$

Aus diesem Grund gilt für eine einfache Transformation aus einem Punkt ${}_{(0)}x_p = \begin{bmatrix} {}_{(0)}r_p \\ 1 \end{bmatrix}$

in einen anderen ${}_{(i)}x_p = \begin{bmatrix} {}_{(i)}r_p \\ 1 \end{bmatrix}$:

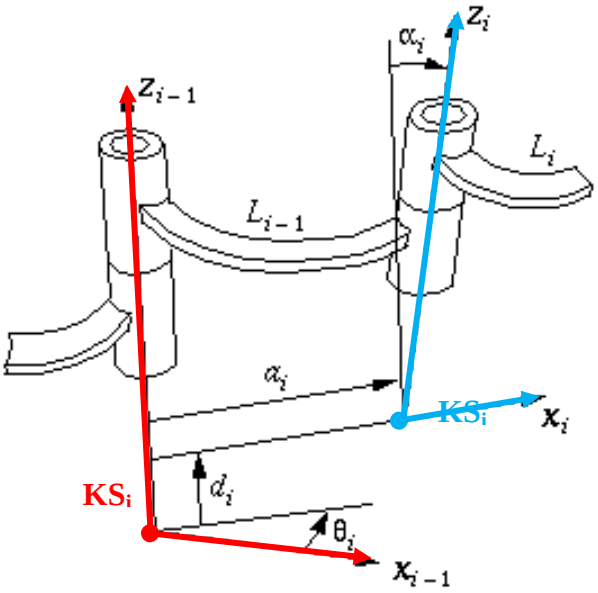
$${}_{(0)}x_p = T_i^0 \, {}_{(i)}x_p$$

mit Transformationsmatrix : $T_i^0 = \begin{bmatrix} {}^{0i}R & {}_{(0)}r_0 \\ 0^T & 1 \end{bmatrix}$

Die Inverse Transformation wird durch $T^{-1} = \begin{bmatrix} R^T & -R^T r_0 \\ 0^T & 1 \end{bmatrix}$ beschrieben.

3.2.3. Denavit-Hartenberg-Notation (DH-Notation)

Die DH-Notation stellt eine Methode zur Beschreibung der kinematischen Bedingungen dar und wurde zur Behandlung der Kinematik räumlicher Mechanismen entwickelt. Die Methode hat eine weite Verbreitung in der Robotertechnik gefunden. Sie basiert auf der bereits vorgestellten 4x4-Matrixdarstellung von Position und Orientierung eines Koordinatensystems und benutzt eine minimale Anzahl von Parametern
(DH-Parameter) zur vollständigen Beschreibung der Kinematik eines Roboters.
Die Idee besteht darin, eine möglichst eindeutige Vorschrift für die körperfesten Koordinatensysteme anzugeben. Dabei ist es aus kinematischen Gründen sinnvoll, die Gelenkachsen eines Mechanismus als eine Koordinatenachse zu wählen. Die Z-Achse zeigt nach DH-Notation in die Richtung des jeweiligen Gelenks.



Regel1: Der Koordinatenursprung KS_i liegt im Schnittpunkt der gemeinsamen Normalen von Gelenk i-1 und i mit der Gelenkachse i.

Regel2: Die Orientierung des KS_i wird so vorgenommen, dass die:

- Z-Achse in Richtung der i-ten Gelenkachse,
- X-Achse in Richtung der verlängerten gemeinsamen Normalen zeigt und die
- Y-Achse sich aus der Bedingung nach einem Rechtsystem bestimmen lässt.

Abbildung 3-6 Denavit-Hartenberg-Notation

Die Lage des i-ten Koordinatensystems (KS_i) relativ zur KS_{i-1} wird durch vier DH-Parameter (θ, d, a, α) bestimmt:

θ_i: Drehwinkel um die Z_{i-1} -Achse

d_i: Verschiebung entlang der Z_{i-1} -Achse

a_i: Verschiebung entlang gemeinsamen Normalen

α_i: Drehwinkel um die X_i -Achse

Anhand der Definition kann man die Transformationsmatrix wie folgt berechnen:

$$T_i^{i-1} = Rotation(z,\theta_i) * Translation(a_i, 0, d_i) * Rotation(x,\alpha_i) \tag{3-5}$$

$$Rotation(z,\theta_i) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-6}$$

$$Translation(a_i, 0, d_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-7}$$

$$Rotation(x, \alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-8}$$

3.2.4. Direkte und inverse Kinematik

Um das kinematische Modell eines Mehrkörpersystems vollständig zu beschreiben, muss man einen Zusammenhang zwischen den verallgemeinerten Koordinaten q sowie Umweltkoordinaten x eines Mehrkörpersystems finden.
$x = f(q)$
Von besonderer Bedeutung ist die Berechnung der Lage des Endeffektors.

Direkte Kinematik:
Jedem $q = [q_1, q_2, ..., q_n]$ entspricht genau eine Lage des Endeffektors (EE) im Umweltkoordinatensystem $x = [x, y, z, \phi, \psi, \theta]$. Die Lage kann durch die einfache Auswertung der Vektorgleichung $x = f(q)$ berechnet werden.

${}_0x_{EE} = T_i^0 {}_i x_{EE}$ mit

$T_i^0 = T_1^0 * T_2^1 * ... * T_i^{i-1}$

Inverse Kinematik:
Zu einer gewünschten Position und Orientierung des Endeffektors werden geeignete verallgemeinerte Koordinaten q gesucht, d.h. es ist eine geeignete Konfiguration des Mehrkörpersystems zu bestimmen.
$q = f^{-1}(x)$
Die Lösung der Umkehrfunktion stößt aus folgenden Gründen auf Probleme:

- Aufgrund der Nichtlinearität von $f(q)$ sind analytische Lösungen lediglich in Ausnahmefällen zu finden.
- Die Lösungen sind nicht zwingend eindeutig. Sie können endlich oder unendlich sein.

Beim Entwurf ist es notwendig, auf folgende Fälle zu achten:

- $\dim(q) = \dim(x)$: Normales System
  Die Anzahl der Freiheitsgrade entspricht der Dimension des Umweltvektors. Die Gleichung $q = f^{-1}(x)$ ist bis Symmetrie eindeutig.
- $\dim(q) < \dim(x)$: Unterbestimmtes System
  Das System hat zu wenige Freiheitsgrade. Die Gleichung ist lediglich in Sonderfällen lösbar.
- $\dim(q) > \dim(x)$: Überbestimmtes System (Redundantes System)
  Das System hat mehr Freiheitsgrade, als zur Lösung benötigt wird. Die Gleichung $q = f^{-1}(x)$ hat unendlich viele Lösungen.

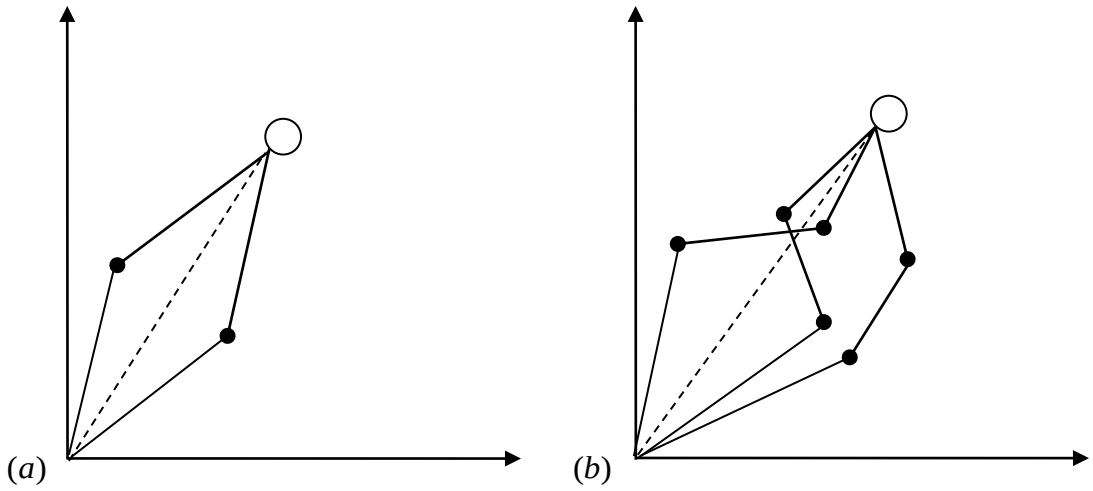


Abbildung 3-7 Lösbarkeit der inversen Kinematik (a) dim(q)=2, dim(x)=2; (b) dim(q)=3, dim(x)=2

In Abbildung 3-7 verdeutlicht sich, wie die Lösbarkeit einer inversen Kinematik von Dimensionen des verallgemeinerten Koordinatensystems und des Umweltkoordinatensystems abhängt.

3.3. Controller Development System

CoDeSys[1] ist ein geräteunabhängiges Steuerungsprogrammiersystem der Softwarehersteller 3S – Smart Software Solutions. Das Programmiersystem hat eine eigene Entwicklungsumgebung für speicherprogrammierbare Steuerungen (SPS) nach dem IEC 61131-3 Standard für die Applikationsentwicklung in der Industrieautomation.

Das Programmiersystem CoDeSys hat viele Fähigkeiten. Es erlaubt u. a. das Einbinden von C-Routinen und unterstützt die objektorientierte Programmierung. Zusammen mit dem CoDeSys-SP-Laufzeitsystem ermöglicht es ebenso die Multi-Device- und Multi-Application-Programmierung. Unter dem Begriff Laufzeitprogramm versteht man ein Gesamtprogramm mit Laufzeit-Eigenschaften, bestehend aus sämtlichen benötigten POUs (**P**rogram**O**rganisation **U**nit) bzw. POEs (**P**rogramm **O**rganisation **E**inheit) sowie allen Tasks. Mit dieser Definition ist ein Laufzeitprogramm eine in sich geschlossene Programmeinheit, die selbstständig in einer CPU ablaufen kann. Die komponentenbasierte Struktur der Programme in CoDeSys macht eine kundenspezifische Konfiguration und Erweiterung der Benutzeroberfläche realisierbar. Über die Grenzen von Ein-Prozessor-Programmen hinaus ermöglicht ein Softwaremodell in CoDeSys nach der IEC 61131-3-Norm das Multi-Tasking einer CPU, die Zusammenarbeit mehrerer CPUs oder die Zusammenarbeit vernetzter Systeme. [10]

3.3.1. Aufbau des Programms

Ein Steuerungsprogramm (Anwenderprogramm) stellt eine in Programm-Organisationseinheiten gegliederte logische Anordnung von Sprachelementen und Sprachkonstrukten dar. Die Programm-Organisationseinheiten unterscheiden sich in drei verschiedene Bausteintypen. Die Bausteintypen sind in Abbildung 3-8 mit einer Funktionalitätsanzeige der einzelnen Bausteintypen veranschaulicht.

Abbildung 3-8 Programm-Organisationseinheiten

[1] **Co**ntroller **De**velopment **Sys**tem (Entwicklungssystem für Automatisierungssysteme)

Alle fünf der von der IEC 61131-3 spezifizierten Programmiersprachen, die in CoDeSys realisiert sind, sind konform zu den Anforderungen der Norm und stehen in CoDeSys zur Verfügung. Die textuellen Sprachen sind:

- Anweisungsliste (AWL),
- Strukturierter Text (ST).

Bei den grafischen Sprachen handelt es sich um:

- Ablaufsprache (AS),
- Kontaktplan (KOP),
- Funktionsplan (FBS)
- ContinuousFunction Chart (CFC),

wobei CFC nicht Teil der IEC 61131-3 ist. Die Benennung der Programmiersprachen ist in den verschiedenen Systemen sowie im deutsch- und englischsprachigen Raum unterschiedlich. Die Unterschiede zwischen der deutschen und englischen Benennung sind in der Tabelle 3-1 aufgelistet.

Tabelle 3-1 Vergleich der Benennung der Programmiersprachen

Deutsch	**English**
Kontaktplan (KOP)	LadderDiargram (LD)
Funktionsbausteinsprache (FBS)	Fuction Block Diagram (FBD)
Anweisungsliste (AWL)	Instruction List (IL)
Strukturierter Text (ST)	Structered Text (ST)
Ablaufsprache (AS)	SequentialFunction Chart (SFC)

Ein wesentliches Leistungsmerkmal der IEC 61131-3-Sprachfamilie ist die Umschaltbarkeit zwischen den Programmiersprachen. [11]

3.3.2. Erste Schritte

In diesem Unterkapitel wird beschrieben, wie ein einfaches Projekt, das ein Programm enthält, in CoDeSys V3 angelegt, erstellt, über den Gateway-Server auf das Zielsystem (Steuerung) geladen und dort gestartet sowie überwacht werden kann.

Um ein neues Projekt anzulegen, wählt man den Befehl **Neues Projekt** im **Datei**-Menü. Die Abbildung 3-9 zeigt, wie ein neues Projekt anzulegen ist.

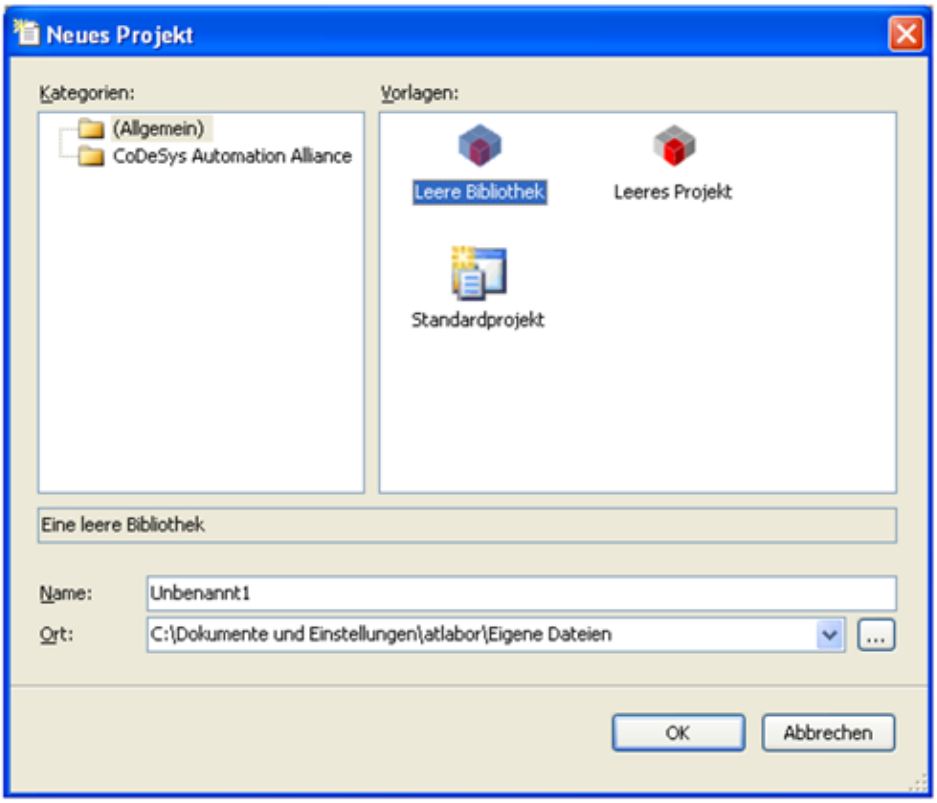


Abbildung 3-9 Neues Projekt anlegen

Im Dialog Neues Projekt wählt man im Feld Vorlagen die Vorlage Standardprojekt. Nach Eingabe des Projektnamens und dessen Speicherorts für die Projektdatei und anschließender Bestätigung der Eingabe durch OK erscheint ein Dialog-Fenster zur Auswahl des programmierbaren Gerätes (Abbildung 3-10).

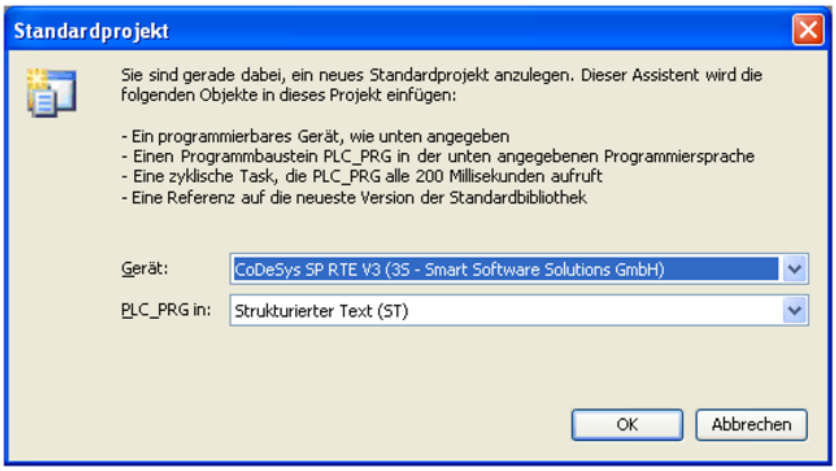


Abbildung 3-10 Geräteauswahl

Nach Auswahl des programmierbaren Gerätes und der Programmiersprache (hier Strukturierter Text ST) des Programmbausteins PLC_PRG erscheint die CoDeSys- Benutzeroberfläche (Abbildung 3-11) mit den zugehörigen POUs- und Geräte-Fenstern.

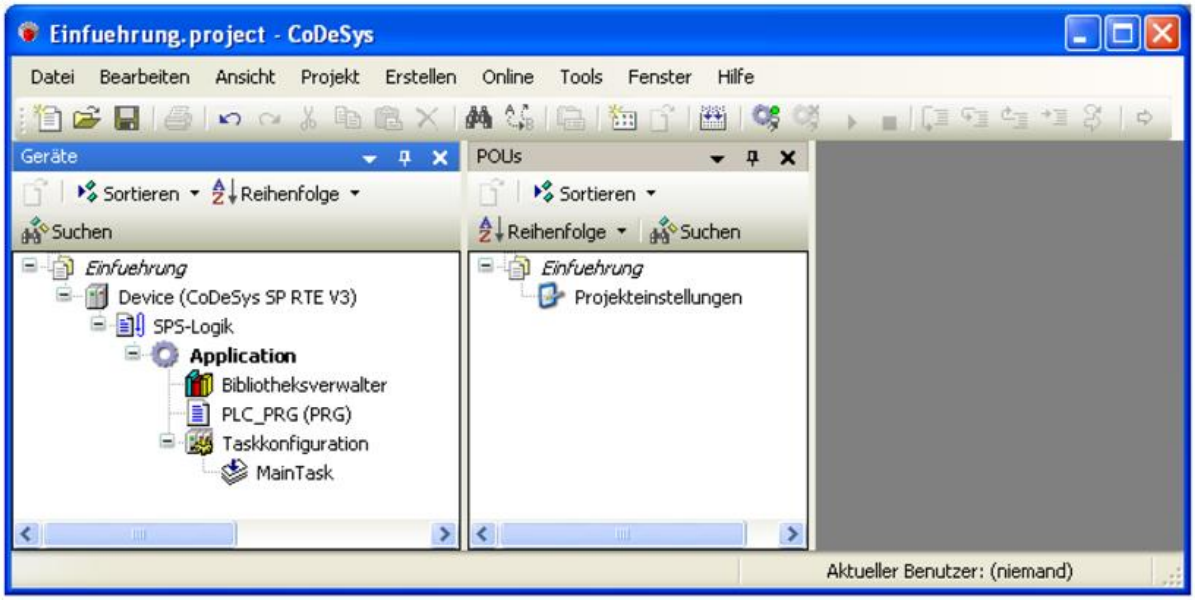


Abbildung 3-11CoDeSys-Benutzeroberfläche

Das POUs-Fenster enthält die Projekteinstellungen. Mit POU bzw. POE werden Programme, Funktionen und Funktionsbausteine bezeichnet, die das Anwenderprogramm, das zur Automatisierung einer Anlage dient, zusammenstellen.

Das Geräte-Fenster weist eine Baumstruktur auf, welche das Gerät Device (CoDeSys SP RTE V3) vom Typ CoDeSys SP RTE V3 mit einer unterhalb eingefügten Application zeigt. Letztere beinhaltet das Programm PLC_PRG sowie die obligatorische Taskkonfiguration mit einer MainTask zur Steuerung von PLC_PRG.

Die Abarbeitung eines CoDeSys-Programms beginnt zwingend mit dem speziellen Baustein PLC_PRG, welcher andere Bausteine aufrufen kann. Weiterhin ist bereits ein Bibliotheksverwalter eingehängt, der automatisch die Bibliothek IoStandard.library enthält, die für E/A-

Konfigurationen benötigt wird, sowie die Standard.library, die alle Funktionen und Funktionsbausteine bereitstellt, die gemäß der Norm IEC 61131-3 als Standardbausteine für ein IEC-Programmiersystem benötigt werden. Der zusätzliche Knoten SPS-Logik unterhalb des Knotens Device (CoDeSys SP RTE V3) ist lediglich ein symbolischer Knoten, welcher anzeigt, dass das Gerät „programmierbar“ ist.

3.3.3. Steuerungsprogramm schreiben

Durch einen Doppelklick auf den Eintrag PLC_PRG im Geräte-Fenster öffnet sich das ST-Editorfenster (Abbildung 3-12). Der Editor besteht aus einem oberen Deklarationsteil sowie einem unteren Implementierungsteil. Im Deklarationsteil werden zwischen den Schlüsselwörtern VAR und END_VAR die Variablen deklariert, d. h. ihre Namen, Datentypen und Initialwerte. Für Ein- und Ausgabevariablen wird u. a. von den Schlüsselwörtern VAR_INPUT für die Formalparameter, VAR_OUTPUT für Rückgabewerte und VAR_IN_OUT für les- und beschreibbare Variablen Gebrauch gemacht. Der Implementierungsteil beinhaltet den in ST geschriebenen Programmcode.

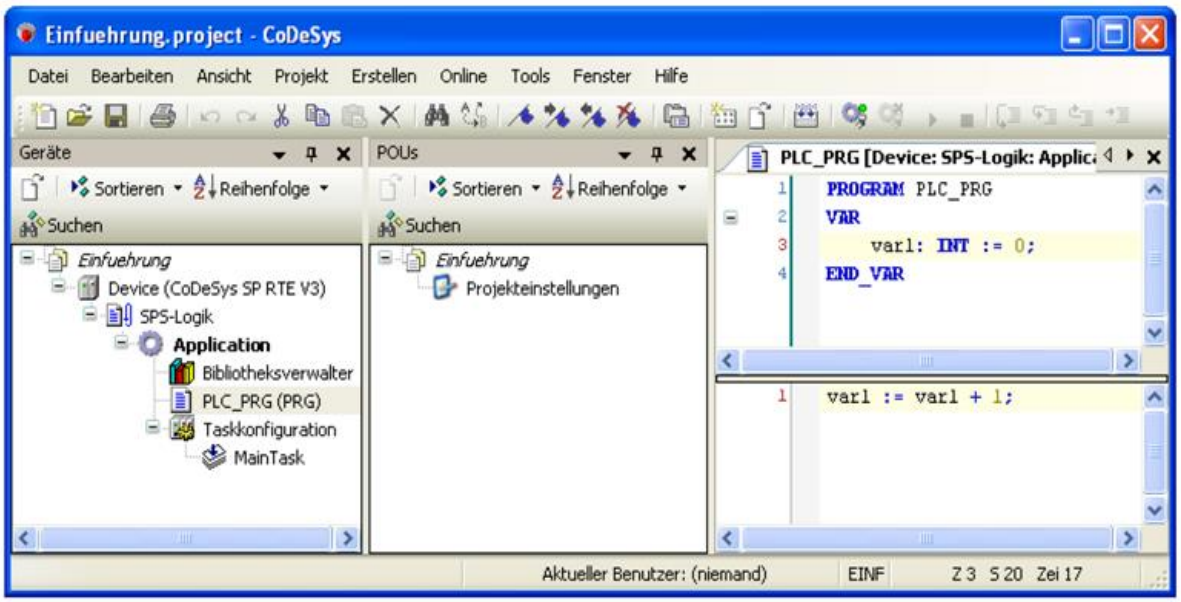


Abbildung 3-12 ST-Editor

Alternativ kann die **Autodeklarations**-Funktion Anwendung finden, um während des Programmierens Variable zu deklarieren. Für jede noch nicht deklarierte Variable der aktuellen Programmzeile wird automatisch der Dialog **Variable deklarieren** geöffnet, wo die Deklaration vorgenommen werden kann (Abbildung 3-13).

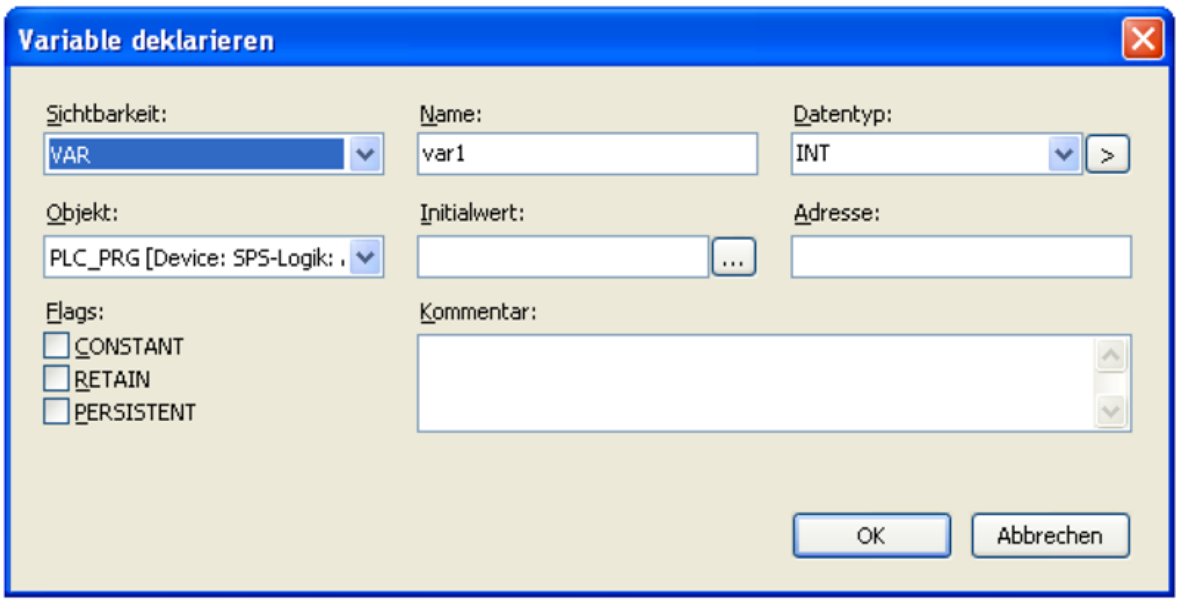


Abbildung 3-13 Autodeklaration einer Variable

Um weitere POUs anzulegen, wählt man den Befehl Objekt hinzufügen im Projekt-Menü (Abbildung 3-14). Zusätzlich stehen im geöffneten Dialog-Fenster alle in CoDeSys V3 einfügbaren Objekte mit ihren Eigenschaften aufgelistet zur Verfügung. Nach Angabe des POU-Namens, des Typs sowie der Implementierungssprache werden mit dem Bestätigen der Schaltfläche Öffnen die Objekteinstellungen vorgenommen und es zeigt sich automatisch das Editor-Fenster des eingefügten Objektes, in welchem sich dieses bearbeiten bzw. programmieren lässt.

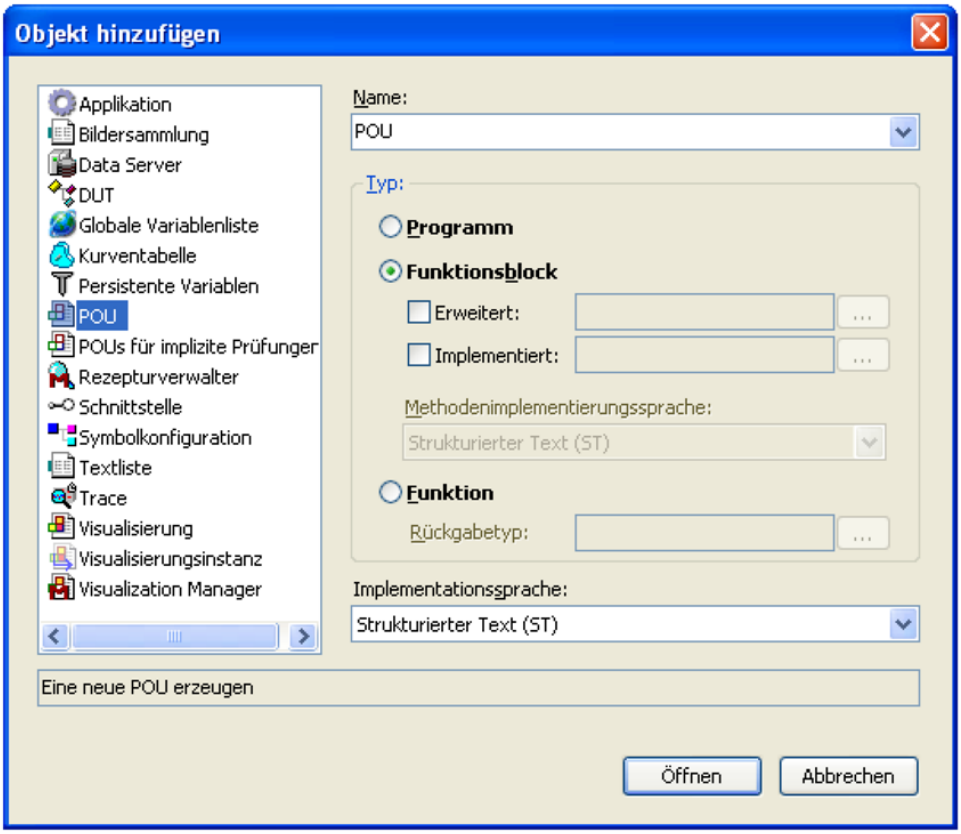


Abbildung 3-14 Objekt hinzufügen

Das Zielsystem CoDeSys SP RTE V3 wird beim Systemstart automatisch als Dienst verfügbar. Bevor ein Programm auf die Steuerung geladen werden kann, muss sichergestellt werden, ob das **CoDeSys SP RTE V3**-Symbol den Status running aufweist. Das Symbol befindet sich in der Systemleiste am unteren Rand des Bildschirms. Abbildung3-15 zeigt den laufenden RTE-Kontroller.

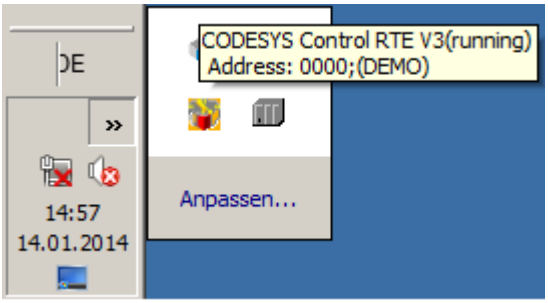


Abbildung3-15 CoDeSys Control RTE (running)

Um eine Applikation als „aktive“ Applikation zu setzen, wählt man im **Geräte**-Fenster den Eintrag Application. Bei einem anschließenden Klick auf die rechte Maustaste erscheint ein Kontextmenü, das den Befehl **Aktive Applikation setzen** enthält. Damit bezieht sich alle Kommunikation mit der Steuerung auf diese Applikation.

3.3.4. Taskkonfiguration

Die Hierarchie des Aufrufs der Programmbausteine wird im Geräte-Fenster mit der Ressource Taskkonfiguration organisiert.

In einer SPS verläuft die Informationsverarbeitung zyklisch. Die Verarbeitungsschritte lassen sich folgendermaßen ordnen:

- Die Eingange einlesen
- Verarbeitung der aktuellen Information im SPS-Programm
- Ausgänge aktualisieren

Eine oder mehrere Tasks können in einer CPU ablaufen. Sie organisiert den zeitlichen Ablauf der Programme. Außerdem können einer Task mehrere Programme zugeordnet sein, die alle mit derselben Zykluszeit abgearbeitet werden. Erst nach Abarbeitung aller zu einer Task zählenden Programme werden die Stellwerte an den Prozess ausgegeben. Trotz der hierarchischen Abarbeitung der Programme erreichen die Ausgangsdaten aller Programme erst am Ende des Bearbeitungszyklus und somit quasi gleichzeitig den Prozess. [12]

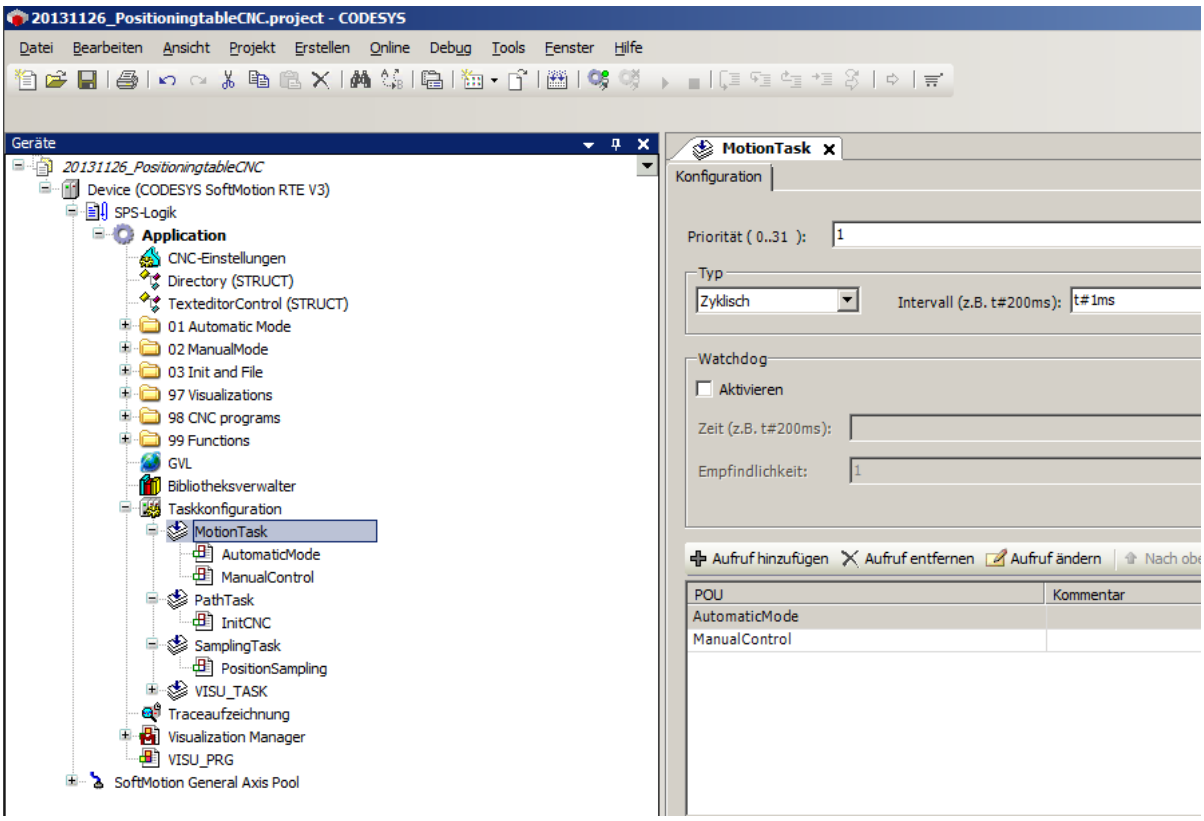


Abbildung 3-16 Taskkonfiguration

Abbildung 3-16 zeigt das **Konfigurationsdialog**-Fenster für den Hauptprogramm-Task. Hier können Definitionen für die folgenden Parameter festgelegt werden:

- Priorität des Tasks,
- Typ: – Zyklisch mit Intervallangabe
  – Ereignisgesteuert mit Angabe der Ereignisvariable
  – Freilaufend,
- Watchdog zur Überwachung der Task aktivieren/deaktivieren, Auslösezeit und dessen Empfindlichkeit,
- Aufzurufende POUs.

3.3.5. Starten einer Applikation

Um lediglich eine syntaktische Prüfung der „aktiven“ Applikation durchzuführen, verwendet man den Befehl **Application […] übersetzen** aus dem Kontextmenü oder dem **Übersetzen**-Menü. In diesem Fall wird noch kein Code generiert. Die Resultate der Prüfung werden im Meldungsfenster ausgegeben, das standardmäßig im unteren Drittel der Benutzeroberfläche platziert ist. Bei fehlerfreier Übersetzung des Quellcodes kann die Software dann auf die Steuerung geladen und anschließend in die Steuerung eingeloggt werden.

Wählt man den Befehl **Start Application** im **Online**-Menü, so läuft das Programm. Die Einträge für Steuerung und Applikation im Geräte-Fenster werden grün hinterlegt und hinter Applikation findet sich der Status RUN (Abbildung 3-17).

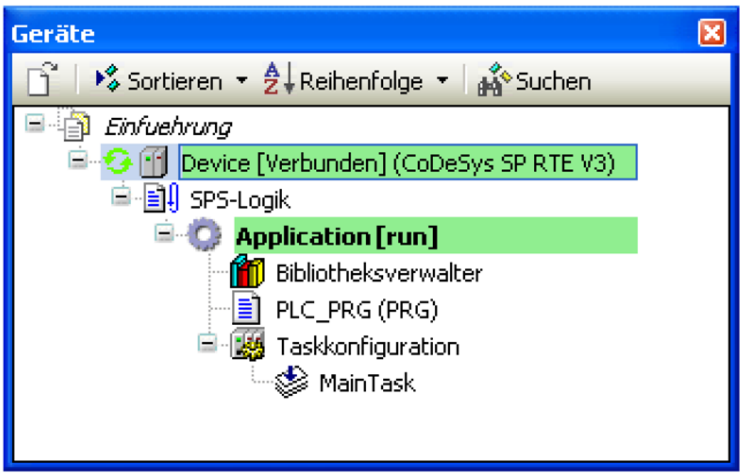


Abbildung 3-17 RUN-Status einer Applikation

Online-Funktionen

Nach dem Einloggen in die Steuerung stehen in CoDeSys Befehle für Online-Funktionen zur Verfügung. Sie sind im Online-Menü aufgelistet. Die Ausführung einiger Online-Funktionen ist abhängig vom aktiven Editor.

Einer der wichtigsten Online-Befehle ist der Reset-Befehl, der ein Rücksetzen der Steuerung bewirkt. Es wird zwischen drei Reset-Arten unterschieden:

Reset (warm): Dieser Befehl setzt, mit Ausnahme der remanenten[2] Variablen, alle Variablen auf ihren Initialisierungswert zurück. Variablen, die nicht explizit mit einem Initialisierungswert versehen wurden, werden auf die Standardinitialwerte gesetzt. Integer-Zahlen werden beispielsweise auf 0 gesetzt. Bevor alle Variablen überschrieben werden, erfolgt eine Sicherheitsabfrage durch CoDeSys. Dies entspricht der Situation bei einem Stromausfall oder beim Aus- bzw. Einschalten der Steuerung während des laufenden Programms.

Reset (kalt): Dieser Befehl setzt mit Ausnahme der persistenten[3] Variablen sowie der Retain-Variablen alle Variablen auf den Initialisierungswert zurück. Dies entspricht einem Neuladen des Steuerungsprogramms auf die Steuerung.

Reset (Ursprung): Dieser Befehl setzt alle Variablen, ebenso die remanenten Variablen, auf ihren Initialisierungswert zurück und löscht das Anwenderprogramm auf der Steuerung, d. h. die Steuerung wird in den Urzustand zurückversetzt.[10]

3.3.6. Fehlersuche und Fehlerbehebung

Die Fehlersuche wird mit den **Debugging**-Funktionen von CoDeSys erleichtert. Sogenannte Haltepunkte (engl. Breakpoints) können beispielsweise im Falle eines Programmierfehlers gesetzt werden. Stoppt die Ausführung des Programms in einem solchen Haltepunkt, können die Werte sämtlicher Projektvariablen zu diesem Zeitpunkt eingesehen werden. Durch schrittweises Abarbeiten (Einzelschritt) kann eine Überprüfung der logischen Korrektheit des Programms stattfinden. [10]

---

[2] Remanente Variable können ihren Wert über die übliche Programmlaufzeit hinaus behalten. Sie werden als Retain-Variablen oder noch strenger als Persistente Variable deklariert.

[3] Persistente Variablen werden mit dem Schlüsselwort PERSISTENT gekennzeichnet. Sie werden nur bei Reset (Ursprung) der Steuerung neu initialisiert.

3.4. SLAM Algorithmen

Die SLAM[4]-Aufgabe besteht darin, eine Karte der Umgebung zu erstellen und gleichzeitig die Position eines Laserscanners zu berechnen. Der Sensor ist dabei i. d. R. auf einem mobilen System installiert und erfasst die Daten, die für die Kartenerstellung und Positionsbestimmung zum Einsatz gelangen. Die Daten bestehen für gewöhnlich aus Beobachtungen von Landmarken und Messungen der Bewegung des Sensors. Sowohl die Beobachtungen der Landmarken als auch die Messungen der Bewegung sind mit einem Fehler überlagert und lassen daher lediglich eine Schätzung der Position und Karte zu.

3.4.1. Sensorzentrierte Position einer Landmarke

Um die Position einer Landmarke zum Zeitpunkt t in einer sensorzentrierten Umgebung wiederzugeben, bedarf es nur einer einzigen Wertekombination: die Landmarkenposition aus der Sicht des Sensors in Polarkoordinaten. Es gilt für die sensorzentrierte Position einer Landmarke:

$$z_t \triangleq (r_t, \varphi_t) \in \mathbb{R} \times [0,2\pi) \tag{3-9}$$

Der erste Wert r (Radius) repräsentiert die Distanz der Landmarke zum Sensor und der zweite Wert φ repräsentiert den Winkel, mit dem die Landmarke beobachtet wurde.

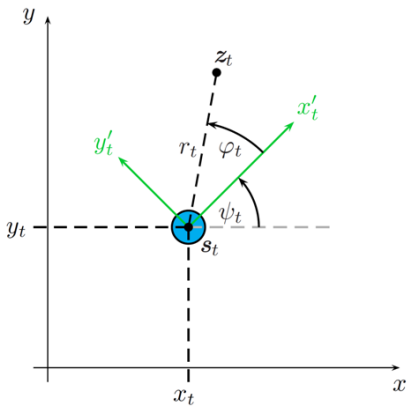


Abbildung 3-18 Sensorzentrierte Position einer Landmarke

Zudem definiert man die Menge aller Landmarkenbeobachtungen z^t. Sie besteht aus den einzelnen sensorzentrierten Landmarkenpositionen $\{z_1, z_2, \ldots, z_t\}$.

[4]Simultaneous Localization and Mapping (von englisch *Simulteous* „simultane“, *Localization* „Lokalisierung“, *and* „und“, *Mapping* „Kartierung“)

3.4.2. Karte der Umgebung

Die Karteθ_t der Umgebung setzt sich aus der Menge aller Landmarken $\{\theta_1,\theta_2,\ldots,\theta_{N_t}\}$ zum Zeitpunkt t zusammen. N_t repräsentiert die Kordialität dieser Menge und ist ebenso von der Zeit t abhängig, da sich die Karte während der Verarbeitung verändern kann. Abbildung 3-19 zeigt ein Beispiel einer Karte mit Landmarken.

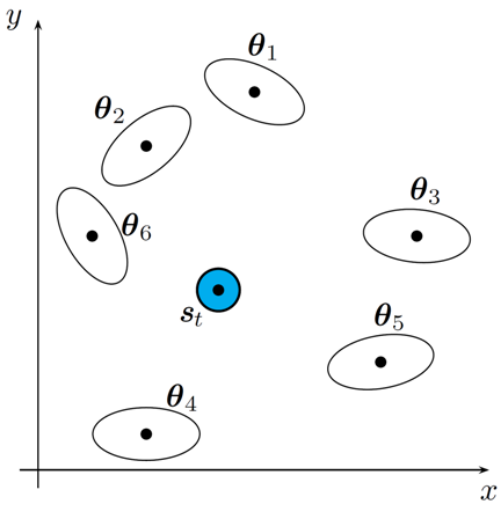


Abbildung 3-19 Die Karte θ_t der Umgebung

3.4.3. Das SLAM Problem

Abbildung 3-20 stellt das SLAM Problem grafisch dar und zeigt die Ungenauigkeit der Positionsschätzung des Sensors entlang des Pfades sowie die Ungenauigkeit der Positionsschätzungen der Landmarken. Zu erkennen ist, dass sich sowohl die Ungenauigkeit der Positionsschätzung des Sensors als auch die Ungenauigkeit der Positionsschätzung der Landmarken mit der Zeit vergrößern. [13]

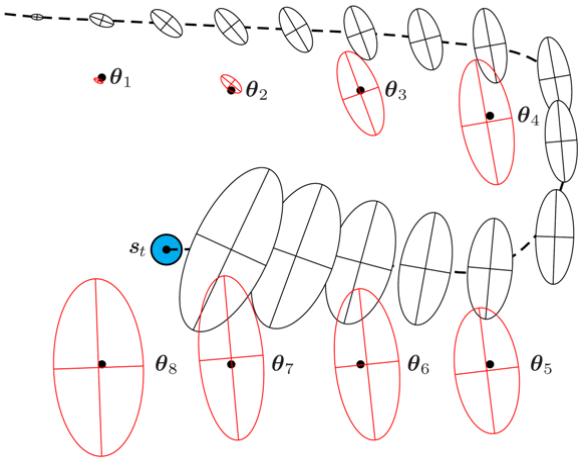


Abbildung 3-20 Vergrößerung der Ungenauigkeit

3.5. Navigationshardware NAV350

Der NAV350 ist ein Laserscanner, welcher sich der in Abbildung 3-21 gezeigten Abtastmethode bedient:

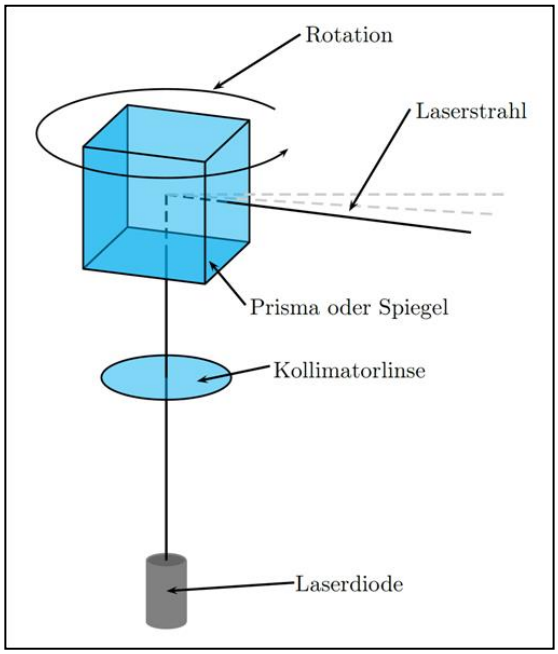


Abbildung 3-21 Funktionsprinzip eines zweidimensionalen Lasermesssystems

Das Funktionsprinzip des zweidimensionalen Lasermesssystems beruht auf der Kombination eines Lasersenders mit einem beweglichen Drehspiegel oder Prisma zur Auslenkung des Strahlenbündels. Das von einer Laserdiode erzeugte Strahlenbündel wird mittels einer Kollimatorlinse gebündelt und kann nun durch Drehung eines Spiegels oder Prismas um einen bestimmten Winkel ausgelenkt werden. Abbildung 3-22 zeigt eine prinzipielle Konfiguration für ein Lasermesssystem mit folgenden Bestandteilen:

PS	Energieversorgung der internen Bauteile
µP	Mikroprozessor
S	Strahlsender
E	Strahlempfänger
M	Antriebsmotor für Prisma oder Spiegel
E-Interface	Ethernet-Schnittstelle
I/O-Interface	Eingabe- und Ausgabeschnittstelle

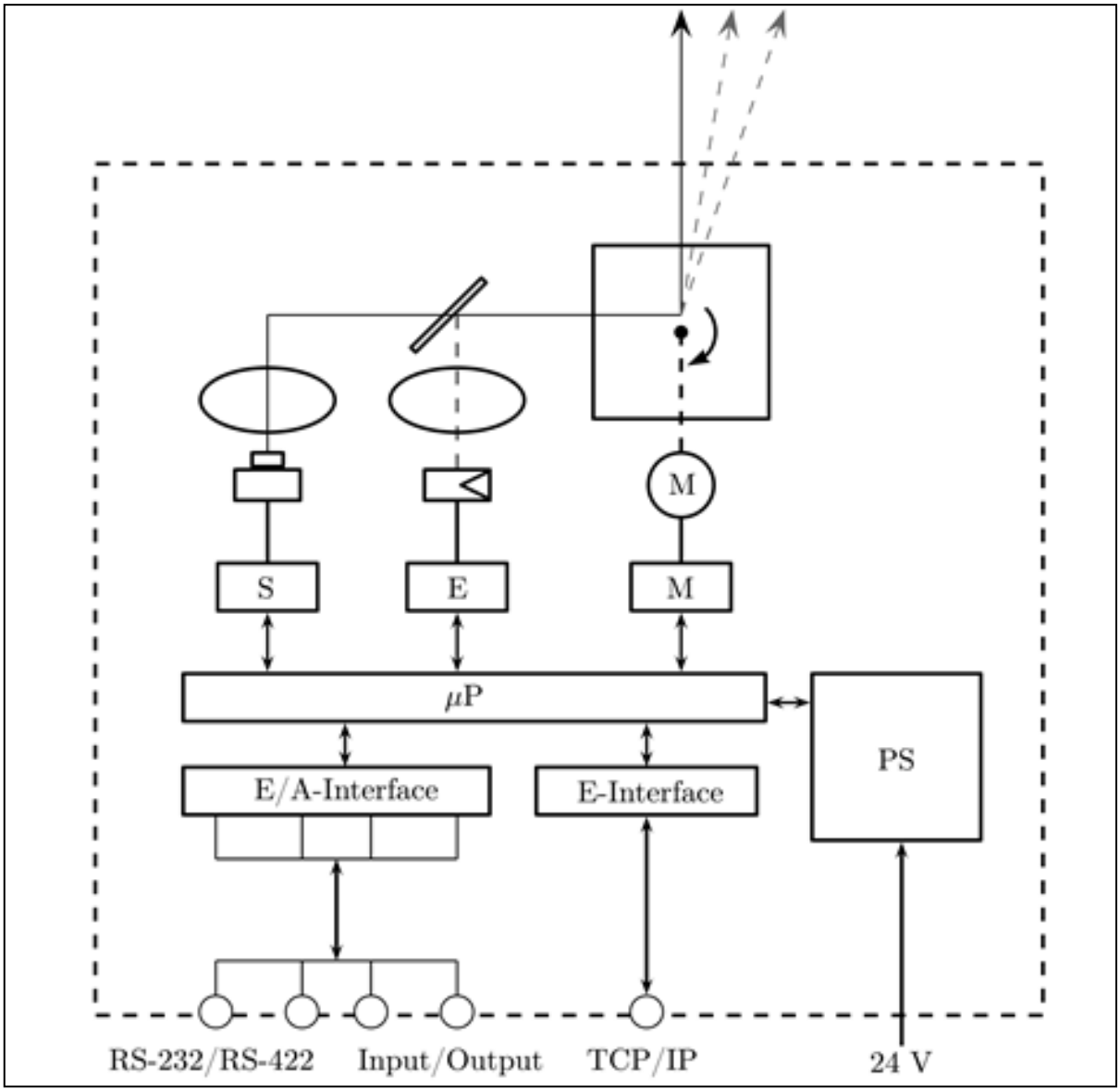


Abbildung 3-22 Aufbau eines zweidimensionalen Lasermesssystems

Der Laserscanner liefert fortlaufend Distanz- und Winkelwerte (r_t, φ_t) aus einem 360°-Umkreis. Der gültige Distanzmessbereich beträgt dabei 0,5 m < r < 250 m. Der NAV350 identifiziert anhand der erhöhten Remissionswerte automatisch Reflektoren in einem Radius r < 70 m und fasst mehrere Distanz- und Winkelwerte, die auf einem Reflektor gemessen werden, zu einem Wertepaar (r,φ) zusammen. Diese Funktionalität ermöglicht die einfache Verwendung von Reflektordaten und erspart damit weitere Berechnungen.

3.5.1. Funktionsprinzip des NAV350

Das Funktionsprinzip des NAV350 entspricht dem in Abbildung 3-21 gezeigten Verfahren, bei dem das Strahlenbündel durch einen drehbar gelagerten Spiegel in einem 360°-Umkreis abgelenkt wird. Weiterhin setzt das Gerät zur Distanzbestimmung die Pulslaufzeitmessung ein. Sowohl die Messung der Remission als auch das Aussenden eines Laserpulses wird jeweils nach einem Winkelschritt von $\Delta\varphi = 0{,}25°$ ausgelöst. Der drehbare Spiegel rotiert mit einer Frequenz von f = 8 Hz. Damit wird in $\Delta t = 0{,}125\ s$ eine Menge von n = $^{360°}/_{0{,}25°}$ = 1440 Messwertpaare (r_t, φ_t) erzeugt.

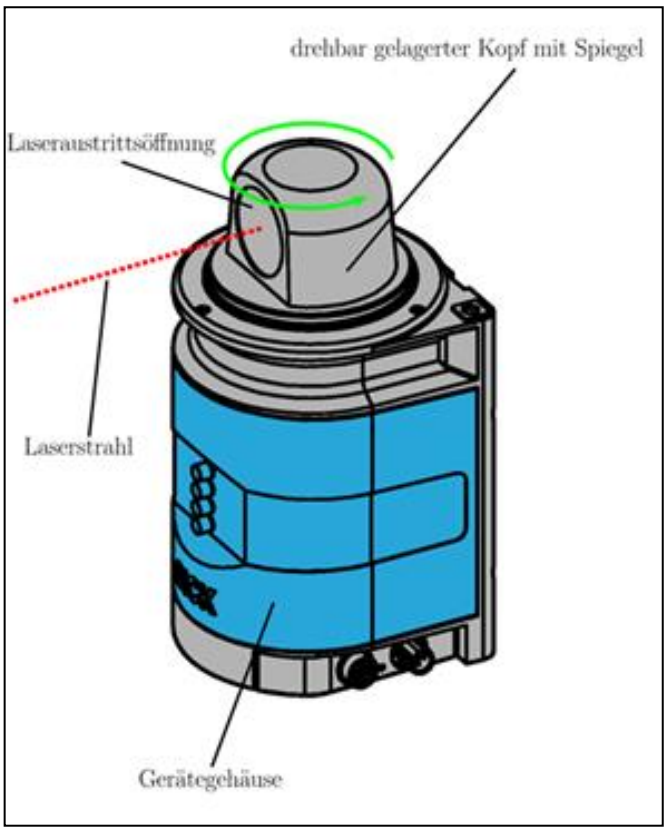


Abbildung 3-23 Funktionsprinzip des NAV350 [14]

Abbildung 3-23 zeigt den Aufbau eines NAV350 und dient zur Veranschaulichung des Funktionsprinzips.

3.5.2. Technische Daten des NAV350

Alle technischen Daten des NAV350 werden in diesem Unterkapitel in der Tabelle 3-2aufgelistet. Weiterführende Informationen zum NAV350 enthält die Betriebsanleitung (siehe [14]).

Tabelle 3-2 Technische Daten des NAV350

Eigenschaft	**Wert**
Allgemeine Daten	
Strahldivergenz	$\theta_{div} = 5\ mrad$
Wellenlänge der Laserdiode	$\lambda = 905\ nm$ [Rotlicht]
Pulsfrequenz	$f_{max} = 14{,}4\ kHz$ $\bar{f} = 12\ kHz$ über $\Delta\varphi = 360°$
Laserklasse des Gerätes	Klasse 1 gemäß EN 60825-1 (1994 + A11, 1996 + A2, 2001), augensicher
Reflektorerkennung	
Distanzbereich	$0{,}5\ m < r < 70\ m$
Winkelbereich	$0° \leq \varphi < 360°$
Winkelauflösung	$\Delta\varphi = 0{,}25°$
Systematischer Fehler der Entfernungsmessung	$e_r = \pm 10\ mm$ bei $\vartheta = 25°C$
Statistischer Fehler der Entfernungsmessung	$\sigma_r = 10\ mm\ (1\sigma)$
Systematischer Fehler der Winkelmessung	$e_\varphi = \pm 0{,}1°$ bei $r < 10\ mm$ $e_\varphi \geq \pm 0{,}25°$ bei $r > 30\ mm$
Statistischer Fehler der Winkelmessung	$\sigma_\varphi = 0{,}05°\ (1\sigma)$

3.6. Fahrerlose Transportfahrzeuge

Fahrerlose Transportfahrzeuge sind Roboter, welche nicht stationär an eine Position gebunden sind – sie können sich im Rahmen ihrer Konstruktionsweise als Fahrzeug frei bewegen. Im Gegensatz zu einem ferngesteuerten Roboter sind autonome Fahrzeuge in der Lage, sich selbstständig von einem Punkt A zu einem Punkt B zu bewegen, sofern diese Verbindung möglich ist. Dabei sind die Probleme der Routenplanung, der Kollisionsvermeidung, der Positionsbestimmung sowie der Kartierung unbekannter Umgebungen Gegenstand aktueller Forschung [32].

Abbildung 3-24 zeigt exemplarisch, wie die Integration mehrerer Teilsysteme zu einem fahrerlosen Transportsystem gestaltet wird. Die Komponenten, welche hier integriert werden, sind das Fahrzeug selbst, die Sensorik, welche zur Erfassung von Landmarken dient, sowie die Sensorik, welche zur Erfüllung von Sicherheitsaspekten dient, Bedienelemente, die zum Eingriff des Anwenders vorgesehen sind, und ein Leitrechner, welcher die Signale der Sensoren und Aktoren zusammenführt und daraus letztendlich ein Gesamtsystem werden lässt.

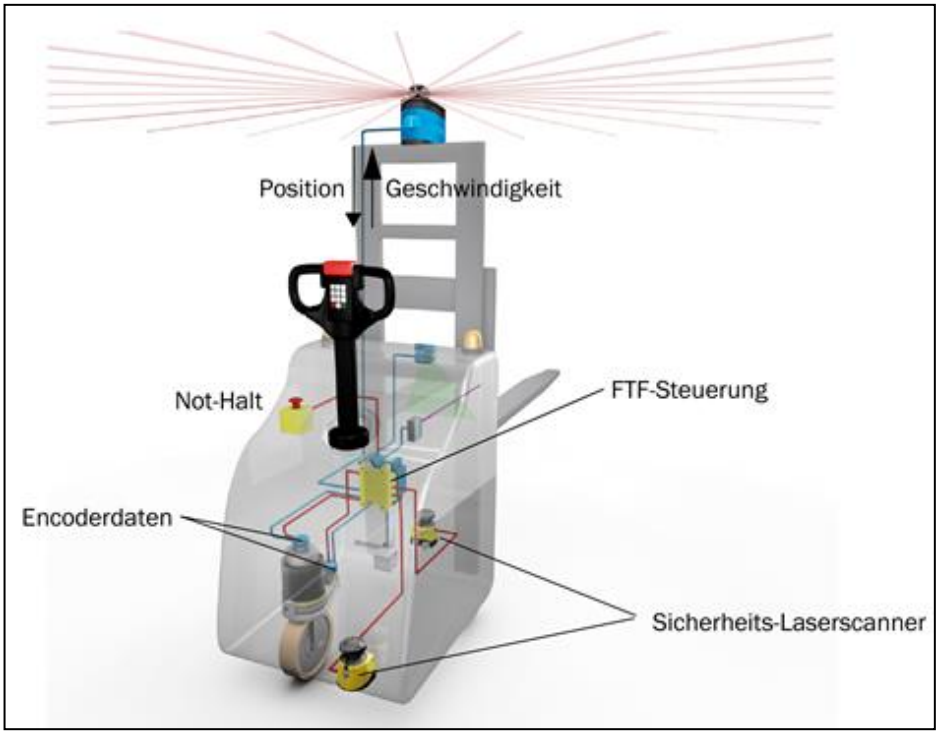


Abbildung 3-24 Fahrerloses Transportfahrzeug

4. Systemanforderungen und Architekturdesign

In Kapitel 4 werden alle Systemanforderung an den Positioniertisch erarbeitet. Ein grobes Architekturdesign des mechanischen und elektrischen Aufbaus wird anschießend bezogen auf die aufgestellten Systemanforderungen erstellt.

4.1. Systemanforderungen

Anforderungen an Man-Maschine-Interface (MMI)

Die gewünschte Trajektorie, die zu verfahren ist, kann über Auswahlkasten ausgewählt werden. Die Visualisierung der Messdaten – wie die Abtastpunkte, Trajektorie, Geschwindigkeit, Beschleunigung, Drehgeschwindigkeit und die Drehbeschleunigung – kann mithilfe der 2D-Diagramme dargestellt werden.

Das MMI soll möglichst flexibel gegenüber der Trajektorien-Planung entwickelt werden. Der Endanwender soll in der Lage sein, die Trajektorien-Parameter über MMI definieren.

Risikoanforderungen

Die Anlage soll abgesichert sein, damit alle Verletzungsgefahren gegenüber dem Anwender ausgeschlossen sind. Aus diesem Grund ist mindestens eine CE-Kennzeichnung für die Anlage erforderlich.

Anforderungen an die Schnittstellen

Der prinzipielle Aufbau des Gesamtsystems ist in Abbildung 4-1 dargestellt. Die IST-Position vom Positioniertisch sowie die Messdaten vom Laserscanner werden auf einen industriellen PC (ECU) übertragen. Der NAV350 ist mit dem Computer über Ethernet verbunden und übergibt in konstanten Zeitabständen von 125 ms seine globale Position und Orientierung an den Computer. Die ECU und Laserscanner sind über ein digitales „Sync“-Signal synchronisiert.

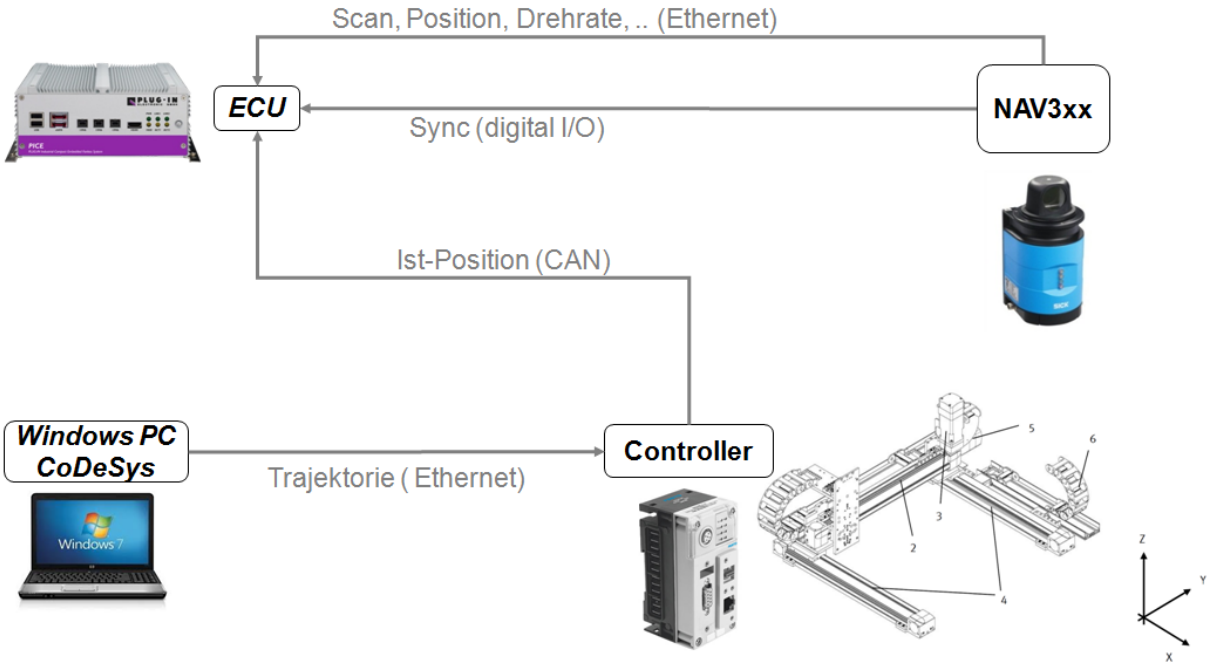


Abbildung 4-1 Prinzipieller Aufbau

Die zu verfahrene Trajektorie soll beispielweise über einen Windows PC erstellt und an den Motorkontroller der einzelnen Achsen übergeben werden. Der Positioniertisch schickt seine aktuelle Position im Echtzeitbetrieb über CAN-Bus an die ECU.

Anforderungen an die Konstruktion

Der Arbeitsraum der Anlage soll die geometrischen Maßen annehmen, sodass eine Trajektorie einer gewöhnlichen AGV-Kurve stattfinden werden kann. Die Anlage kann in Form eines Portalroboters konstruiert werden. Der Portalroboter soll mit zwei Schub- sowie einem Drehaktuator ausgerüstet werden. Die Aktuatoren ermöglichen die Linearbewegung in X- bzw. in Y-Richtung und Rotationsbewegung um die Z-Achse. Die Abbildung 4-2 zeigt die Skizze des Portalroboters mit den gekennzeichneten Hauptkomponenten.

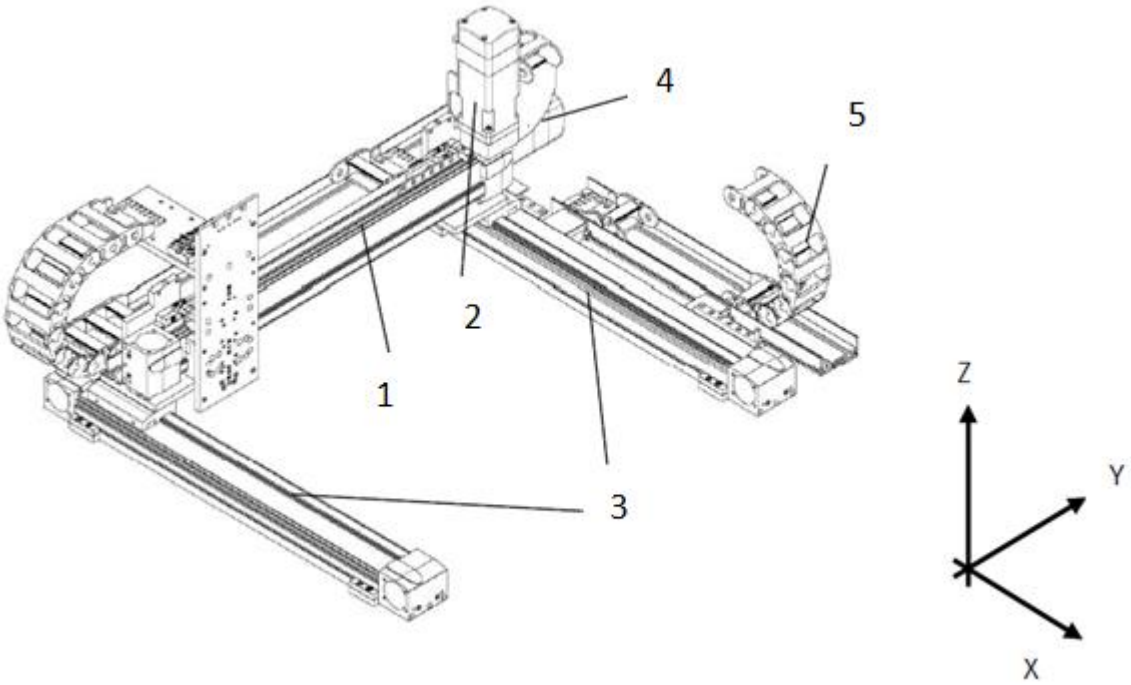


Abbildung 4-2 Positioniertisch in Form eines Portalroboters

Tabelle 4-1 Hauptkomponenten des Positioniertisches

Hauptkomponenten	
1	Y-Achse: Linearachse
2	Antriebspaket: Servomotor
3	X-Achse: Linearachse
4	Antriebspaket: Servomotor
5	Komponenten für die Energieführung

Breite: 6-9 m

Länge: 4-9 m

Höhe: 2 m

Hohe Vibrationen entstehen durch die dynamischen Bewegungen der Achsen. Die komplette Konstruktion soll robust gegen die Vibrationen sein. Wenn Vibrationen oder Materialverformungen der Achse entstehen, soll dies sicher detektiert werden.

Anforderungen an die Antriebe

Absolute Genauigkeit der geradlinigen Antriebe:	<1mm
Absolute Genauigkeit der Rotationsantriebe:	<0.1°
Mögliche Geschwindigkeit:	>3m/s
Mögliche Drehgeschwindigkeit:	720°/s
Mögliche Beschleunigung:	>5m/s²
Mögliche Drehbeschleunigung:	120°/s²

Der Zahnriemen verformt sich während der Bewegung der Achse. Die Elastizitätsverformung des Zahnriemens ändert sich überdies in Abhängigkeit der Position des zu bewegenden Objekts. Je weiter das zu bewegende Objekt vom antreibenden Motor entfernt ist, desto höher ist die elastische Verformung des Zahnriemens. Die dynamischen Parameter, wie Geschwindigkeit und Beschleunigung, spielen an dieser Stelle ebenso eine große Rolle. Die Verformung trägt den Anteil der Ungenauigkeit in die Messung der notwendigen Daten ein. Es kann anschließend zum Drift der Motoren während der Messung kommen. Der Drift und die Verformungen sollen an dieser Stelle auch in Betracht gezogen werden. Der Einfluss der Größen auf das Gesamtsystem kann durch ein zusätzliches Messungssystem außerhalb der Aktuatoren oder Controller analysiert werden. Das zusätzliche Messungssystem kann in Form eines einfachen Lineals oder mit zusätzlichen Sensoren an die Achsen angebracht werden. Die abgelesenen Messwerte des zusätzlichen Messungssystems werden mit den Messwerten des gesamten Systems verglichen. Eine genaue Aussage über den beschriebenen Einfluss kann im nächsten Schritt getätigt werden.

Anforderungen an die Steuerung

Die verwendete Steuerung soll dem Anwender die Möglichkeit geben, auf einfache Weise komplexe und präzise Trajektorien zu definieren. Die Verarbeitungszeit der Steuerung darf nicht die Genauigkeit des Positioniertisches beeinträchtigen. Da mehrere Antriebe synchron angesteuert werden, ist es notwendig, dass die Lagesollwerte absolut zeitsynchron ausgegeben werden. Weiterhin muss die Steuerung in der Lage sein, umfangreiche Daten lokal zu speichern bzw. vom übergeordneten System zu laden.

4.2. Architekturdesign

Es gibt nach dem aktuellen Stand der Technik zwei Möglichkeiten, um das Architekturdesign des Positioniertisches zu realisieren. Die beiden Arten einer möglichen Realisierung sind in diesem Kapitel beschrieben. Zum Schluss werden die Lösungen verglichen und anhand der erfüllten Anforderung wird entschieden, wie das Design des Positioniertisches gestaltet sein soll.

4.2.1. Modulare Portalroboter und Linearroboter

Linearroboter bestehen aus einer seriellen, üblicherweise rechtwinklig verketteten Anordnung von mindestens drei Linearachsen. Am weitesten verbreitet sind modulare Portalroboter als Flächenportal, Linienportal oder Kragarmroboter. Dazu zählen auch Portal- und Auslegerlagen als Komponenten zum Aufbau von Mehrachs-Linearsystemen. Grundeinheit der Portalsysteme sind angetriebene Linearachsen mit verfahrbaren Schlitten, an denen weitere Achsen, Greifer oder Werkzeuge befestigt werden. Der Schlittenantrieb erfolgt typischerweise durch Elektromotoren mit Kraftübertragung durch Zahnriemen, Zahnstangen oder Spindeln, auch Linearmotor-Direktantriebe oder kolbenstangenlose Pneumatikantriebe finden Einsatz.

Bei Portalkonstruktionen wird die Y-Achse von zwei parallelen Führungsachsen in X-Richtung verfahren. Portalroboter ermöglichen anwendungsspezifisch eine Beweglichkeit in XZ- oder XYZ-Richtung und können zusätzliche Drehachsen sowie Greifer umfassen. Als vertikale Achsen gelangen Ausleger- und Teleskopachsen zur Anwendung. Im Gegensatz zu Pick-and-Place-Einheiten oder Gelenkarmrobotern lassen sich mit modularen Portalrobotern bauartbedingt sehr große Verfahrwege und Arbeitsräume herstellen und ebenso hohe Lasten handhaben. Linearroboter bestehen aus einzelnen Führungsachsen oder Kragachsen ohne Unterstützung und eignen sich deswegen lediglich für kleinere Arbeitsräume.[5]

Abbildung 4-3 Modularer Portalroboter (Quelle http://img.directindustry.de)

Der Portalroboter(Abbildung 4-3) würde teilweise die Konstruktionsanforderung des Positioniertisches erfüllen.

[5]http://www.xpertgate.de/produkte/Portalroboter.html

4.2.2. SCARA Roboter

Der Selective Compliance Assembly Roboter Arm, auch SCARA Roboter genannt, ist ein besonderer Typ Industrieroboter, dessen Arbeitsraum in der horizontalen Ebene definiert ist. Ein SCARA-Roboter besitzt in der Regel vier Achsen sowie vier Freiheitsgrade. Sämtliche Achsen sind als serielle Kinematik ausgeführt, d.h. der Koordinatenursprung der folgenden Achse ist abhängig von der Position der vorhergehenden. Bei einem SCARA-Roboter sind die erste und zweite Achse rotatorischer Natur, die dritte und die vierte Achse sind vielfach aus einem Bauelement hergestellt (der Kugelrollspindel) und erlauben eine rotatorische sowie eine Linearbewegung. Das Werkzeug des Roboters wird am unteren Ende der Z-Achse montiert.

SCARA-Roboter gibt es in unterschiedlichen Größen – die Reichweite der am weitesten verbreiteten Modelle kann zwischen 100 mm und 1.200 mm liegen. Roboter dieses Typs können Nutzlasten im Bereich von 1 kg bis zu 200 kg handhaben.

Der Arbeitsbereich eines solchen Roboters ist aufgrund seiner Armgeometrie typischerweise nierenförmig. Die konkreten Abmessungen hängen dabei natürlich von den konkreten Randbedingungen wie Armlängen, Winkel etc. ab.[6]

Abbildung 4-4 SCARA Roboter (Quelle http://img.directindustry.de)

Der SCARA Roboter (Abbildung 4-4) kann eingesetzt werden, um verschiedene Trajektorien zu verfahren. Einen Nachteil des Roboters stellt der Arbeitsraum dar. Die Arbeitsfläche deckt nicht die kompletten geforderten Maße ab.

[6] http://de.wikipedia.org/wiki/SCARA-Roboter

4.2.3. Bewertung und Auswahl

Um eine richtige Auswahl des Designs zu treffen, sollen die essenziellsten Anforderungen erfüllt sein – dabei handelt es sich um:

- Dynamische Anforderungen
  - Geschwindigkeit
  - Beschleunigung
- Arbeitsraum: Die möglichen geometrischen Maße der Anlage sollen es dem Anwender möglich machen, eine Kure des realen AGV zu simulieren.
- Die Genauigkeit der Anlage soll den Anforderungen entsprechen

Tabelle 4-2 zeigt, welches Design am besten den gestellten Anforderungen entspricht.

Tabelle 4-2 Vergleich der beiden Designmöglichkeiten

Design / Anforderung	SCARA	Portalroboter
Dynamik	erfüllt	erfüllt
Arbeitsraum	teilweise erfüllt	erfüllt
Genauigkeit	erfüllt	erfüllt

Der SCARA und Portal Roboter sind in der Lage, die notwendige Dynamik und Genauigkeit erfüllen. Leider kann der SCARA Roboter nicht den gewünschten Arbeitsraum abdecken, deshalb wird das Design nicht für Realisierung des Positioniertisches gewählt. Das Design des Portalroboters entspricht hingegen allen gestellten Anforderungen und kommt bei der Realisierung zur Anwendung.

5. Auswahl der Komponenten

Nachdem genaue Spezifikation der Einrichtung sowie das grobe Design des Positioniertisches definiert sind, kann man im nächsten Schritt mit dem detaillierten Design des gesamten Systems beginnen. Das detaillierte Design des Systems beinhaltet mehrere Abschnitte:

- Schlittenantrieb
- Sicherheitskonzept
- Steuerungskonzept

5.1. Der Schlittenantrieb

Der Schlittenantrieb erfolgt typischerweise durch Elektromotoren. Im Kapitel werden die essenziellsten Schlittenantriebe mit deren Eigenschaften und Funktionsprinzipien beschrieben. Eine Auswahl wird anschließend getroffen.

Die am häufigsten verwendeten Antriebe sind Zahnriemen-, Spindel- und Linearmotor-Direktantrieb. Die Antriebe können durch ihre Eigenschaften bei der Realisierung des mechanischen Aufbaus vom Positioniertisch eingesetzt werden.

5.1.1. Zahnriemen

Die Bewegungsübertragung bei dem Zahnriemenantrieb erfolgt ausschließlich über einen Zahnriemen. Der Antriebsmotor dreht eine Welle und die Welle überträgt die Drehbewegung des Motors mit oder ohne Getriebe an den Zahnriemen. Der am Zahnriemen befestigte Schlitten bewegt sich folglich längs seiner Bewegungsachse. In Abbildung 5-1 ist der Funktionsschnitt des Zahnriemenantriebes grafisch dargestellt.

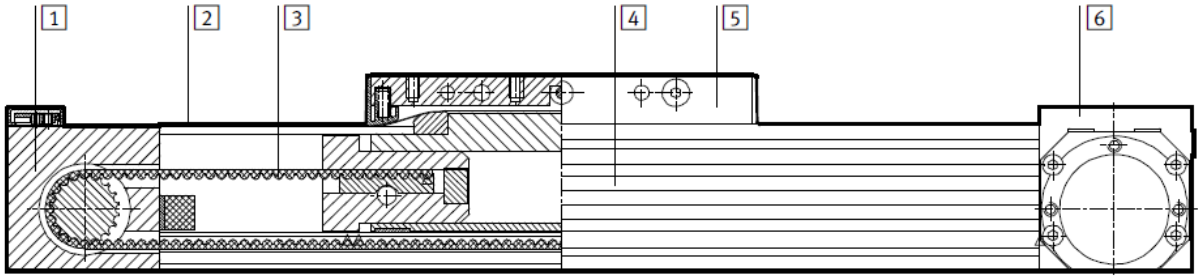


Abbildung 5-1 Funktionsschnitt des Zahnriemenantriebes

1. Umlenkungsgehäuse
2. Abdeckband
3. Zahnriemen
4. Profil
5. Schlitten
6. Antriebsgehäuse

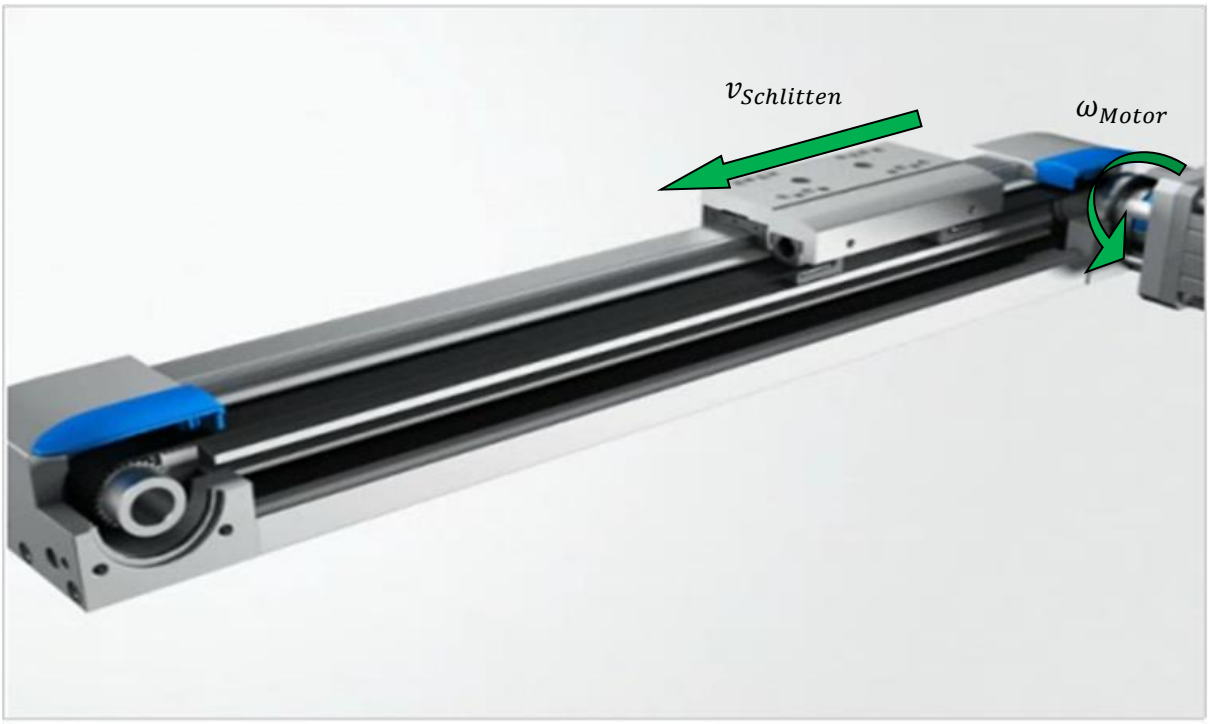


Abbildung 5-2 Bewegung des Schlittens (Zahnriemenantrieb)

Die Bewegung des Schlittens in Abhängigkeit von Motordrehung ist in Abbildung 5-2 ersichtlich. Die dynamischen und geometrischen Eigenschaften des Zahnriemenantriebes entsprechen teileweise den gestellten Anforderungen.

5.1.2. Spindel

Die Bewegung des Spindelantriebes erfolgt über die Drehung der Spindel. Die Drehbewegung des antreibenden Motors wird im Vergleich zum Zahnriemenantrieb direkt auf die Spindel übertragen. Die Übertragung der rotatorischen Bewegung erhöht die Genauigkeit des Antriebes. Abbildung 5-3 zeigt den Funktionsschnitt des Spindelantriebes grafisch.

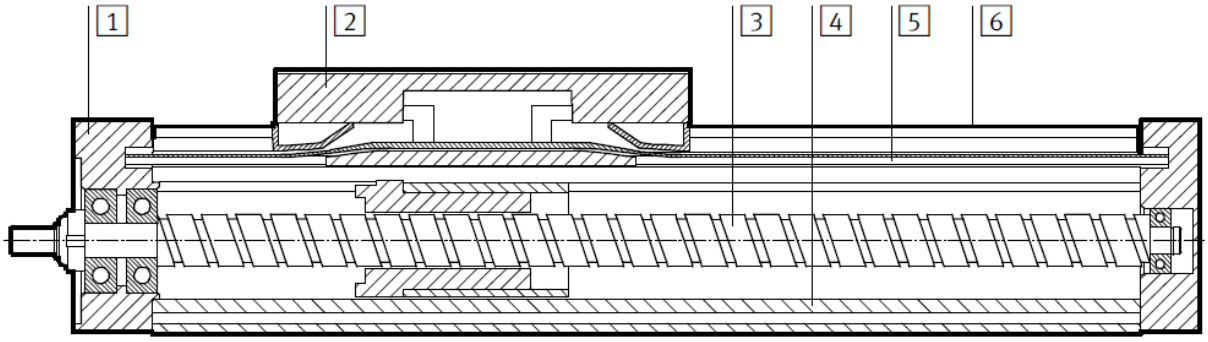


Abbildung 5-3 Funktionsschnitt des Spindelantriebes

1. Abschlussdeckel
2. Schlitten
3. Spindel
4. Profil
5. Abdeckband
6. Führungsschiene

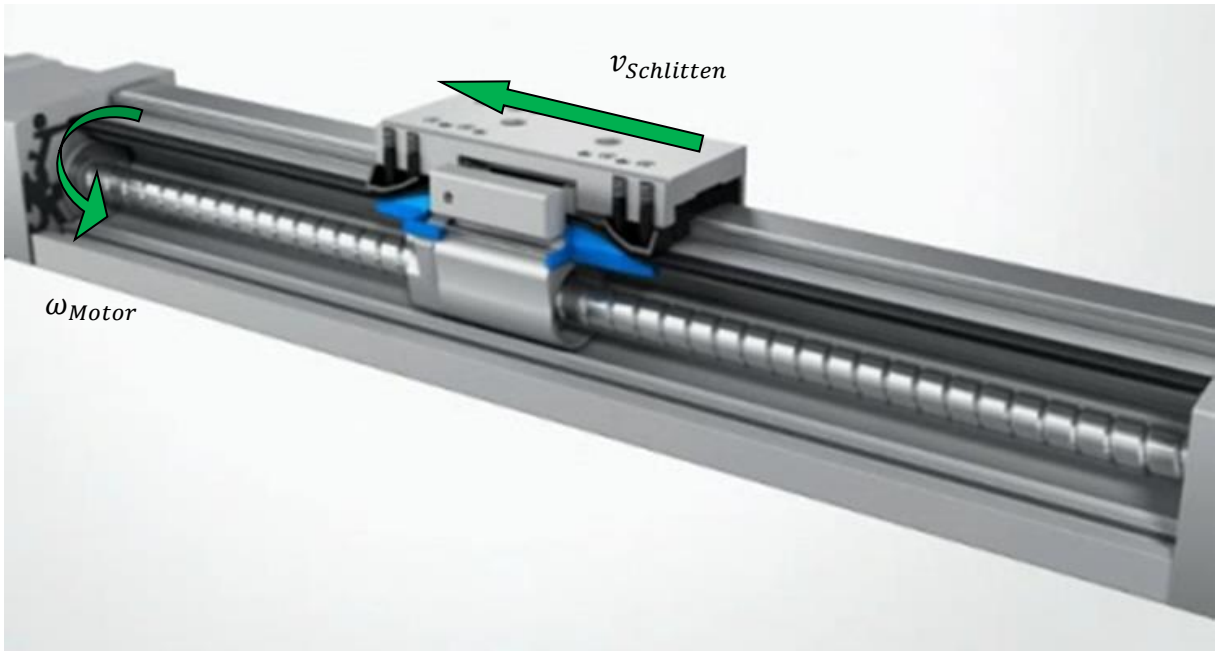


Abbildung 5-4 Bewegung des Schlittens (Spindelantrieb)

Die Bewegung des Schlittens in Abhängigkeit von der Motordrehung ist in Abbildung 5-2 zu sehen. Die dynamischen und geometrischen Eigenschaften des Spindelantriebes entsprechen teilweise den gestellten Anforderungen.

5.1.3. Linearmotor-Direktantrieb

Die Bewegung des Schlittens erfolgt durch die direkte Kraftübertragung ohne mechanische Übertagungselemente. Zur Ermittlung der aktuellen Position des Schlittens wird standardmäßig ein zusätzliches Weg-Messsystem verwendet, welches in der Achse integriert ist.

Abbildung 5-5 Lineare Achse mit einem Direktantrieb (Quelle: www.schunk.com)

Durch die direkte Antriebstechnik kann eine sehr hohe Genauigkeit auch bei hoher Dynamik erreicht werden. Der Linearmotor-Direktantrieb entspricht den gestellten Anforderungen.

5.1.4. Bewertung und Auswahl

Alle beschriebenen Antriebe entsprechen den aufgestellten Anforderungen. Um eine Auswahl zu treffen, sollen die Eigenschaften genauer verglichen werden. Tabelle 5-1 zeigt den Vergleich.

Tabelle 5-1 Vergleich von Linearantrieben

Anforderung \ Antrieb	Zahnriemen	Spindel	Direktantrieb
Arbeitshub	++	+	++
Geschwindigkeit	++	+	++
Beschleunigung	+	+	++
Genauigkeit	+	++	++
Wartung	++	++	+
Komplexität der Implementierung	++	++	+
Preis	+	+	-

Der Arbeitshub des Spindelantriebes ist im Vergleich zum Zahnriemen- oder Direktantrieb deutlich kürzer. Der maximal mögliche Hub beträgt nach der Erforschung der Lieferanten drei Meter bei geforderten acht Metern. Die dynamischen Anforderungen sowie Geschwindigkeit und Beschleunigung erfüllen wieder die Direkt- und Zahnriemenantriebe. Der dritte Typ an der Stelle erreicht Beschleunigungen bis zu 80 m/s². Die Spindel- sowei Direktanrieb sind durch eine direkte Kraftübertagung mit einer sehr hohen Genauigkeit von ±10µm gekennzeichnet. Der Direktantrieb ist wesentlich anspruchsvoller im Vergleich zu anderen Kandidaten, was die Wartung und Komplexität der Implementierung betrifft. Der Preis liegt bei den meisten Antrieben im mittleren Bereich, außer dem Direktantrieb. Der Preis vom Direktantrieb fällt im Vergleich des Preis-Leistung-Verhältnis höher als anderen aus.

Der Zahnriemenantrieb entspricht allen gestellten Anforderungen. Die absolute Position des Schlittens wird von zwei Messsystemen erfasst: vom Inkrementalgeber des Motors und vom internen Mess-Weg-System. Der Antrieb findet zur Realisierung des mechanischen Aufbaus Verwendung.

5.2. Sicherheitskonzept

Das Sicherheitskonzept ist einer der wichtigen Punkte, die bei der Konzipierung einer Anlage betrachtet werden. Der Anwender darf unter keinen Umständen verletzt werden. Der Positioniertisch soll aus diesem Grund sorgfältig abgesichert werden. Im Kapitel werden drei Sicherheitskonzepte entwickelt.

5.2.1. Anforderungen

Für die Anlage ist ein Platz in einem ca. 25 m x 12 m großen Raum vorgesehen. Die markierte Fläche in Abbildung 5-6 ist für die Anlage geplant.

Abbildung 5-6 Testhalle

Wenn die Anlage angeschaltet, aber nicht im Betrieb ist (keine Bewegung der Achsen), soll die Anlage auch merken, dass der Anwender die abgesicherte Fläche betreten hat und anschließend die betretene Fläche verlassen hat.

Sensoraufnahme:

Die Anlage darf nicht starten, bevor der NAV350 an der Aufnahme befestigt ist. Wenn der NAV350 sich während der Fahrt von der Aufnahme löst, soll die Fahrt unmittelbar gestoppt werden. Des Weiteren soll der Sensor eventuell mit einer Sicherheitskette an der Anlage befestigt sein. Die Sicherheitskette soll sicherstellen, dass, wenn der Sensor sich von der Aufnahme löst, der Anwender trotzdem in Sicherheit bleibt.

Der auf dem Positioniertisch montierte NAV350 soll den gesamten Raum abscannen können. Das heißt die Scanhöhe beträgt ca. 2.2 m.

5.2.2. Sicherheitskonzept 1

Damit niemand während der Testfahrt verletzt wird, soll die Fläche mit ca. 1.80-2 m hohen Gitterzäunen abgetrennt werden. Um die Wartung der Anlage und Sensorfixierung durchführen zu können, soll in einer der Abtrennungen eine abgesicherte Tür platziert werden. Die Anlage darf nicht starten, solange die Tür geöffnet ist. Wenn die Tür während der Fahrt geöffnet wird, soll die Fahrt unmittelbar gestoppt werden.

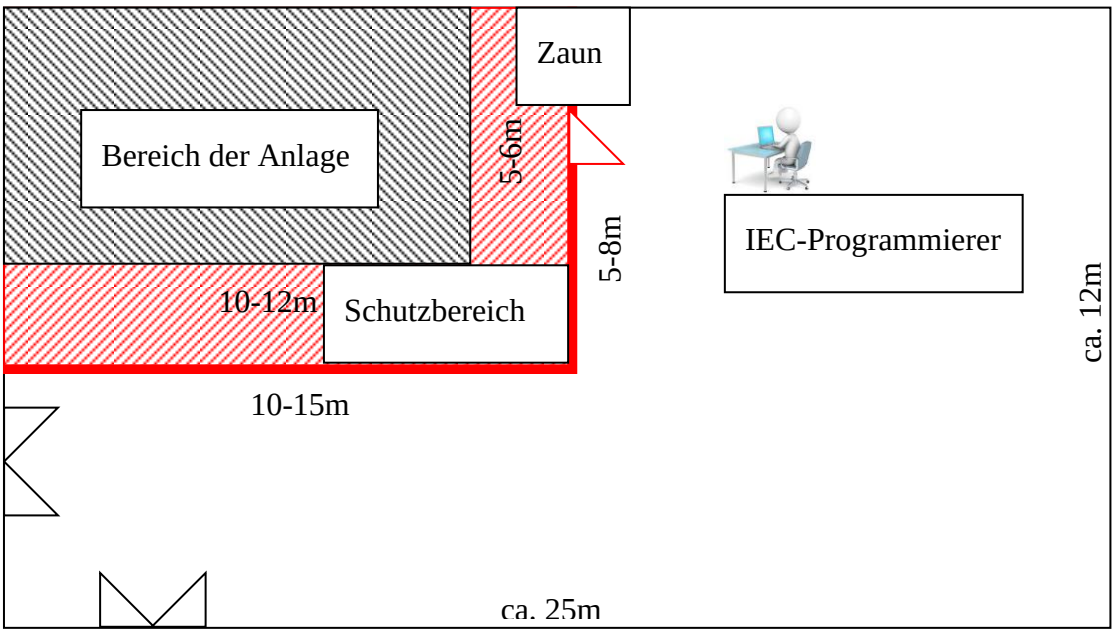


Abbildung 5-7 Eine Skizze des Sicherheitskonzeptes 1

Die Realisierung des Konzeptes ist recht einfach und die Kosten sind niedrig. Man benötigt eine sensorüberwachte Abzäunung der Anlage. Die Sensoren können in Form der einfachen Init-Sensoren (Induktivsensoren) ausgestaltet sein.

5.2.3. Sicherheitskonzept 2

Um wiederum für eine 100%ige Sicherheit zu sorgen, sollen um die Anlage die SICK Laserscanner (z.B. S3000Advanced) platziert werden. Die Scanner sollen die komplette Fläche um der Anlage abscannen, wenn der Sicherheitsabstand betreten wird, soll die Anlage unmittelbar abgeschaltet werden. Wenn der Sicherheitsabstand während der Fahrt betreten wird, soll die Anlage unmittelbar gestoppt werden.

Man kann mit SICK Laserscanner eine Sequenzerkennung realisieren. Die Sequenzerkennung ermöglicht eine sichere Detektion, ob eine Person den Gefahrbereich betreten oder verlassen hat. Der Laserscanner merkt die Reihenfolge mit der die Sicherheitsstufen betreten wurden und erwartet, dass die gleichen Sicherheitsstufen in umgekehrter Reihenfolge schalten, somit wird sichergestellt, dass die Person in Sicherheit ist, und die Anlage kann gestartet werden. Wenn die Person sich im Gefahrbereich befindet, kann die Anlage nicht starten. Die Anzahl der Sicherheitsfelder ist vom Typ des Lasers abhängig und variiert von 1-8 Felder. Eine Variante der Realisierung des Sicherheitskonzeptes ist in Abbildung 5-8 dargestellt.

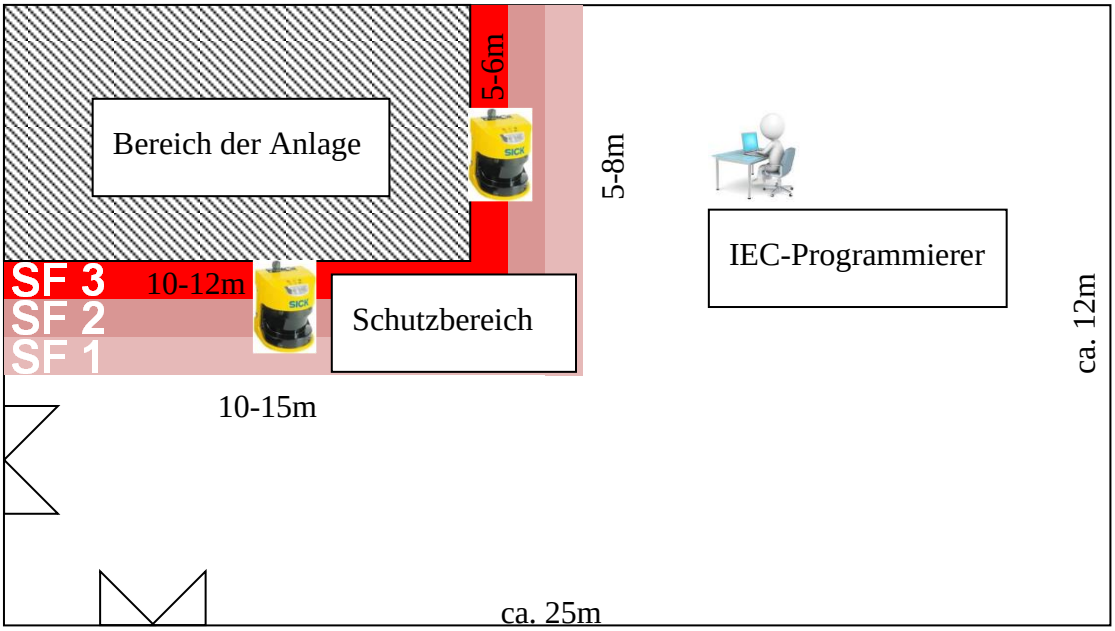


Abbildung 5-8 Eine Skizze des Sicherheitskonzeptes 2

Der Aufbau eines Positioniertisches ist in einer Testhalle geplant. Der Positioniertisch kann einen Laserscanner verfahren sowie drehen. Der Laserscanner vermisst während der Bewegung seine Umgebung und ermittelt anhand von Reflektor-Landmarken seine absolute X/Y Position im Raum.

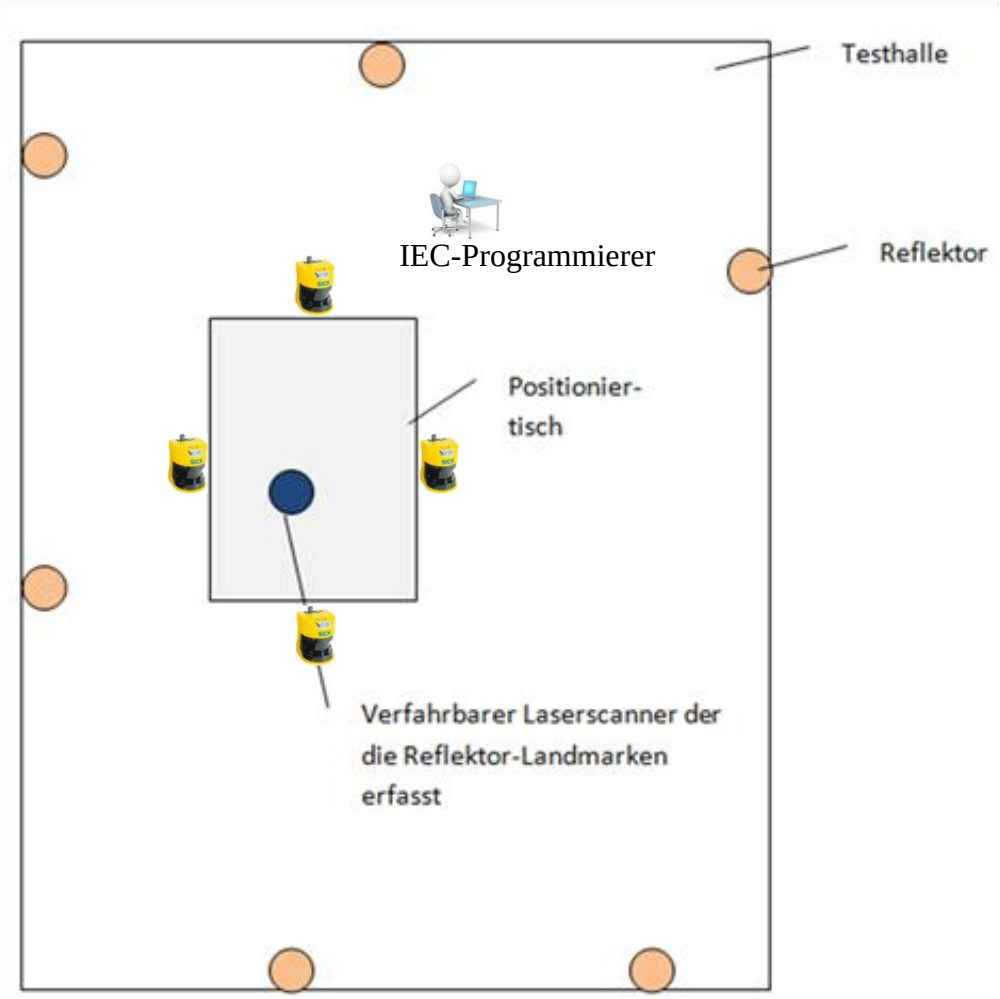


Abbildung 5-9 Weitere Möglichkeit der Position des Positioniertisches in der Testhalle

In Abbildung 5-9 ist eine weitere Möglichkeit der Position des Positioniertisches in der Testhalle dargestellt. In diesem Fall soll die komplette Fläche um die Anlage (360°) mit den SICK Laserscannern abgesichert werden. Die Sicherheitsstufen decken den kompletten Bereich um die Anlage ab.

5.2.4. Sicherheitskonzept 3

Das letzte entwickelte Sicherheitskonzept ist ebenso wie Variante 3 mechanisch aufgebaut. Die Gefahrzone soll mit einer Abzäunung abgetrennt werden; damit niemand während der Testfahrt in der Schutzzone unbemerkt bleibt, wird die Zone mit den Laserscanner abgesichert. Somit merkt die Anlage das Betreten sowie das Verlassen des Gefahrbereichs. Die Abbildung 5-10 zeigt den Sicherheitskonzept grafisch.

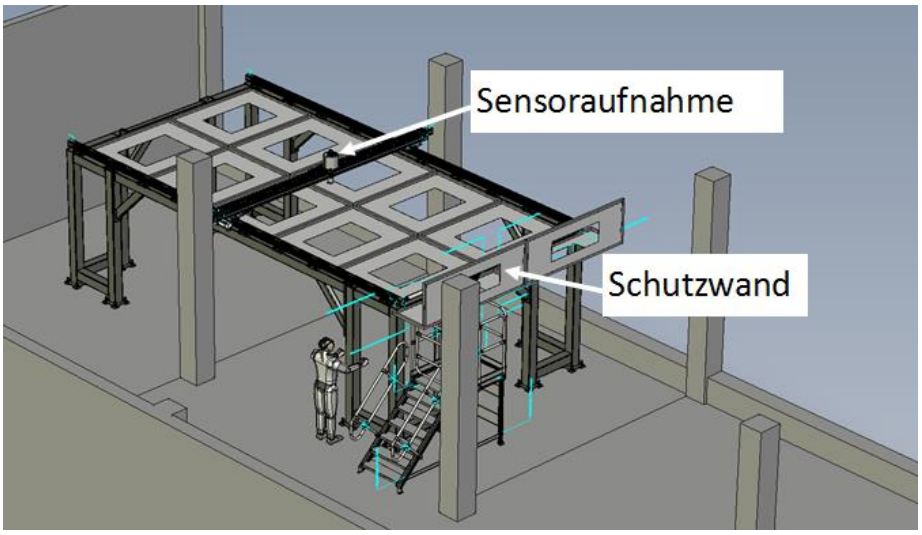


Abbildung 5-10 Sicherheitskonzept 3

Der Positioniertisch soll nach Vorgaben dieses Sicherheitskonzeptes über 2,5m hoch sein und ist mit Schutzwänden an allen vier Seiten der Anlage vorgesehen. Die einseitige Absicherung mit einer Schutzwand wird in Abbildung 5-10 gezeigt.

5.2.5. Bewertung und Auswahl

Der detaillierte Vergleich der entwickelten Sicherheitskonzepte ist in der Tabelle 5-2 aufgeführt.

Tabelle 5-2 Bewertung der Sicherheitskonzepte

Konzept / Anforderung	1	2	3
Risikoreduzierung	+	++	++
Montagekomplexität	+-	+	+-
Programmierung	++	++	++
Sequenzerkennung	--	++	++
Preis	++	+	+

Die Risikoreduzierung der Variante 1 ist im Vergleich zu anderen Sicherheitskonzepte kleiner, da das Detektieren vom Betreten und vom Verlassens des Gefahrbereichs mit einer einfachen Abzäunung nicht möglich ist. Das Sicherheitskonzept 3 wird zur Realisierung gewählt, da es alle Anforderungen erfüllt.

5.3. Steuerungskonzept

Die Steuerung stellt das zentrale Element im zukünftigen Positioniertisch dar. Heutzutage spielen nur noch digitale Steuerungen, die softwaregesteuert arbeiten, eine Rolle. Diese existieren in verschiedenen Ausführungen:

- Speicherprogrammierbare Steuerung (SPS)
- Numerische Steuerung (CNC)

Die beiden Steuerungen werden im Kapitel beschrieben, deren Eigenschaften werden verglichen. Zum Schluss erfolgt die Entscheidung, welche von den beiden Steuerungen bei der Realisierung Anwendung finden soll.

5.3.1. CNC-Steuerung

Die CNC-Steuerung[7] hat als Hauptaufgabe die Führung von mehreren Achsen, z. B. zweidimensional mit x- und y-Achsen oder dreidimensional mit x-, y- und y-Achsen. Der Gesamtablaut der Führung besteht aus einzelnen Verfahrbewegungen mit einer vorgegebenen Geschwindigkeit, die über NC-Sätze beschrieben sind. Ein NC-Satz ist dabei ein Bearbeitungsschritt, der die Relativbewegung im mehrdimensionalen Raum beschreibt. Für die Beschreibung dieser Bearbeitungssätze gelangt die DIN 66025, der sogenannte „G-Code“, zum Einsatz.

Um den Bearbeitungsablauf zu verarbeiten, führt die CNC-Steuerung folgende Funktionen aus:

- Ein Interpreter verarbeitet die einzelnen Bearbeitungssätze und löst interne Programmfunktionen aus.
- Die Geometriedatenverarbeitung berechnet aus den einzelnen Bewegungen unter Berücksichtigung der Kinematik der Maschine (Transformation in die Bewegungsrichtungen der Servoachsen) und realisierbaren Geschwindigkeiten sowie Beschleunigungen Verläufe für die einzelnen Achsen der Maschine.
- Der Interpolator ermittelt aus den Bahnabschnitten Winkellagesollwerte für jede Antriebsachse in einem festen Zeitraster.
- Lagesollwerte werden dann vom elektrischen Antrieb mit seinen Regelungsfunktionen (Lage-, Geschwindigkeits- und Drehmomentregelung) in entsprechende Bewegungen umgesetzt.

Der Programmablauf ist in Abbildung 5-11 grafisch dargestellt.

[7] CNC – „Computerized“-computerbasiert „Numerical“- numerisch „Control“ - Steuerung

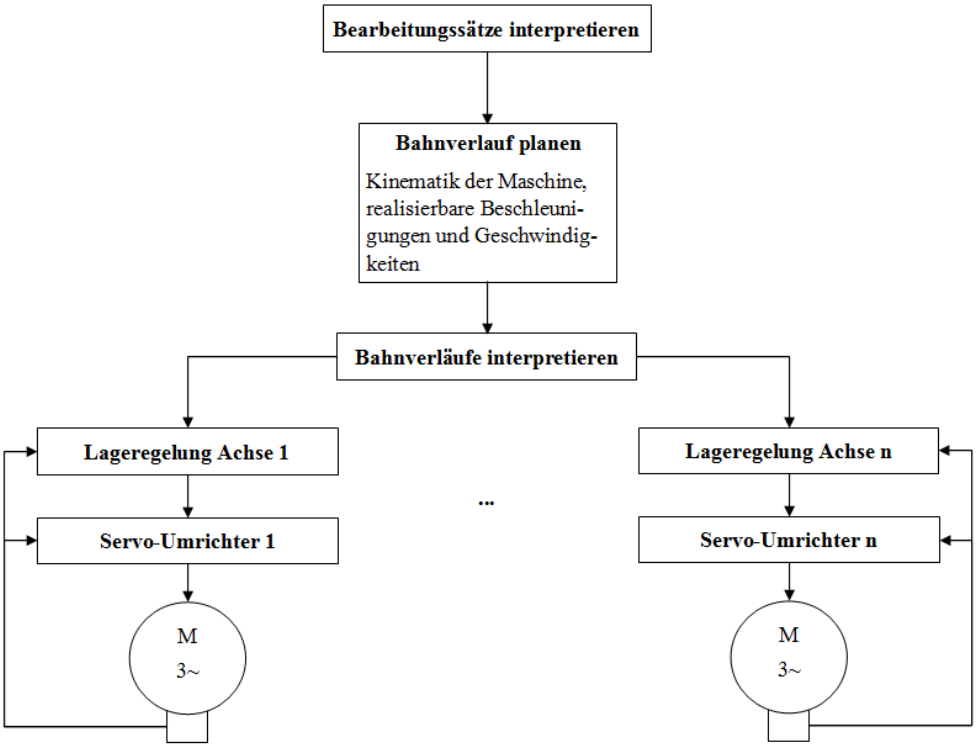


Abbildung 5-11 Programmablauf in einer CNC-Steuerung

Das interne Programm läuft zyklisch durch, um in festen Zeitabständen Ausgangswerte für die folgende Aktuatorik zu liefern. Da mehrere Antriebe synchron angesteuert werden, ist es notwendig, dass die Lagesollwerte absolut zeitsynchron ausgegeben werden.

CNC-Steuerungen erfordern durch die komplexen sowie rechenintensiven Algorithmen für die Geometriedatenverarbeitung und Interpolation eine hohe Rechenleistung. Weiterhin verfügen CNC-Steuerungen in der Regel über eine lokale Bedienung zur Anzeige des Bearbeitungsablaufs.

5.3.2. SoftMotion Konzept unter CoDeSys

Da die Prozesssteuerung, Datenerfassung und Datenverarbeitungsaufgabe (so wie Visualisierung) ausgeführt werden, bietet sich der Einsatz von PC-basierter Steuerung an. Die PC-basierten Steuerungen differenzieren sich in zwei Arten:

PC + SPS-Karte = Slot - SPS

Merkmale: SPS-Steuerung ist unabhängig vom PC-Betriebssystem

Hartes Echtzeitverhalten der SPS

Höhere Ausfallsicherheit

PC + SPS - Software = Soft - SPS

Merkmale: SPS-Steuerung basiert auf Windows-Betriebssystem

Weiches Echtzeitverhalten der SPS

Geringere Ausfallsicherheit

Die beide Arten der PC-basierten Steuerung, sowohl Slot-SPS als auch Soft-SPS, sind in der Lage, ihre Prozessdaten über ein angeschlossenes Feldbussystem (z.B. Industrial Ethernet - TCP/IP) zu erhalten.

SoftMotion ermöglicht das Realisieren von Bewegungen – von einfachen Einachs- über Kurvenscheiben- bis hin zu komplexen Bewegungen in mehreren Dimensionen unter der Entwicklungsumgebung CoDeSys.

Die gesamte Programmlogik wird im SPS-Programm behandelt und lediglich die reine Bewegungsinformation in den Bibliothekbausteinen abgearbeitet. Eine Übersicht über die Bibliothekbausteine des SoftMotion Konzeptes findet sich in Abbildung 5-12.

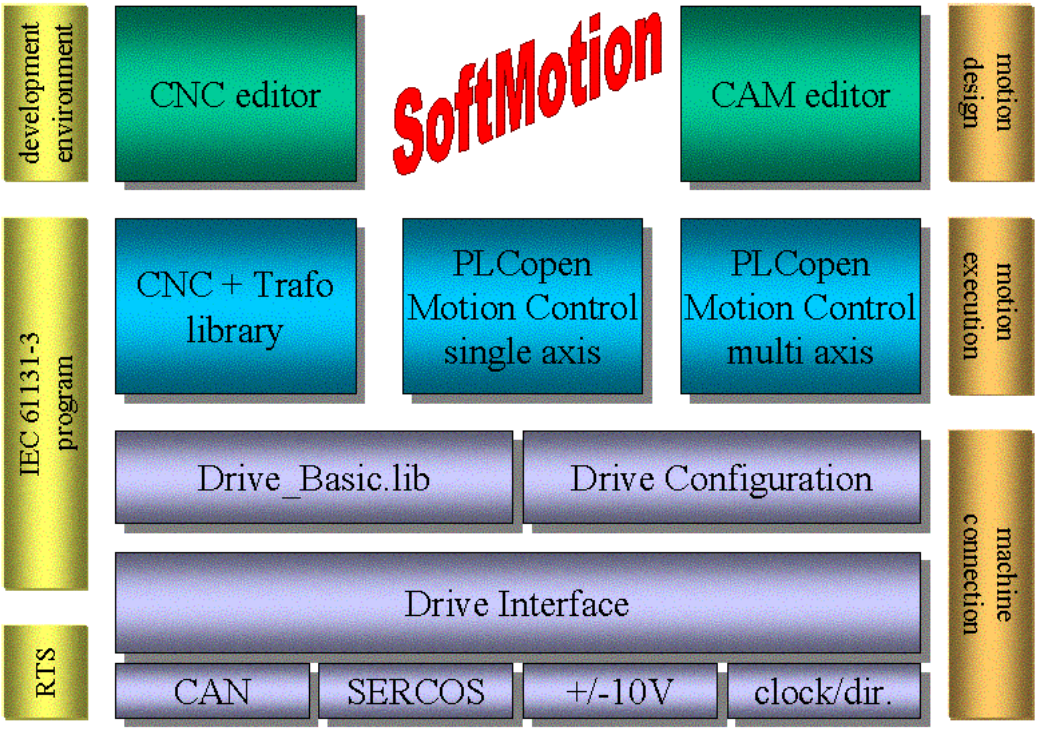


Abbildung 5-12 Übersicht der Komponenten der SoftMotion Bibliothek (Quelle http://de.codesys.com/download)

SoftMotion gliedert sich in folgende Komponenten:

- **Drive Interface**
  Die Komponente ist für Kommunikation zwischen Anwendersteuerungsprogramm und den Antrieben verantwortlich. Sie besteht aus der Bibliothek SM_DrivBasic.lib sowie Antriebs- und Bussystem-spezifischen Bibliotheken und Treibern.
  Der Programmierer bildet im Konfigurationseditor die Struktur und Parametrierung der Antriebshardware ab. CoDeSys erzeugt daraus mithilfe der DriveInterface-Bibliotheken IEC-Datenstrukturen, die die Antriebe abstrahiert darstellen. Das Drive Interface kommuniziert nach der Konfiguration automatsch mit den Antrieben und sorgt für eine laufende Aktualisierung der Antriebsdatenstrukturen und die Übertragung der darin geänderten Daten. Um einen steuernden Einfluss auf die Antriebs-Hardware zu nehmen, kann das Programm auf Datenstrukturen des Antriebes entweder mittels Standardbausteinen aus SoftMotion-Bibliotheken(SM_CNC.lib, SM_PLCOpen.lib) oder mit vom Programmierer selbst erstellten Programmen zugreifen.
- **CNC-Editor**

  Der Editor dient zur grafischen und textuellen Programmierung von mehrdimensionalen Antriebsbewegungen. Für jede erzeugte Bahn wird von CoDeSys automatisch eine globale Datenstruktur (CNC Data) angelegt, die im IEC-Programm zur Verfügung steht.
- **CAM-Editor**

  Der CAM-Editor stellt den in CoDeSys integrierten, grafisch bedienbaren Kurvenscheiben-Editor dar. Er dient der Programmierung von Kurvenscheiben (CAM's) zur Steuerung von mehrachsigen Antrieben. Aus jeder programmierten Kurvenscheibe generiert CoDeSys automatisch eine globale Datenstruktur(CAM Data), die im IEC-Programm zur Verfügung steht.
- **CNC-Bibliotheken**

  Die Bibliotheken SM_CNC.lib, SM_CNCDiagnostic.lib und SM_Trafo.lib offerieren dem Programmierer Bausteine, mit deren Hilfe er die im CNC-Editor erzeugten oder zur Laufzeit geplanten Bewegungen bearbeiten, darstellen oder ausführen kann.
- **PLCopen-Bibliothek**

  Die Bibliothek SM_PLCopen.lib besteht aus den Bausteinen, mit denen die Steuerung einzelne Achsen oder auch die synchronisierte Bewegung zweier Achsen leicht programmiert und realisiert werden kann. Die Bibliothek beinhaltet neben Bausteinen zur Statusabfrage, Parametrierung sowie allgemeinen Bedienung auch die Funktionsblöcke, die eine Achse gemäß vorgegebenem Geschwindigkeits- und Beschleunigungsverhalten bewegen können. Die PLCopen-Komponenten der SoftMotion-Bibliothek sind in der Lage, neben der Steuerung einer Achse auch die Synchronisation zweier Achsen zu realisieren. In diesem Fall dient eine Achse als Master und steuert unter einer Vorschrift eine zweite Achse (Slave). Überdies existieren Funktionsblöcke zur Programmierung elektronischer Getriebe und Phasenverschiebung.

5.3.3. Bewertung und Auswahl

Steuerung / Anforderung	CNC-Steuerung	Speicherprogrammierbare Steuerung
Anzeige des Bearbeitungsablaufs	++	++
Ausgabe von Lagesollwerten zeitsynchron	++	+ (++ mit SoftMotion Bibliothek)
Lokale Speicherung der Daten	++	++
Einfache Erstellung beliebigen Trajektorie	++	++
Preis	-	++ (+ mit SoftMotion Bibliothek)

Die beiden Steuerungen erfüllen alle gestellten Anforderungen. Die SPS beginnt bei ihrer zyklischen Programmausführung den nächsten Zyklus, sobald der vorhergehende Zyklus abgeschlossen wurde. Dabei wird keine Rechenzeit verschenkt, allerdings führen die Unterschiede in der Rechenzeit zu einer asynchronen Ausgabe von Lagesollwerten. Der Zeitablauf der Programmausführung ist nicht vollständig deterministisch. Jedoch kann das Verhalten der Steuerung durch die Verwendung der SoftMotion Bibliothek kompensiert werden. Die Bibliothek stabilisiert die Rechenzeit.

6. Realisierung

6.1. Mechanischer Aufbau

Die praktische Realisierung des mechanischen Aufbaus wurde mit den Komponenten der Fa. Festo implementiert. Der Aufbau hat nicht die geplanten geometrischen Maße von 4x6m, sondern ist mit einer linearen Achse realisiert. Die Achse ist ungefähr einen Meter lang und entspricht den aufgestellten Anforderungen. Der Aufbau ist in Abbildung 6-1 dargestellt.

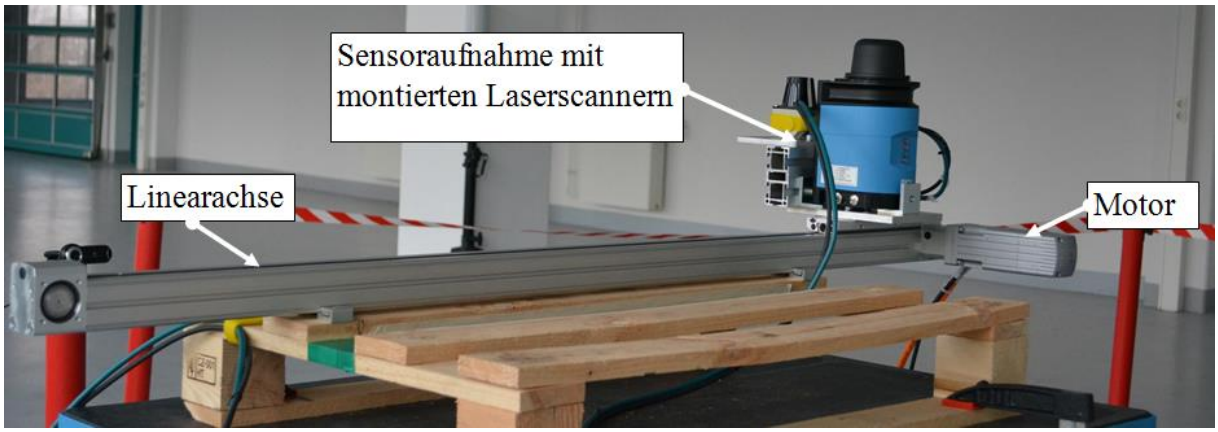


Abbildung 6-1 Mechanischer Aufbau

6.1.1. Komponenten

Der komplette Aufbau besteht aus vier mechanischen Komponenten.

- **Lineare Achse DGE-25-1045-ZR-RF-GK** – eine Zahnriemenachse mit einer Rollerführung sowie einem Hub von 1045mm.

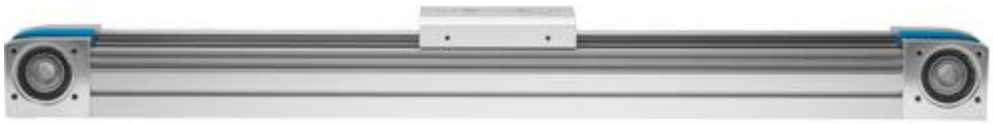


Abbildung 6-2 Lineare Achse

- **Motor EMMS-AS-55-S-TM** – bürstenloser, permanenterregter Synchron-Servomotor mit einem digitalen Absolutmesssystem.

Abbildung 6-3 Motor EMMS-AS

- **Motorkontroller CMMS-AS-C4-3A** – digitaler Servokontroller mit einem CANOpen Anschluss, integriertem EMV-Filter, einem Positionskontroller mit Lageregelung, einem Geschwindigkeitskontroller sowie integrierten Sicherheitsfunktionen.

Abbildung 6-4 Motorkontroller CMMS-AS

- **Steuerung CPX-CEC-C1** – eine CoDeSys Steuerung mit einer Ethernet und einer CANopen Schnittstelle. Außerdem ist die SPS mit zusätzlichen vier I/O Blöcken und mit einer Ventilinsel erweitert.

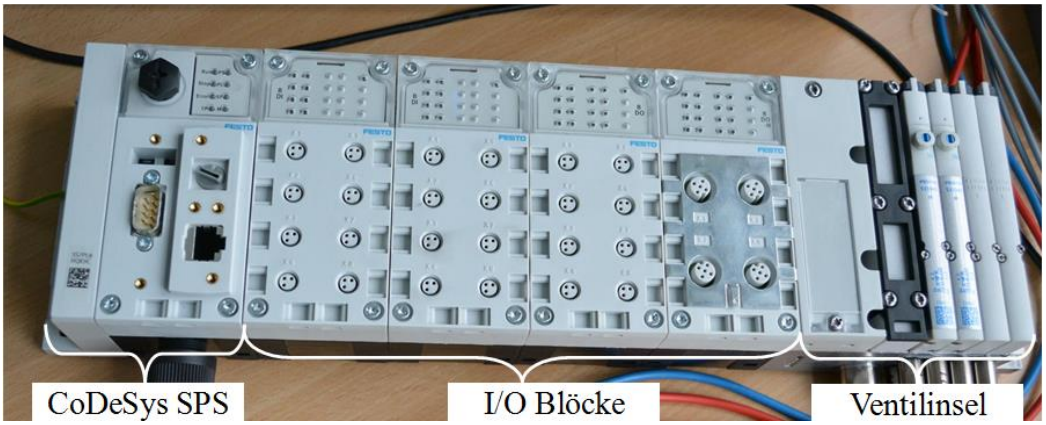


Abbildung 6-5 CoDeSys SPS

6.1.2. Inbetriebnahme der linearen Achse

Die Konfiguration des Aufbaus wie auch die ersten Bewegungen der Achse konnten über das Festo Configuration Tool (FCT) vorgenommen werden. Das Tool ist eine spezielle Software, welche die Konfiguration eines kompletten Systems ermöglicht.

Die Inbetriebnahme erfolgt in wenigen Schritten. Als Erstes soll ein Projekt im FCT mit allen verwendeten Komponenten erstellt werden. Die fertige Projektübersicht ist in Abbildung 6-6 ersichtlich.

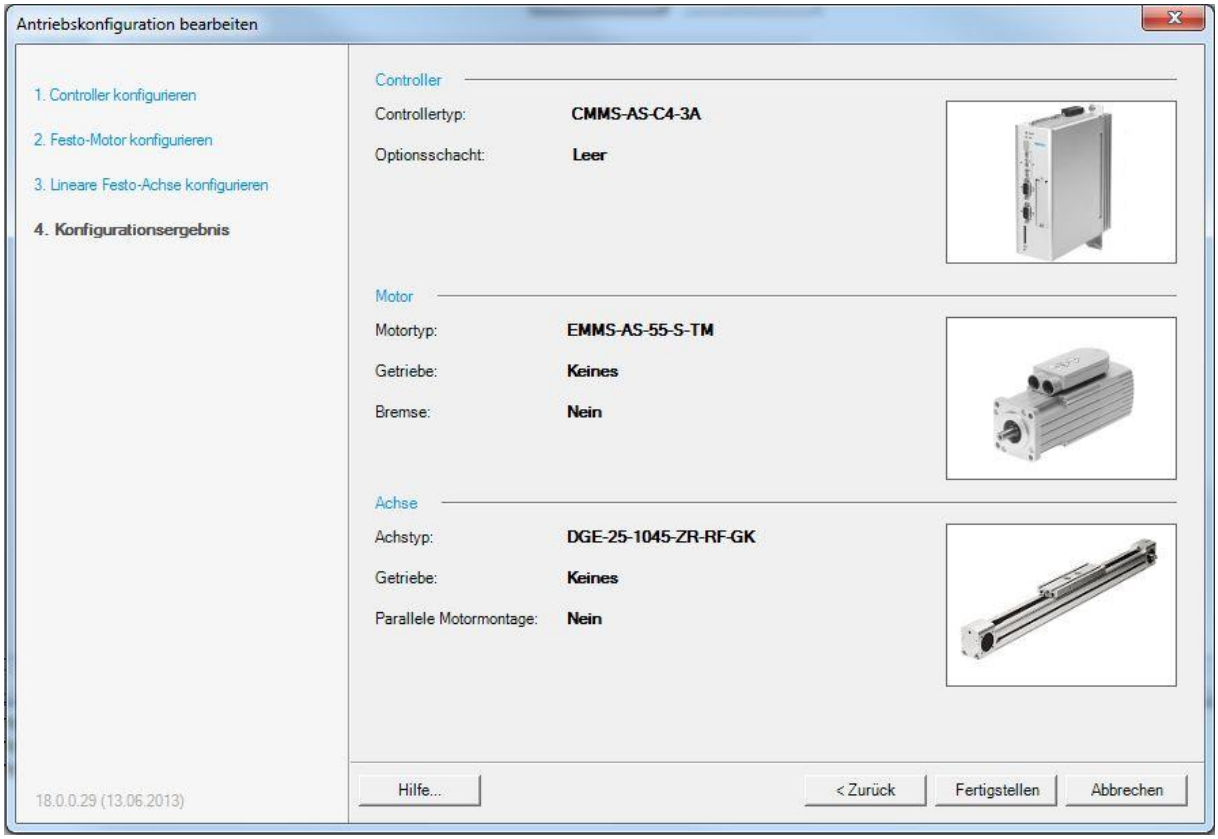


Abbildung 6-6 Konfigurationsergebnis

Damit der Motorkontroller aktiviert wird, sollen bestimmte Eingänge des 25-poligen D-SUB-Steckers geschaltet sein. Es sind insgesamt drei Pins, auf denen ein Hochsignal sein soll. Die Eingänge sind:

- DIN 4 Endstufenfreigabe
- DIN 5 Reglerfreigabe
- DIN13 Stop

Abbildung 6-7 zeigt den Anschluss von Eingängen des Controllers an die SPS.

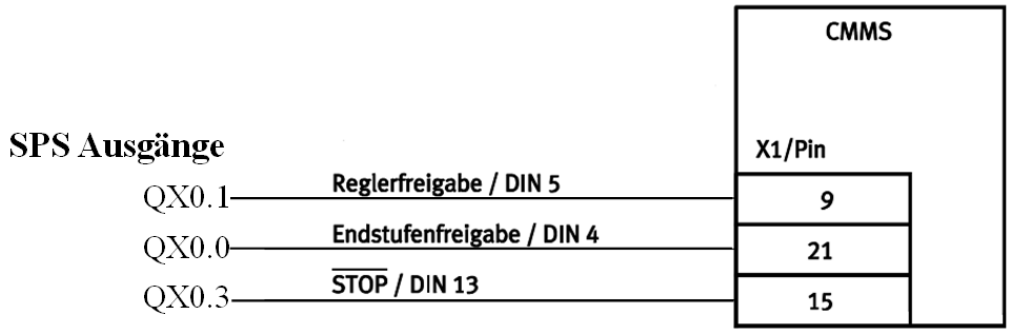


Abbildung 6-7 Anschluss des Controllers an SPS

Wenn der Motorkontroller an den PC über eine serielle Schnittstelle angeschlossen ist, ist es möglich, den Kontroller per FCT online zu schalten. Bevor man die ersten Bewegungen ausführen kann, sollen im Bedienfenster FCT und die Reglerfreigabe (siehe Abbildung 6-8) aktiviert sein.

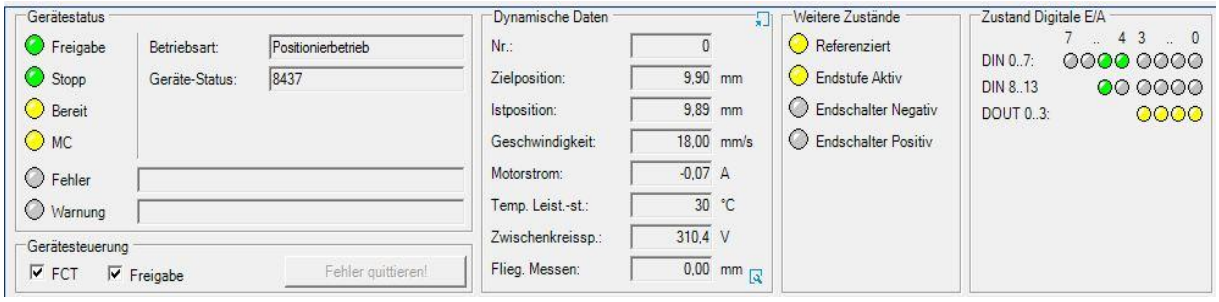


Abbildung 6-8 Bedienfenster des FCT

Bei Einstellung der CANopen-Schnittstelle sollte man unbedingt die Auflösung der Position definieren. Damit die Auflösung im Mikrometerbereich liegt, soll der Exponent auf 10^{-3} eingestellt sein (siehe Abbildung 6-9)

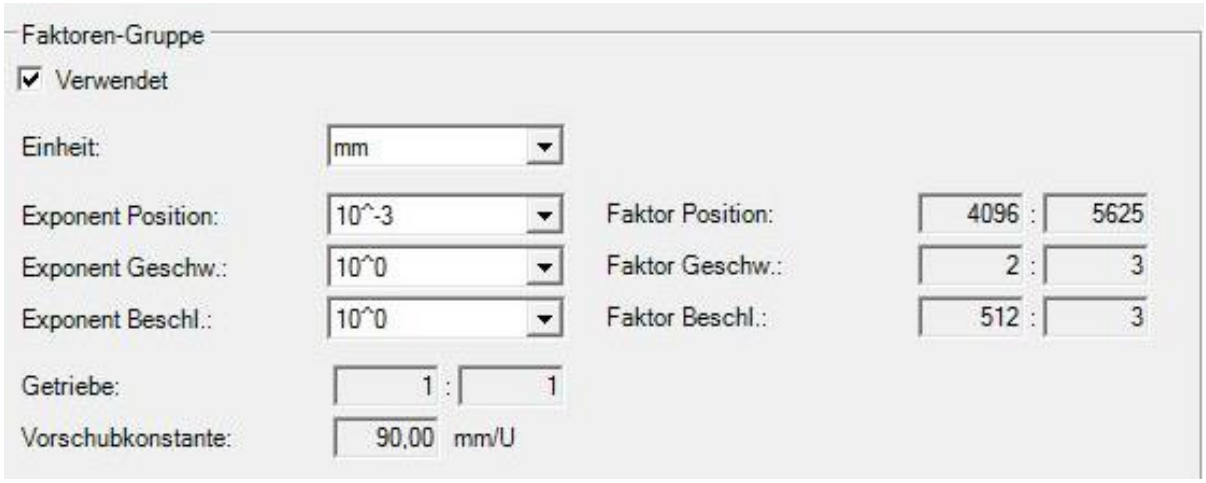


Abbildung 6-9 Einstellung der Auflösung der Position

6.1.3. Aufbau des Versuchsstandes in der Testhalle

Die Position des Versuchsstandes ist in Abbildung 6-10 skizziert.

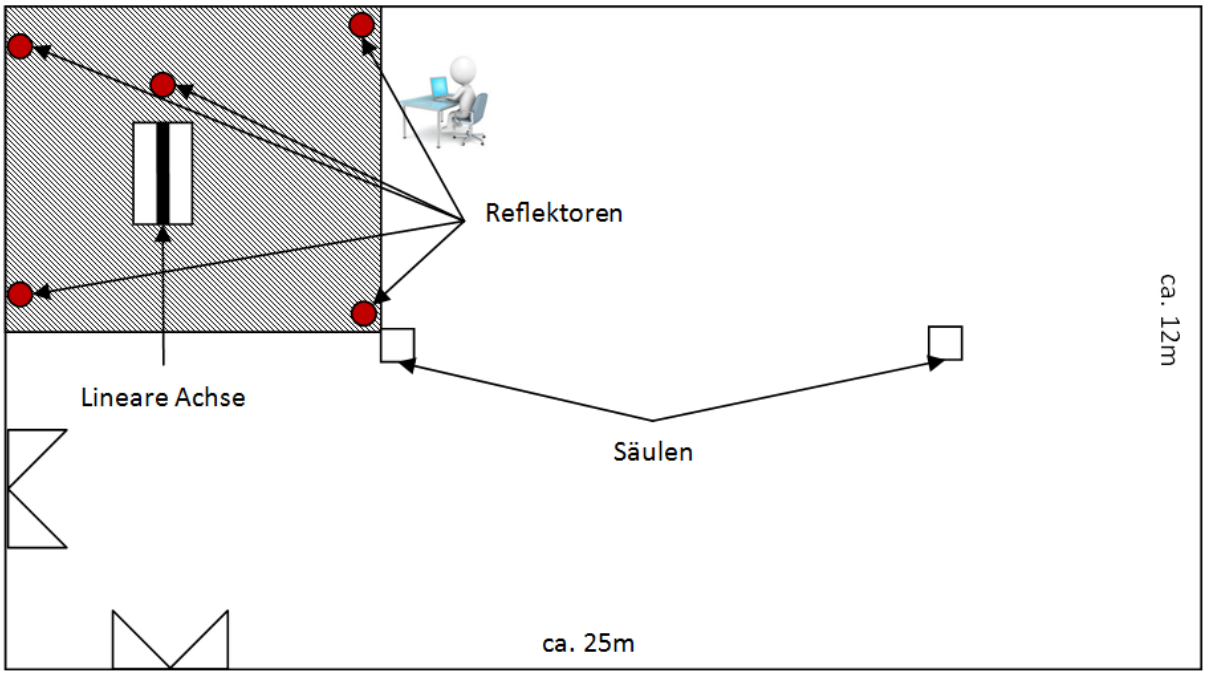


Abbildung 6-10 Position des Versuchsstandes in der Testhalle

6.2. Steuerung des Positioniertisches (CoDeSys V3)

In diesem Kapitel wird beschrieben, wie das mit CoDeSys V3 entwickelte Programm zur Steuerung des Positioniertisches aufgebaut ist. Die Steuerung ist in Form einer Soft-SPS realisiert. Das heißt, dass die Implementierung der Steuerung auf einem industriellen PC erfolgte.

Der Aufbau der Steuerung ist in Abbildung 6-11 grafisch dargestellt.

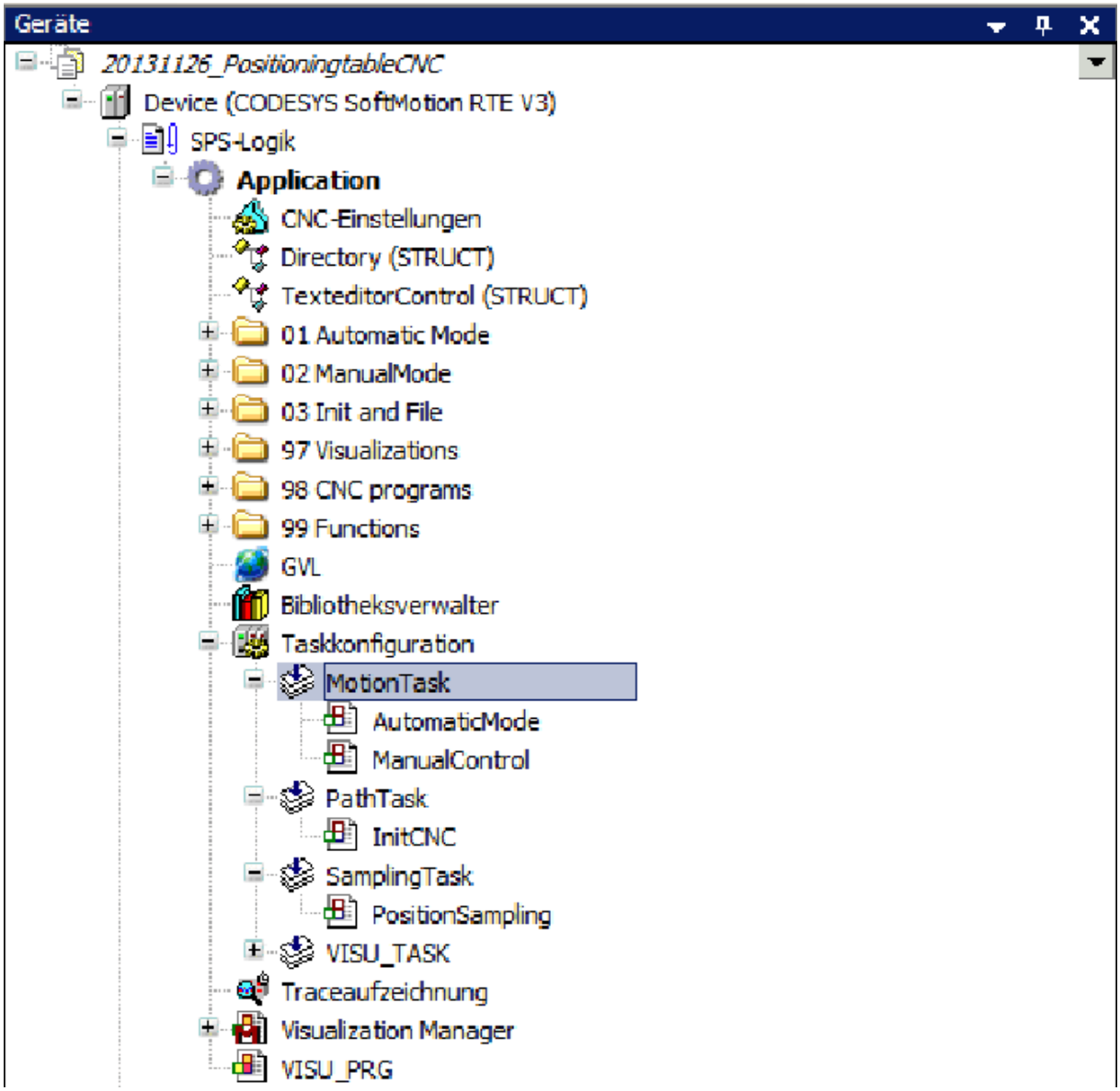


Abbildung 6-11 Aufbau der Steuerung(CoDeSys V3)

Das Programm ist so aufgebaut, dass man im manuellen oder automatischen Modus die Achsen des Positioniertisches betreiben kann. Die beiden Modi werden in den folgenden Kapiteln anhand der Ablaufdiagramme beschrieben – eine detaillierte Beschreibung mit Strukturen, Variablen und Funktionen ist im Anhang zu finden.

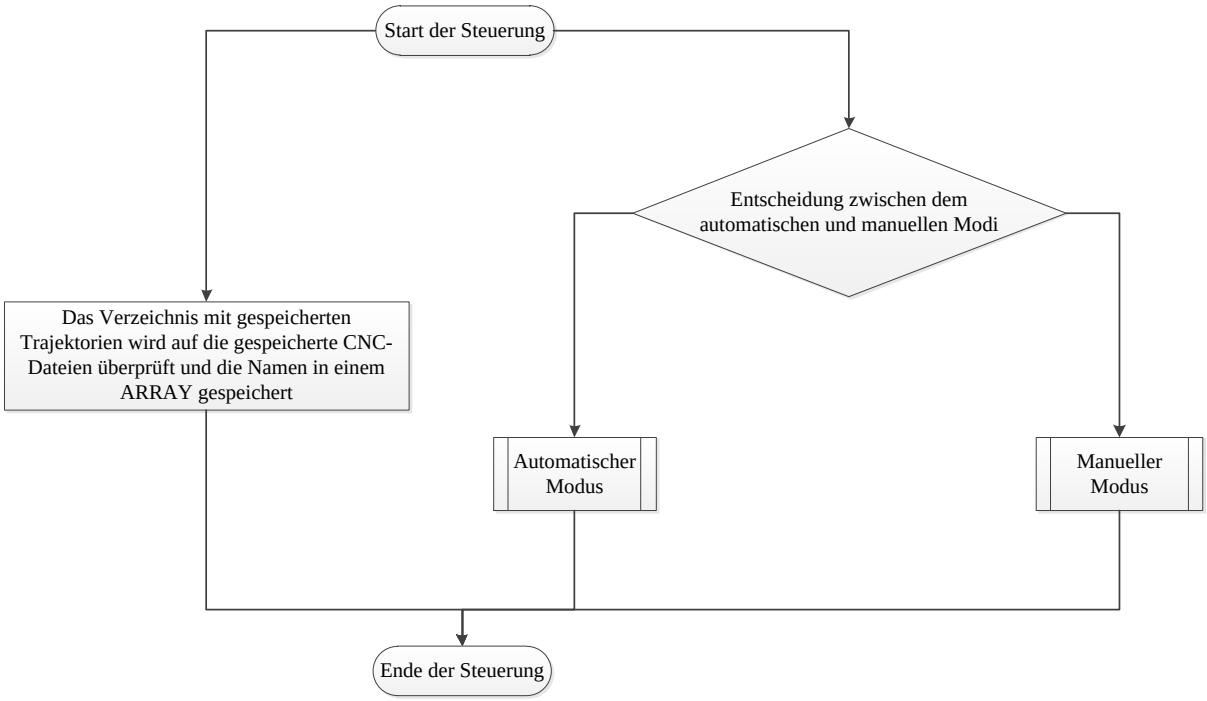


Abbildung 6-12 Allgemeiner Ablauf der Steuerung

Nach dem Start der Steuerung wird das Verzeichnis, in dem die CNC-Dateien gespeichert sind, auf das Vorhandensein der Dateien überprüft. Die Dateien können entweder online über die Steuerung erstellt werden oder offline direkt im Betriebssystem. Die Namen der entdeckten Dateien werden in alphabetischer Reihenfolge in einem ARRAY gespeichert. Die Entscheidung, in welchem Modus die Steuerung des Positioniertisches laufen soll, wird parallel zur Überprüfung getroffen.

6.2.1. Manueller Modus

Der Anwender kann im manuellen Modus die einzelne Achse steuern. Die Orientierung und die Position der Sensoraufnahme können beliebig geändert werden. Ein Ablauf der manuellen Steuerung ist in Abbildung 6-13 ersichtlich.

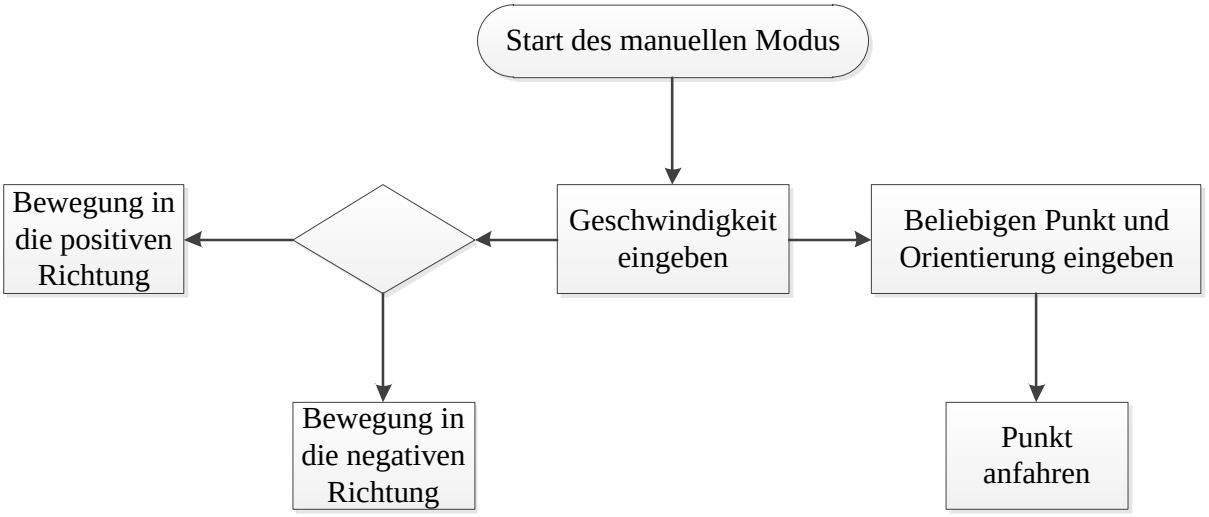


Abbildung 6-13 Ablauf der manuellen Steuerung

Das Programm ist so aufgebaut, dass nach dem Start alle Antriebe eingeschaltet werden und somit bereit zum Ansatz sind. Die Beschleunigung bleibt im manuellen Modus konstant, lediglich die Geschwindigkeit kann variiert werden. Außerdem kann jede beliebige Position und Orientierung durch eine Eingabe der Koordinaten angefahren werden.

6.2.2. Automatischer Modus

Der automatische Modus ist etwas komplexer als der manuelle aufgebaut. Er ist in drei Unterprogramme aufgeteilt und zwar:

- Programmierung
- Ausführung
- Analyse

Im Programmiermodus kann der Anwender beliebige Trajektorie erstellen oder eine bereits erstellte ändern und als eine CNC-Datei abspeichern. Die Funktion ist durch einen Texteditor-Funktionsblock der CoDeSys-Biblothek gegeben.

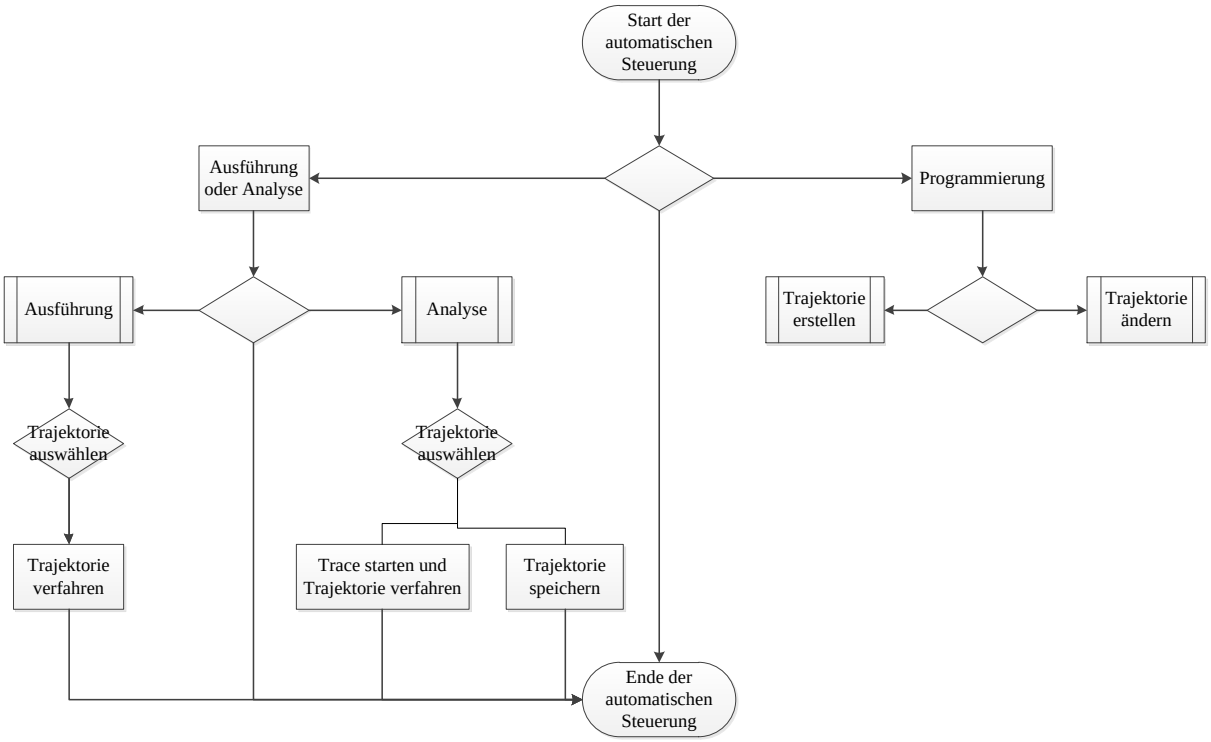


Abbildung 6-14 Ablauf der automatischen Steuerung

Abbildung 6-14 zeigt den Ablauf der automatischen Steuerung. Die Unterprozesse Ausführung und Analyse ermöglichen das Verfahren über eine der gespeicherten Trajektorien. Die Orientierung der Sensoraufnahme liegt tangential zum Verfahrweg. Die CNC-Dateien werden mit dem NC-Decoder gelesen und mithilfe des Funktionsblocks ein Zeiger auf die SMC_OUTQUEUE-Struktur generiert. Die Struktur dient als Eingangssignal für den Interporator, der einzelne Koordinaten in definierten Zeitabständen an die Achsen gibt.

Der Anwender kann im Analyse-Modus die Trace-Diagramme in Echtzeit betrachten. Der Modus hat als weitere Funktion die Möglichkeit, eine Trajektorie in eine Datei zu speichern. Die Funktion ist mit zwei synchronisierten Timern realisiert: Ein Timer ist für die Dauer der kompletten Testfahrt zuständig, der andere repräsentiert die Abtastzeit.

7. Validierung

Im Kapitel 7 erfolgt die Validierung der entwickelten Steuerung anhand einer Simulation sowie die Untersuchungen mit reellem Aufbau.

7.1. Simulation

Die Simulation der Scanaufnahme von Laserscannern erfolgt nach einem einfachen Ablauf, dieser ist in der Abbildung 7-1 grafisch dargestellt.

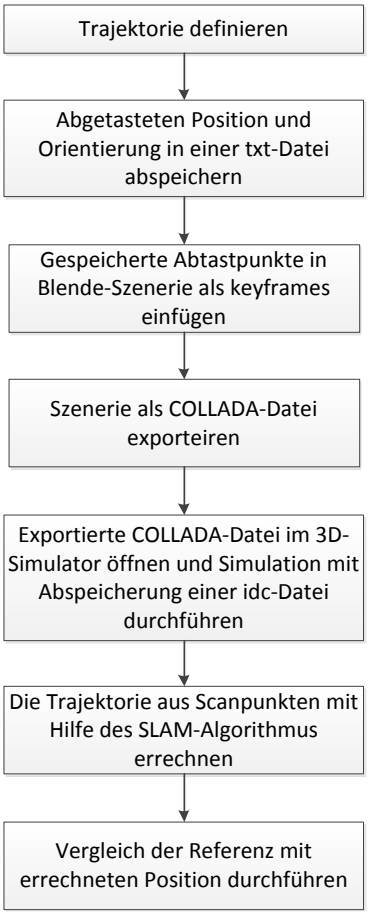


Abbildung 7-1 Ablauf der Simulation

7.1.1. 3D Szenerie mittels Blender

Ein 3D Model. des Positioniertisches wurde im Blender erstellt (Abbildung 7-2). Das Model kann in jede beliebige, auch im Blender erstellte Umgebung eingefügt werden. Der Leserscanner ist auf dem Schlitten der X-Achse montiert. Die X-Achse ist mit zwei parallelen Y-Achsen verbunden. Das Portal ist wie spezifiziert vier Meter breit und sechs Meter lang.

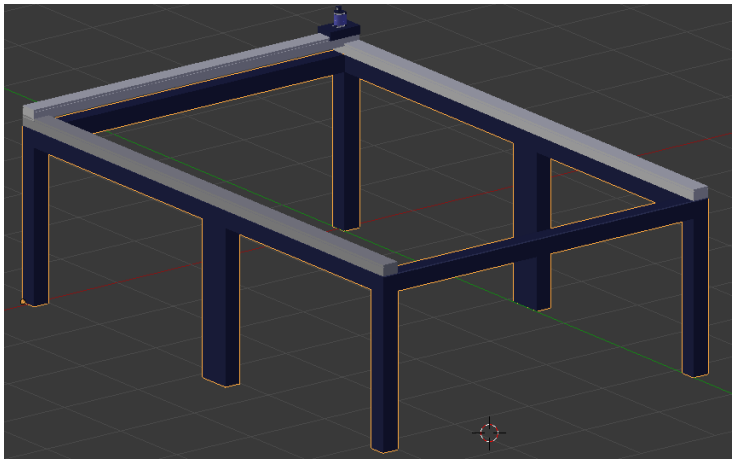


Abbildung 7-2 Blender Modell des Positioniertisches

Das Modell der Testhalle ist wie der Positioniertisch mit Blender erstellt. Alle geometrischen Abmessungen der Halle wurden aus einer technischen Zeichnung entnommen und auf das Modell übertragen. In Abbildung 7-3 ist der Positioniertisch in der Testhalle zu sehen.

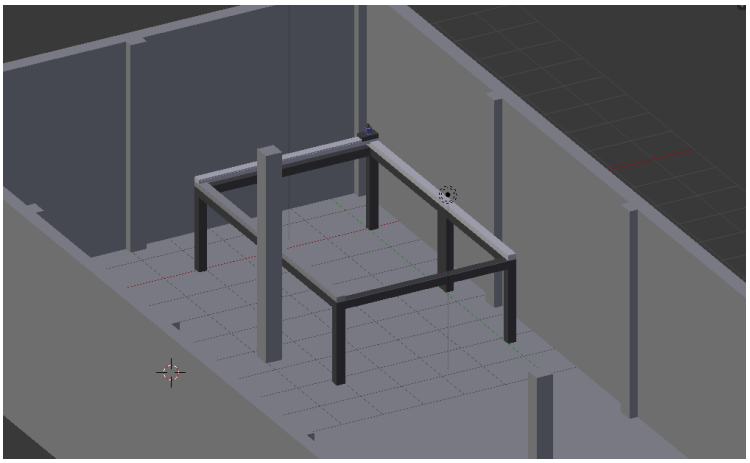


Abbildung 7-3 Blender Szenerie

Man soll bei der Erstellung der Animation darauf achten, dass in einer Sekunde maximal 24 Keyframes eingefügt werden können. Das heißt, dass maximal eine Abtastfrequenz von 24 Hz erreicht wird. Abbildung 7-4 zeigt ein Beispiel einer Realisierung der 8 Hz (Scanfrequenz vom NAV350).

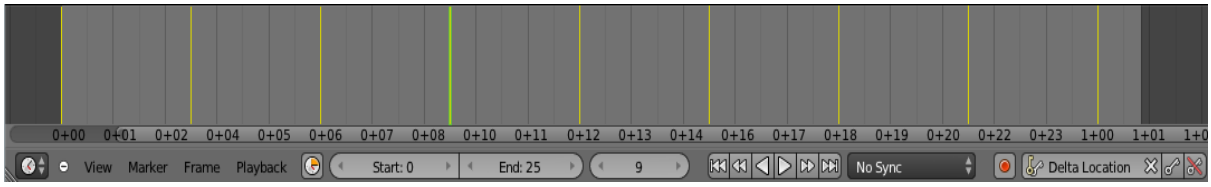


Abbildung 7-4 Keyframes (8 Hz)

Das automatische Einfügen der Keyframes ist mit einem Python-Script möglich. Die in einer txt-Datei gespeicherte Trajektorie wird im Script aufgerufen und einzelne Abtastpunkte werden als Keyframes eingefügt. Dabei ist auf folgende Punkte zu achten (Abbildung 7-5):

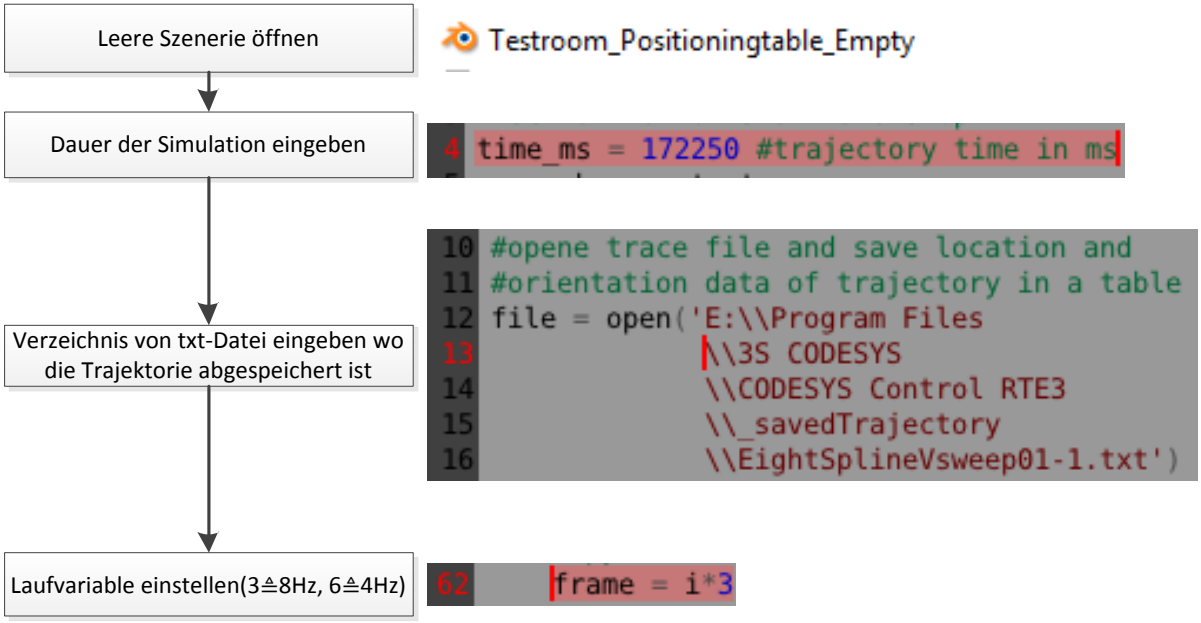


Abbildung 7-5 Ablauf der automatischen Erstellung der Szenerie

7.1.2. 3D Simulation

Im letzten Schritt wird die erstellte Szenerie als COLLADA-Datei exportiert. Im Weiteren wird die COLLADA-Datei im 3D-Simulator geöffnet.

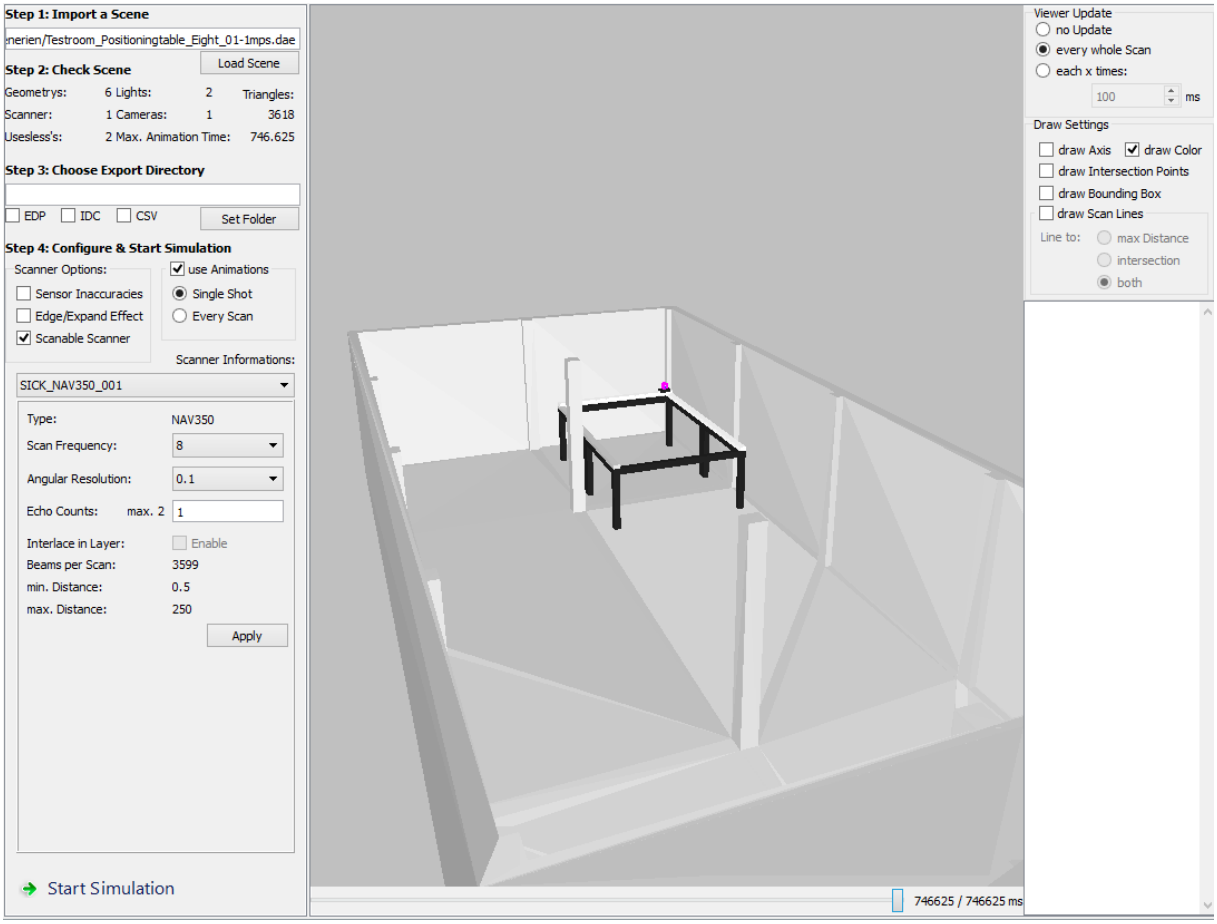
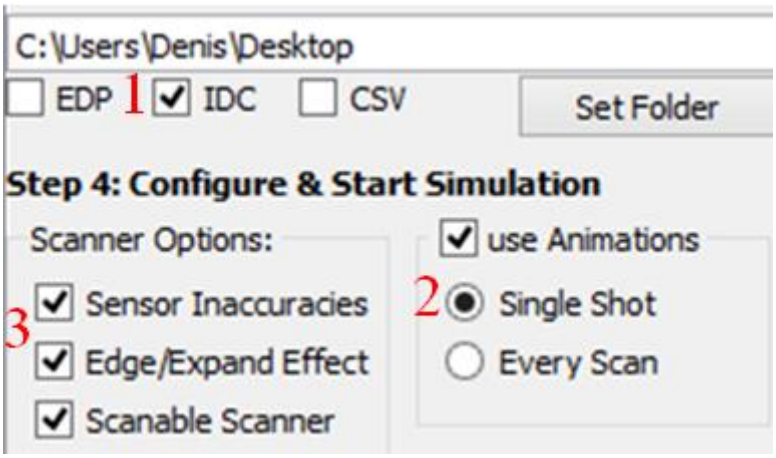


Abbildung 7-6 3D-Simulation

Bevor die Simulation gestartet wird, sollen wichtige Einstellungen vorgenommen werden. Die Einstellungen sind:

1. IDC-Datei speichern
2. Simulation mit Auswertung jedes einzelnen Laserstrahls
3. Messstreuung und Eckeneffekt einschalten



Nachdem alle Einstellung getätigt sind, kann die Simulation gestartet werden.

7.1.3. Laserview

Laserview ist ein SICK Tool. Man kann mit diesem Tool die gespeicherte IDC-Dateien abspielen oder auch die Erstellung einer neuen IDC-Datei vornehmen. Eine essenzielle Aufgabe vom Laserview ist die Visualisierung der Scans.

Nachdem Abschluss der Simulation und der Auswertung der IDC-Datei mit dem SLAM-Algorithmus, kann die neugenerierte IDC-Datei im Laserview geöffnet werden. Die IDC-Datei beinhaltet bereits eine Visualisierung der Trajektorie, die in Abbildung 7-7 zu sehen ist.

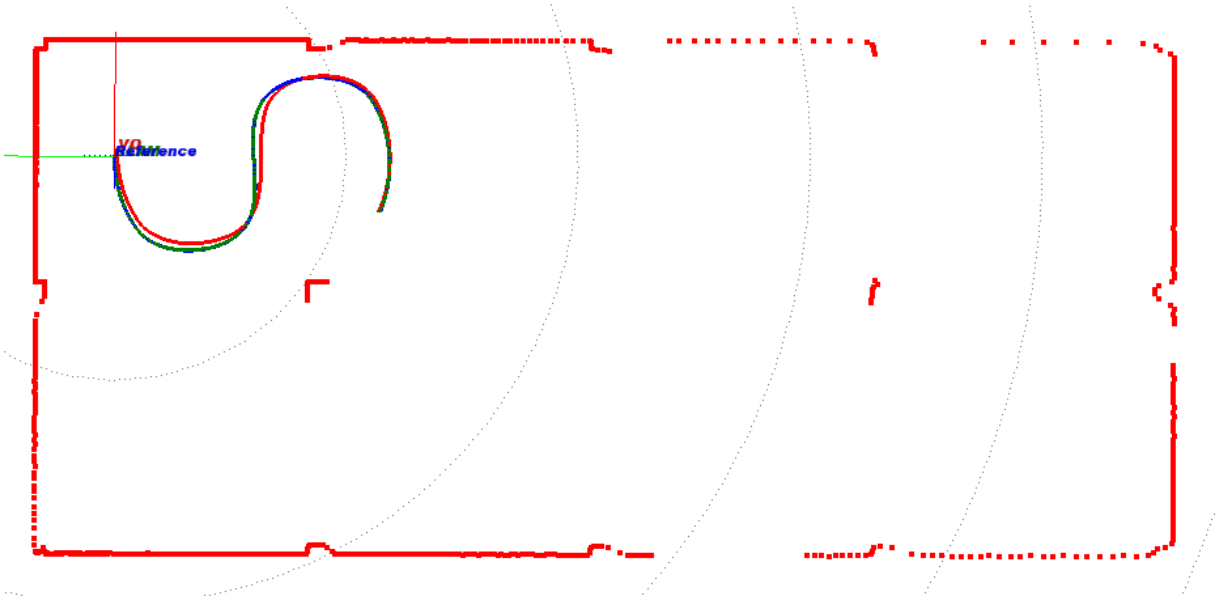


Abbildung 7-7 Simulierte Scans mit errechneter Trajektorie

7.2. Testfahrt: Simulation

Eine der wichtigsten Untersuchungen, die mit der Simulation der Steuerung durchgeführt werden kann, ist die Analyse des SLAM-Algorithmen (siehe Unterkapitel 3.4). Das Fehlerverhalten des Algorithmus ist nicht bekannt. Um den Fehler zu kompensieren, sollen mehrere Fahrten mit unterschiedlichen Trajektorien vorgenommen werden.

Das Ziel der Untersuchung besteht darin, anhand der verschiedenen Trajektorien eine Aussage über die Entwicklung der Fehler während der Fahrt zu tätigen. Die Aussage soll die Abhängigkeit der Fehler von der Geschwindigkeit, Beschleunigung sowie der Fahrtzeit darstellen.

Definierte Trajektorien:

X-Sweep eine Fahrt auf der X-Achse mit Änderung der Geschwindigkeit bei konstanter Beschleunigung

Y-Sweep eine Fahrt auf der Y-Achse mit Änderung der Geschwindigkeit bei konstanter

Beschleunigung

7.2.1. X-Sweep

Um die erste Auswertung zu vereinfachen sowie eine Vorstellung über die Entwicklung der Fehlern zu gewinnen, ist es sinnvoll, eine Trajektorie mit einfacher Dynamik zu nehmen.

Die Trajektorie ist so definiert, dass der NAV350 auf der X-Achse des Positioniertisches bewegt wird.

Tabelle 7-1 Eigenschaften der X-Sweep-Trajektorie

	X	Y	W
Start Position	0 m	0 m	180 °
Endposition	3.95 m	0 m	180 °
Min. Geschwindigkeit	0.2 m/s	0 m/s	0 °/s
Max. Geschwindigkeit	1 m/s	0 m/s	0 °/s
Beschleunigung	0.5 m/s²	0 m/s²	0 °/s²

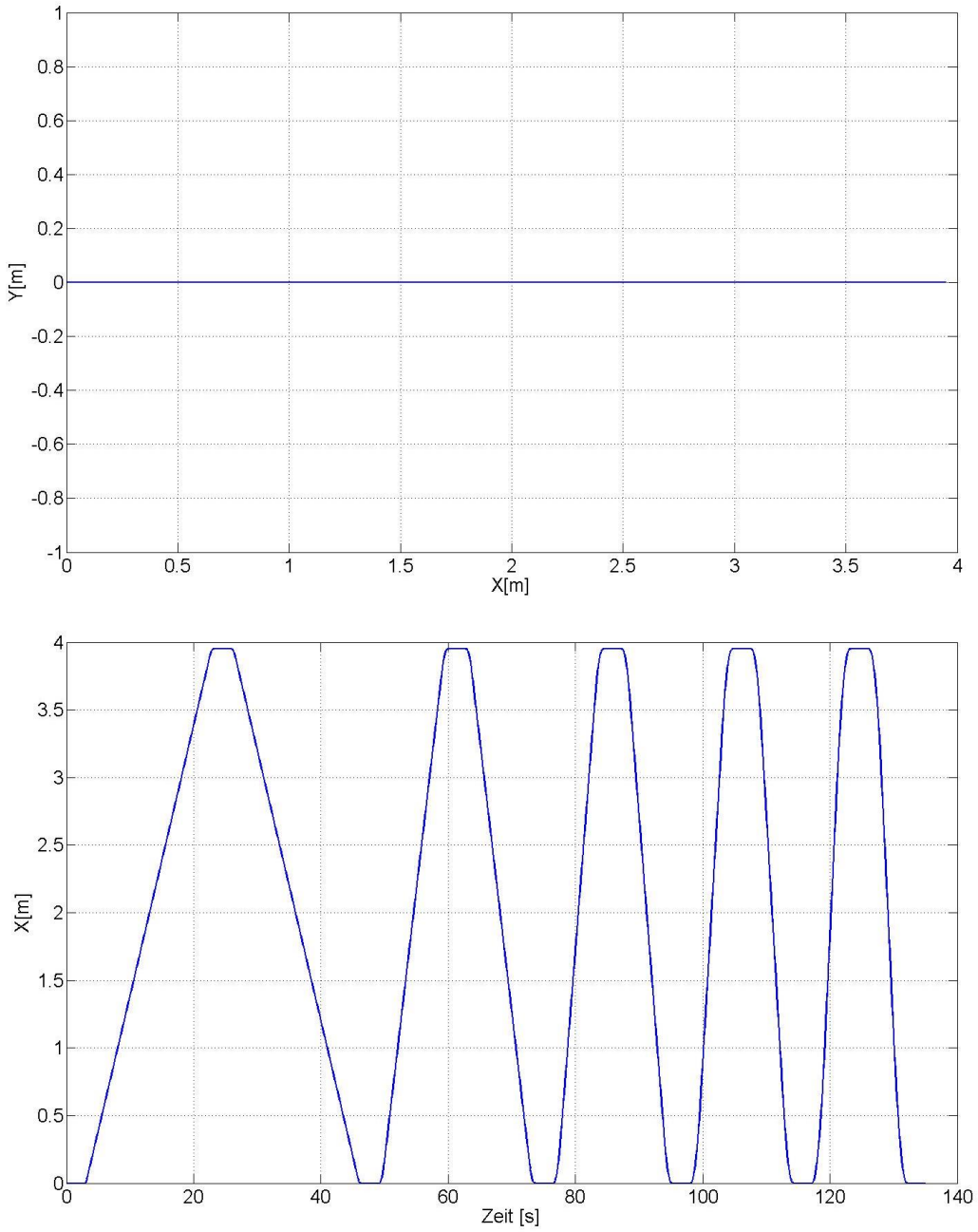


Abbildung 7-8 Ist-Position des NAV's während der Bewegung

Die Geschwindigkeit der Bewegung steigt linear von 0,2 m/s bis 1m/s in 0,2 m/s Schritten. Der Verlauf der Geschwindigkeitsänderung ist in Abbildung 7-9 dargestellt. Die Orientierung des NAV's ist während der kompletten Fahrt konstant und beträgt 180°.

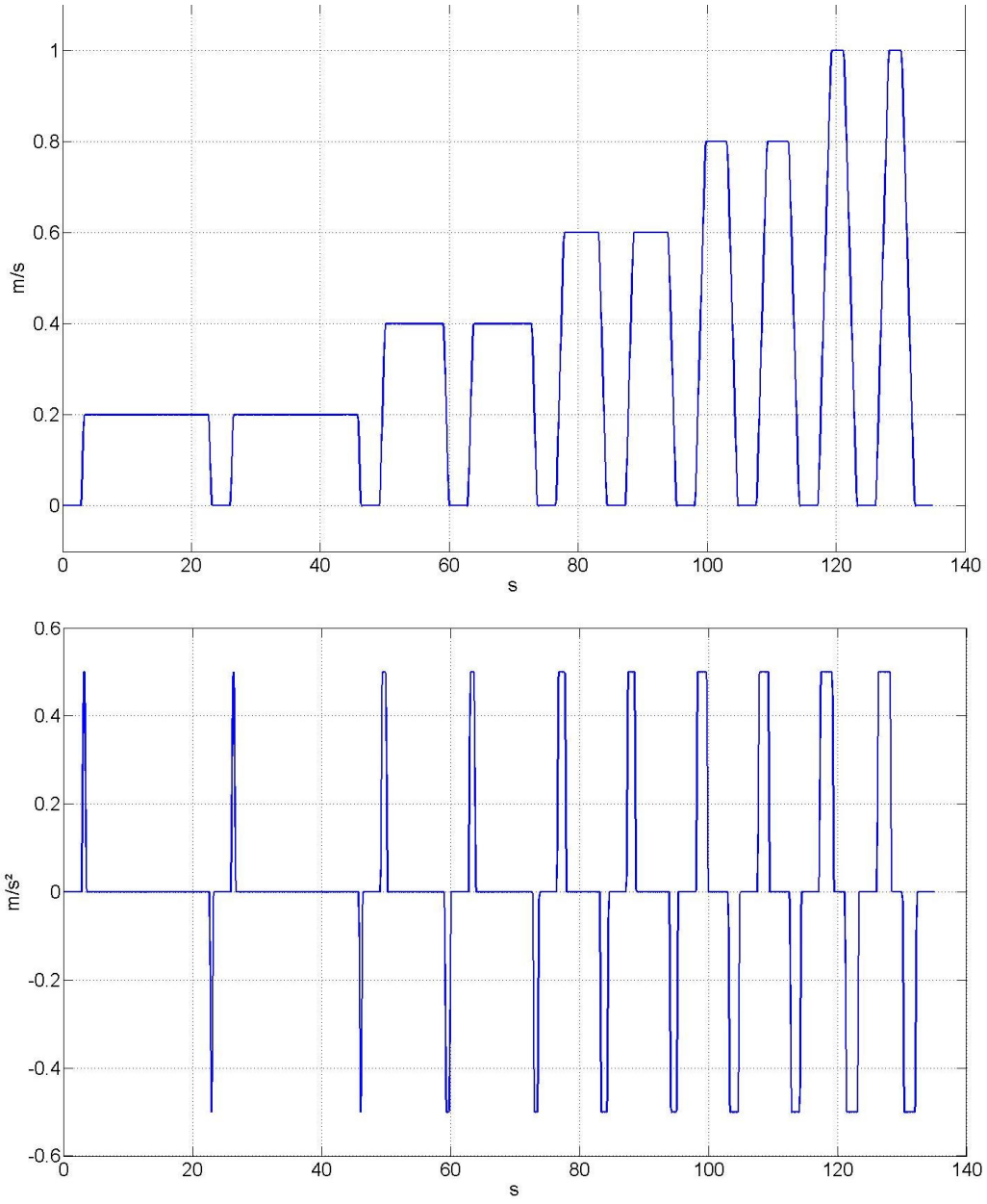


Abbildung 7-9 Geschwindigkeit-Zeit und Beschleunigung-Zeit Diagramme der X-Sweep Trajektorie

In Abbildung 7-9 sind fünf Fahrten mit unterschiedlicher Geschwindigkeit zu sehen. Die Beschleunigung bzw. Verzögerung während der kompletten Fahrt bleibt konstant. Wenn die Zielposition der einzelnen Bewegung erreicht ist, erfolgt eine Pause, die eine Verweilzeit die Achsen in einer Position hält.

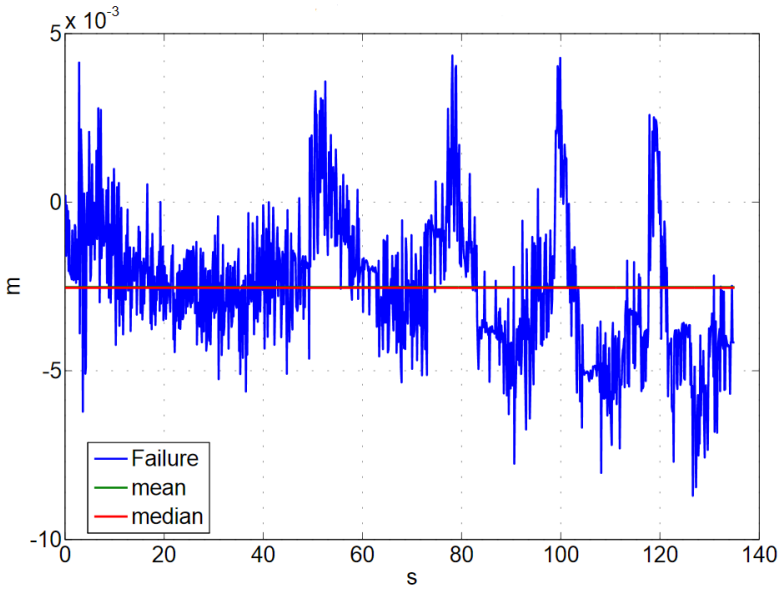


Abbildung 7-10 Fehler in die Y-Richtung (X-Sweep Trajektorie)

In Abbildung 7-10 kann man erkennen, dass der Fehler in die Y-Richtung klein ist und den maximalen Wert von 7 mm annimmt. Das heißt, dass die Entwicklung des Fehlers in der Richtung orthogonal zur Fahrtrichtung minimal ausfällt. Das Verhalten ist verständlich, da die größten Kräfte, die durch Dynamik entstehen, in die Fahrtrichtung wirken. Der maximale Fehler ist in Abbildung 7-11 zu sehen. Hier wird der Verlauf des Fehlers in Fahrtrichtung dargestellt.

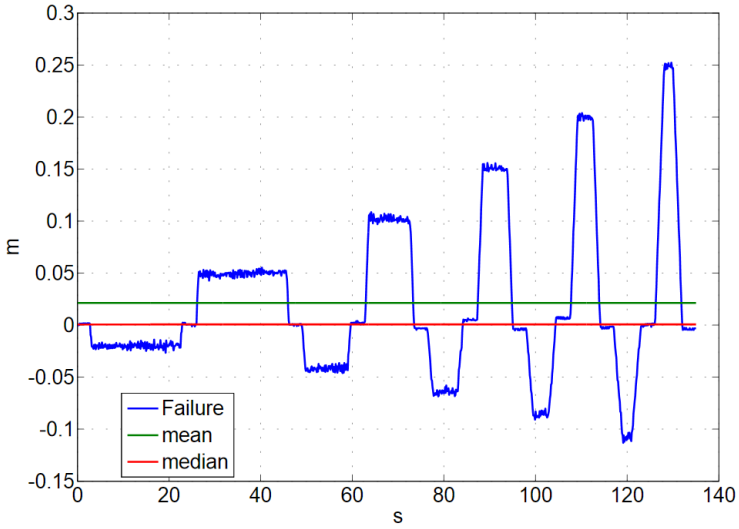


Abbildung 7-11 Fehler in die Fahrtrichtung (X-Sweep Trajektorie)

Der Fehler in die Fahrtrichtung nimmt mit der Geschwindigkeit näherungsweise linear zu. Der Maximalwert beträgt 25 cm bei einer Geschwindigkeit von 1 m/s. Überdies ist erkennbar, dass der Fehler auf der Rückfahrt um ca. 50% höher ist als während der Hinfahrt. Die Erklärung dafür sind die Zeitstempel des Simulators. Wenn die Abtastwerte des Simulators und des

Auswertealgorithmus unterschiedlich sind, kann es dazu kommen, dass durch eine Interpolation zwischen zwei Abtastpunkte zusätzliche Interpolationsfehler kommen.

7.2.2. Y-Sweep

Die Trajektorie ist so definiert, dass der NAV350 auf der Y-Achse des Positioniertisches bewegt wird. Bei der Trajektorie ist die Dynamik höher als bei der X-Sweep, somit kann das Fehlerverhalten bei Geschwindigkeiten kontrolliert werden, bei denen der Laserscanner ebenso in der Realität eingesetzt ist.

Tabelle 7-2 Eigenschaften der Y-Sweep-Trajektorie

	X	Y	W
Start Position	0 m	0 m	90 °
Endposition	0 m	0 m	90 °
Min. Geschwindigkeit	0 m/s	0,5 m/s	0 °/s
Max. Geschwindigkeit	0 m/s	3 m/s	0 °/s
Beschleunigung	0 m/s²	2 m/s²	0 °/s²

In Abbildung 7-12 lässt sich erkennen, dass der Fehler in die X-Richtung klein ist und den maximalen Wert von 9 mm annimmt. Das heißt, dass die Entwicklung des Fehlers in der Richtung orthogonal zur Fahrtrichtung sowie bei der vorherigen Testfahrt minimal ist. Der Mittelwert beträgt ca. 2,2 mm.

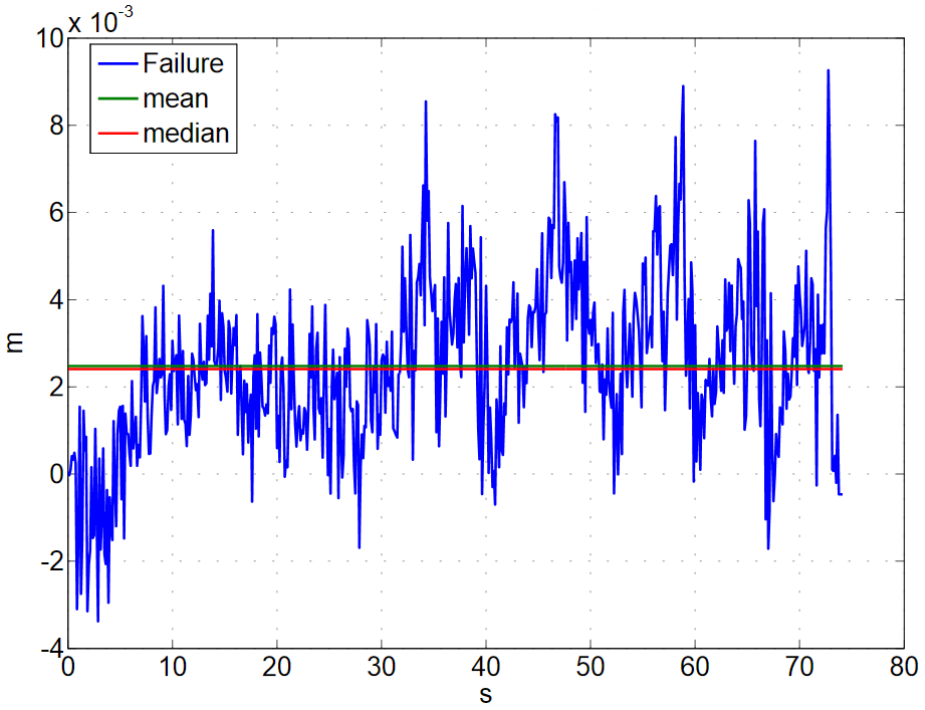


Abbildung 7-12 Fehler in die X-Richtung (Y-Sweep Trajektorie)

Das Fehlerverhalten in die Fahrtrichtung ist in Abbildung 7-13 ersichtlich. Der Fehler bei der maximalen Geschwindigkeit beträgt bedeutende 76 cm.

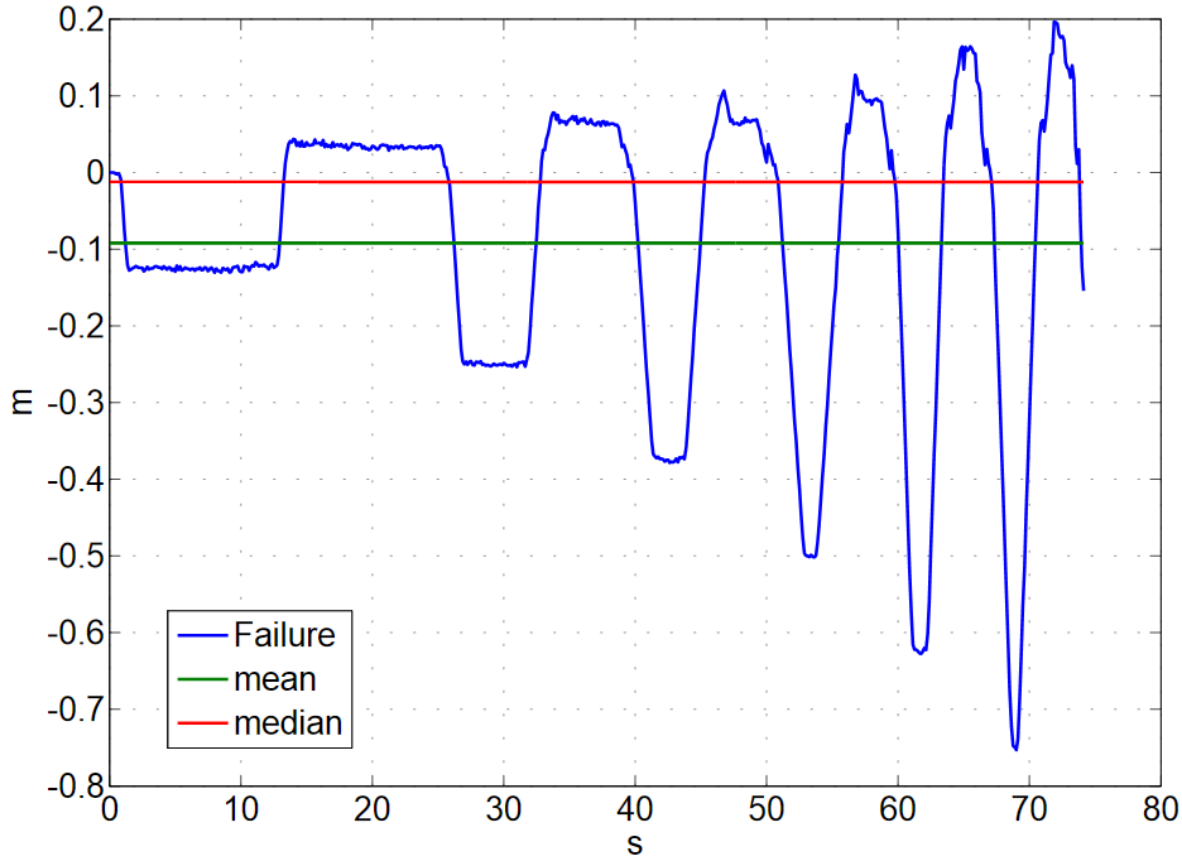


Abbildung 7-13 Fehler in die Fahrtrichtung (Y-Sweep Trajektorie)

Fazit

Das Fehlerverhalten konnte durch die einfachen Trajektorien X-Sweep und Y-Sweep mit einer linearen Veränderung der Geschwindigkeit und konstanten Beschleunigung detektiert sowie ausgewertet werden. Im dynamischen Betrieb, insbesondere bei hohen Geschwindigkeiten, erreicht der Fehler einen Wert von über 75 cm. Da der Simulator und der Auswertealgorithmus nicht synchronisiert sind, hängt das Verhalten des Fehlers stark von dem Zeitstempel der einzelnen Abtastpunkte ab. Die Position des Laserscanners NAV350 kommt verspätet, das heißt, dass das Signal um wenige Millisekunden auf der Zeitachse relativ zur Referenz verschoben ist. Die Abbildung 7-14 zeigt den zeitlichen Versatz. Weiterhin kann man in der Abbildung zwei Positionen bei 69 s der Fahrt erkennen: Der Fehler beträgt in dem Fall $\Delta Y = 753mm$.

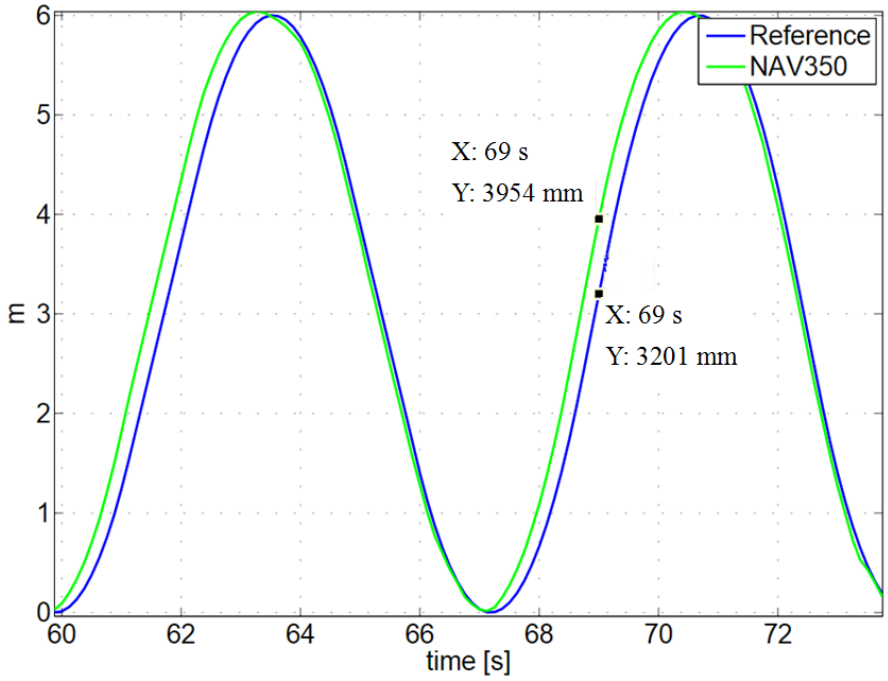


Abbildung 7-14 Zeitlicher Versatz der Signale (Y-Sweep mit v=3 m/s)

Wenn man die NAV Position um 125 ms auf der Zeitachse nach rechts verschiebt, sieht das Fehlerverhalten logischerweise anders aus. Der maximale Fehler beträgt nur noch 37 cm, was ungefähr der Hälfte des Fehlers vom Originalsignal entspricht. Die Optimierung des Fehlers ist in Abbildung 7-15 dargestellt.

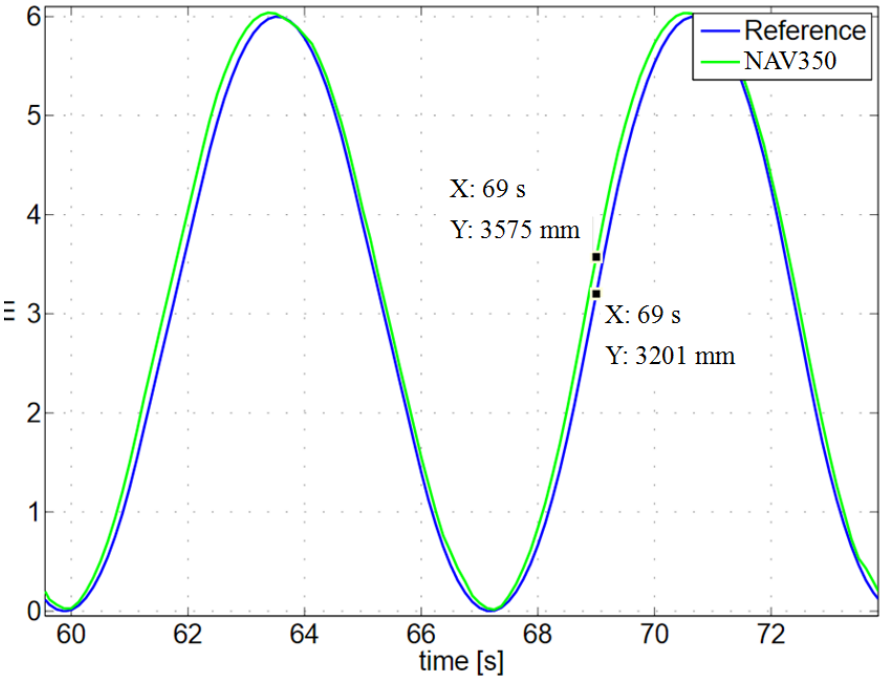


Abbildung 7-15 Verbessertes Signal vom Laserscanner

7.3. Testfahrt: lineare Achse

Mit der linearen Achse von Festo wurden mehrere Testfahrten vorgenommen. Wie in Kapitel 7.2 basieren die durchgeführten Fahrten auf der linearen Änderung der Geschwindigkeit. Die Werte für Beschleunigung und Verzögerung blieben konstant. Somit konnte die Position vom NAV350 mit der Position der linearen Achse verglichen und eine Differenz zwischen den beiden Positionen gebildet werden. Die Abhängigkeit des Fehlers von der Dynamik konnte überdies genauer analysiert werden.

7.3.1. Versuchsaufbau

Die Position und Orientierung der linearen Achse wurden bereits in Kapitel 6.1 beschrieben und blieben während der gesamten Versuchsreihe unverändert. Der Laserscanner wurde mithilfe einer Adapterplatte auf den Schlitten der Achse montiert.

7.3.2. Aufnahme der ersten Scans und Mapping

Die Konfiguration des Laserscanners wurde mithilfe einer Software namens „SOPAS“ ermöglicht. Der erste Scan der Testhalle konnte somit aufgenommen werden. Abbildung 7-16 zeigt den ersten Scan der Testhalle. in Abbildung sind die Reflektoren als farbige Kreise zu erkennen. Die von den Fenstern reflektierten Laserstrahlen sind ebenso in der Abbildung 7-16 zu sehen. Sie sind erkennbar an den spiegelverkehrten Messpunkten im oberen Bereich der Abbildung.

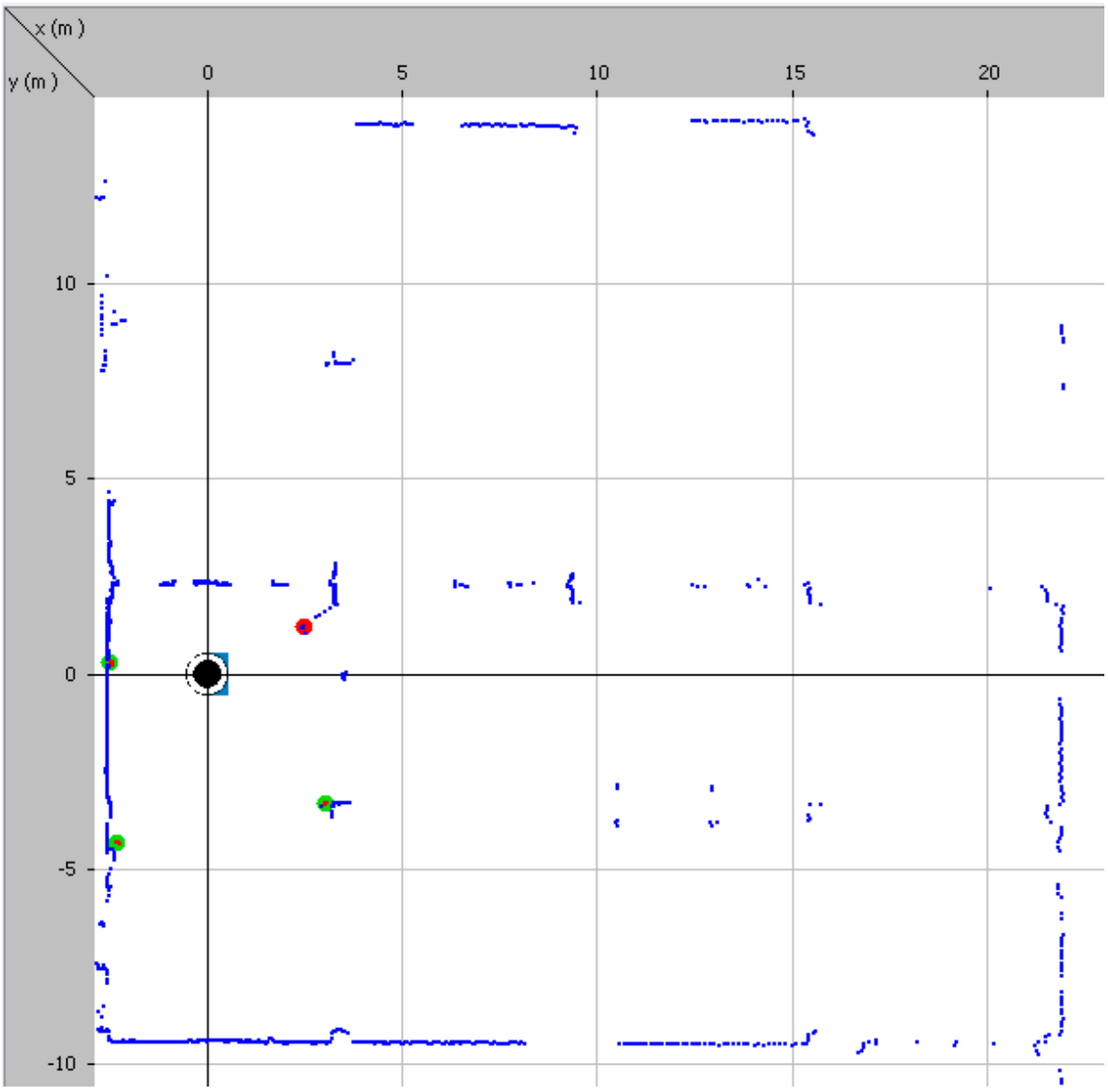


Abbildung 7-16 Erster Scann der Testhalle

Damit die genaue Position des Laserscanner in der Testhalle berechnet werden kann, soll vor jedem Versuch ein neues Layout mit Landmarken erstellt werden. Die Reflektoren werden im Landmarkenerkennung-Betrieb des Laserscanners als Landmarken interpretiert und die Position der Reflektoren im nächsten Schritt während des Mapping-Betriebes in das Layout der Umgebung eingetragen. Die Position des Laserscanners wird als absoluter Null-Punkt im globalen Koordinatensystem interpretiert. Man muss an der Stelle achten, dass die Orientierung des Laserscanners mit der Orientierung der linearen Achse übereinstimmt. Die Y-Achse des Laserscanners soll auf der Bewegungslinie der Achse liegen. Dafür muss beim Mapping die Orientierung des NAV350 auf 180° gesetzt werden. Das Layout der Testhalle mit den Landmarken ist in Abbildung 7-17 dargestellt.

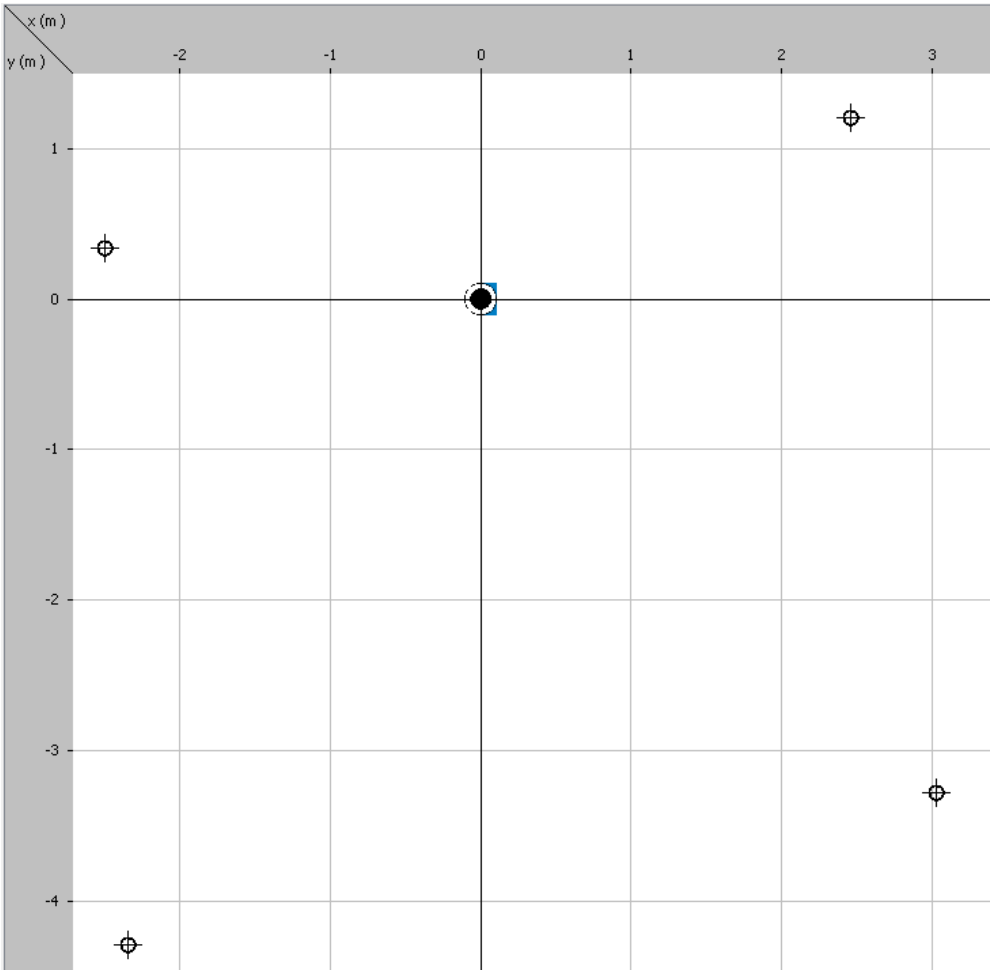


Abbildung 7-17 Layout mit Landmarken

Nachdem das Layout in das Gerät übertragen wurde und der Navigation-Betrieb eingeschaltet ist, berechnet der Laserscanner anhand der Entfernung von den Reflektoren seine absolute Position im globalen Koordinatensystem. Der Scan der Testhalle im Navigation-Betrieb des Laserscanners ist in Abbildung 7-18 dargestellt.

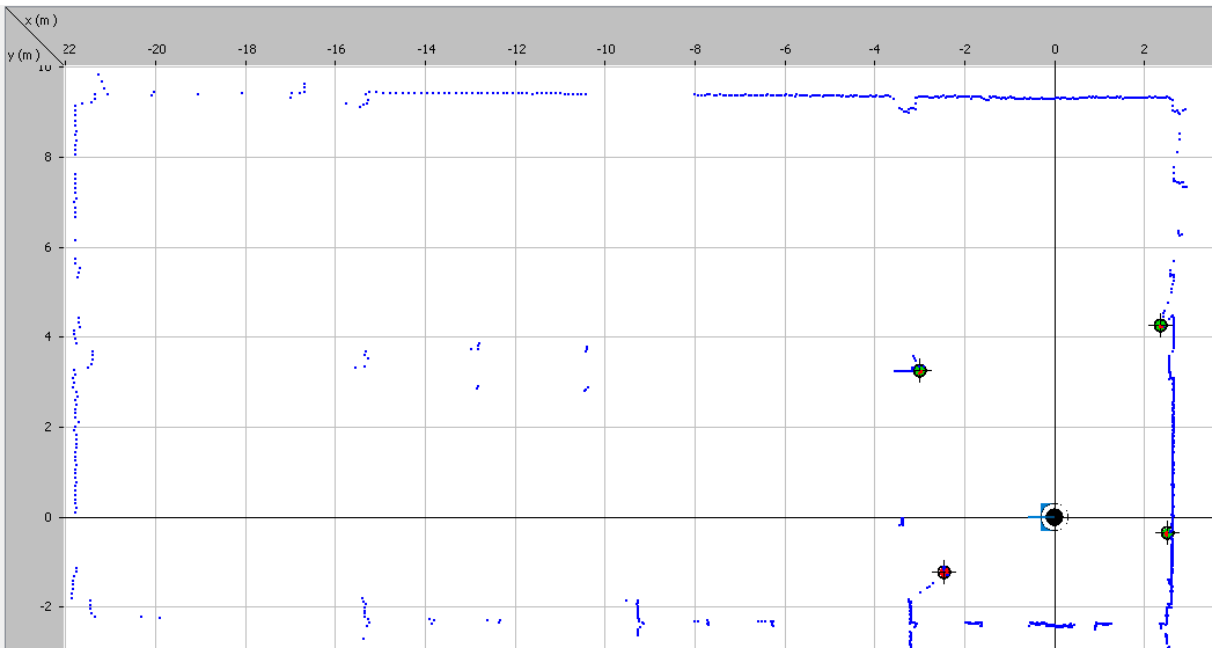


Abbildung 7-18 Navigation-Betrieb des Laserscanners

In Abbildung 7-18 ist zu erkennen, dass der Laserscanner die Position (0,0) hat. Wenn der Schlitten der linearen Achse um 1000 mm zur unteren Wand der Testhalle bewegt wird, ändert sich die Position des NAV350 dementsprechend. Die Änderung der Position ist in Abbildung 7-19 ersichtlich.

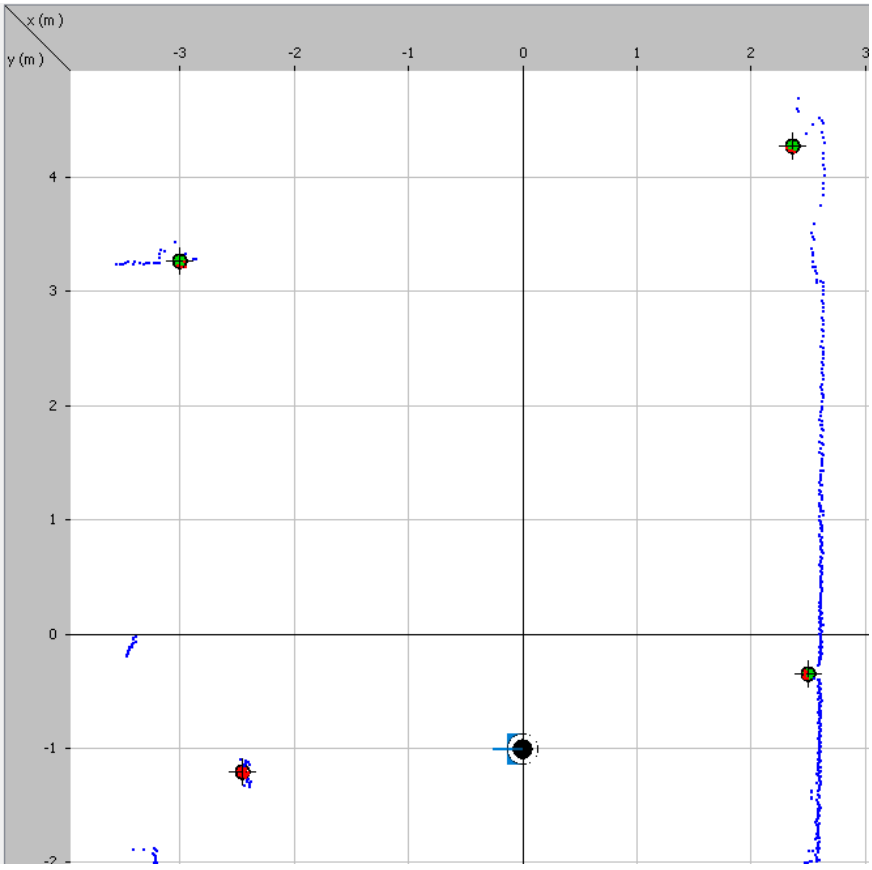


Abbildung 7-19 Position des NAV350 nach der Bewegung

7.3.3. Ergebnisse des Positionsvergleichs ohne Bewegung

Im Versuch soll gezeigt werden, wie sich die Messdaten sowie die Referenzposition der linearen Achse im Laufe der Zeit ohne Bewegung verhalten. Dazu wurde der Laserscanner auf die Position von 10 mm bewegt und während einer Stunde die Messdaten aufgenommen.

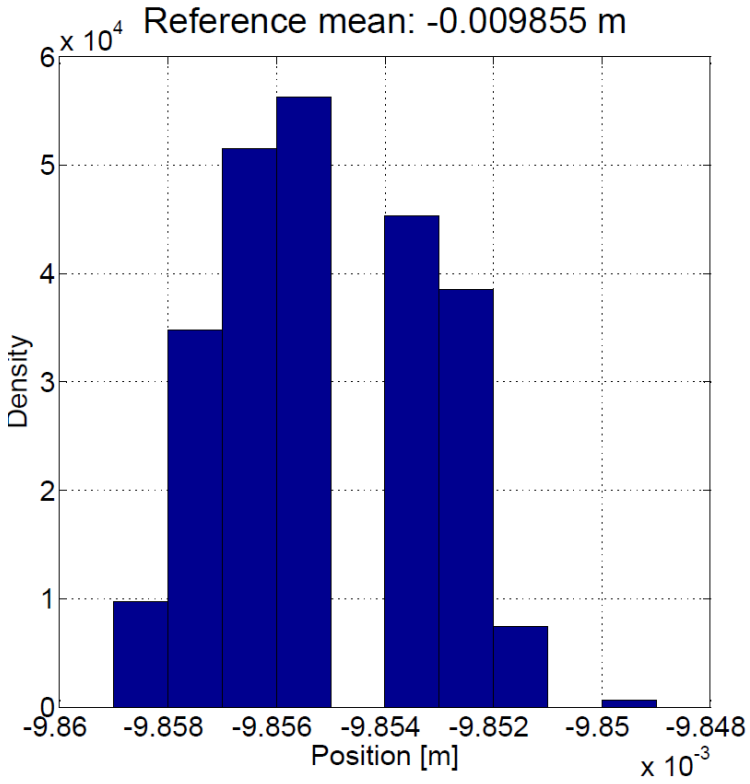


Abbildung 7-20 Referenzposition der linearen Achse

Die Abbildung 7-20 zeigt eine Verteilung der Referenzposition. Man erkennt, dass der Mittelwert von der Vorgabe um ca. 145 µm abweicht. Eine weitere Erkenntnis ist, dass die Referenzposition der linearen Achse normal verteilt ist und die Standardabweichung $\sigma = 2{,}013\ \mu m$ beträgt.

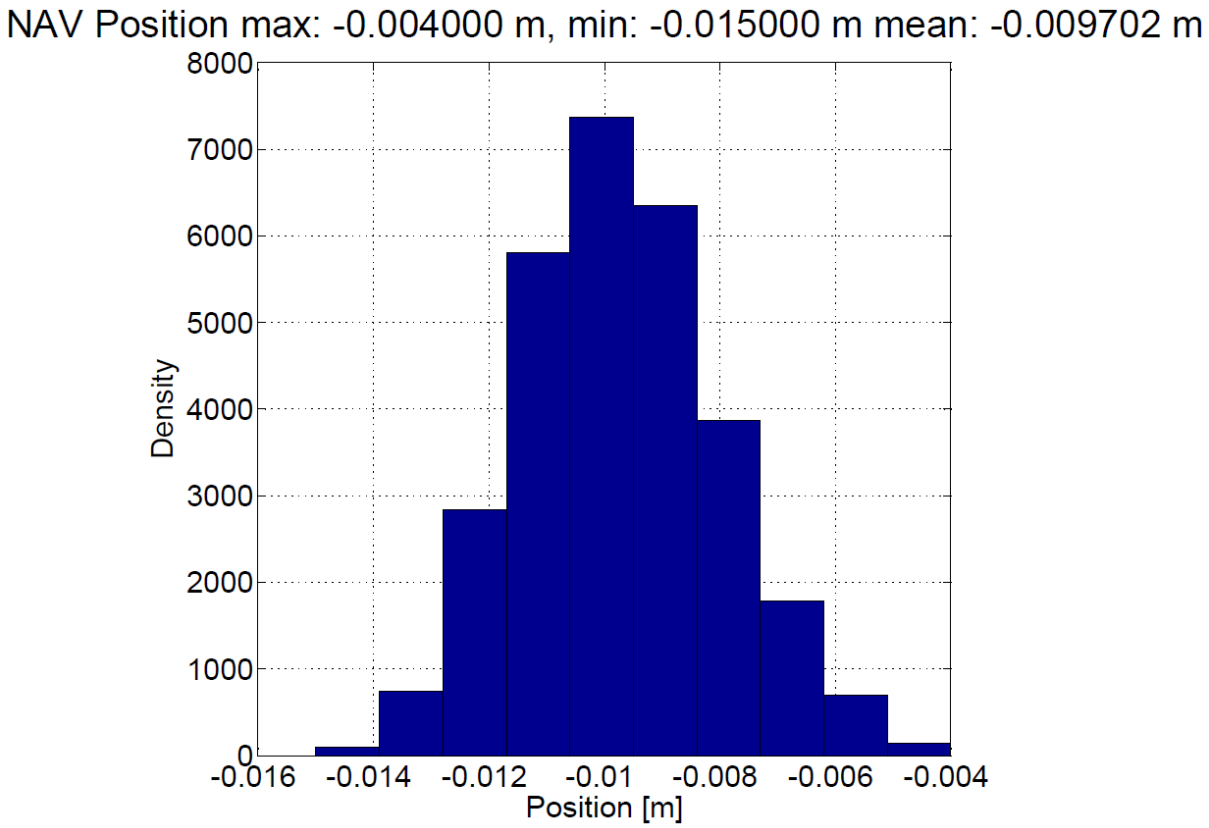


Abbildung 7-21 NAV Position ohne Bewegung

In Abbildung 7-21 kann man sehen, dass die NAV-Position normal verteilt ist und der Mittelwert von der Referenz um 150 µm abweicht. Die Standartabweichung der Verteilung beläuft sich auf 1,6 mm.

7.3.4. Ergebnisse der Testfahrt mit v = 200 mm/s

Die Testergebnisse mit einer Geschwindigkeit von 200 mm/s haben gezeigt, dass bei einer kleinen Dynamik die Referenzposition gleich der von NAV350 errechneten Position ist. Die Abbildung 7-22 zeigt, dass die gemessene Geschwindigkeit mit der Vorgabe übereinstimmt.

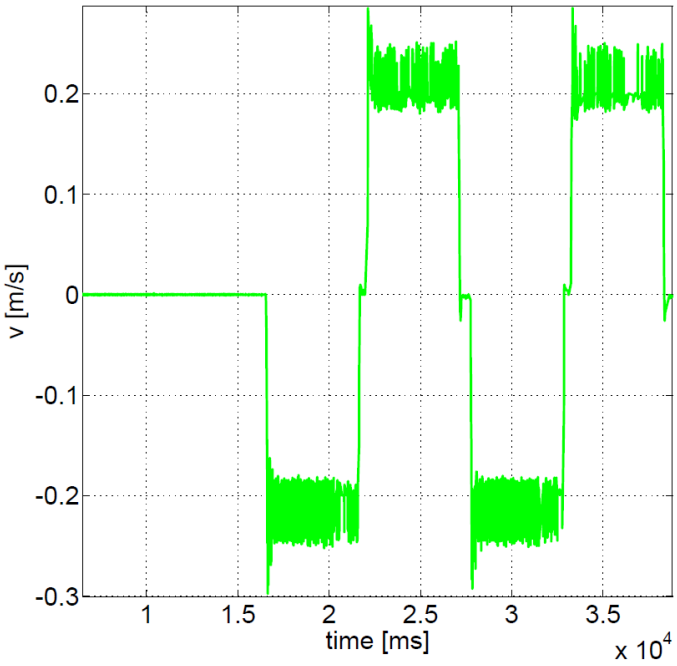


Abbildung 7-22 Geschwindigkeit-Zeit-Diagramm (v = 200mm/s)

In Abbildung 7-23 ist zu sehen, dass der Unterschied zwischen den Positionen mit bloßen Augen nicht erkennbar ist. Wenn man die Differenz zwischen den abgetasteten Werten bildet, erweist sich, dass der Mittelwert des Fehlers über 50 Fahrten mit der gleichen Geschwindigkeit ca. 1 cm beträgt.

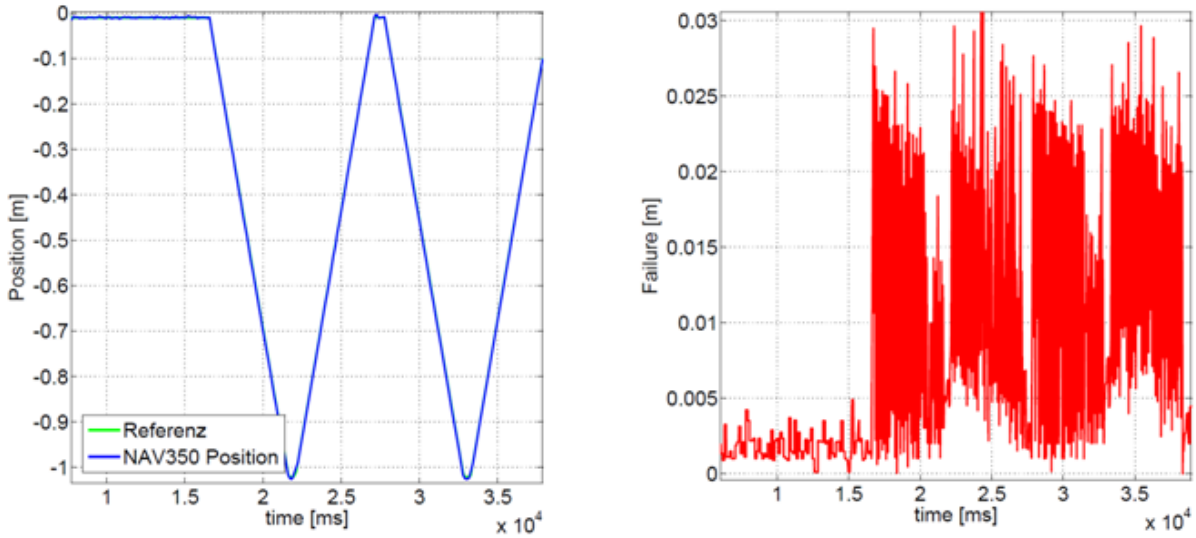


Abbildung 7-23 NAV350 <-> Referenz Vergleich (v = 200mm/s)

Der maximale Unterschied zwischen der NAV350 Position und der Referenz beträgt 3 cm, des Weiteren beläuft sich der minimale Fehler auf 0.

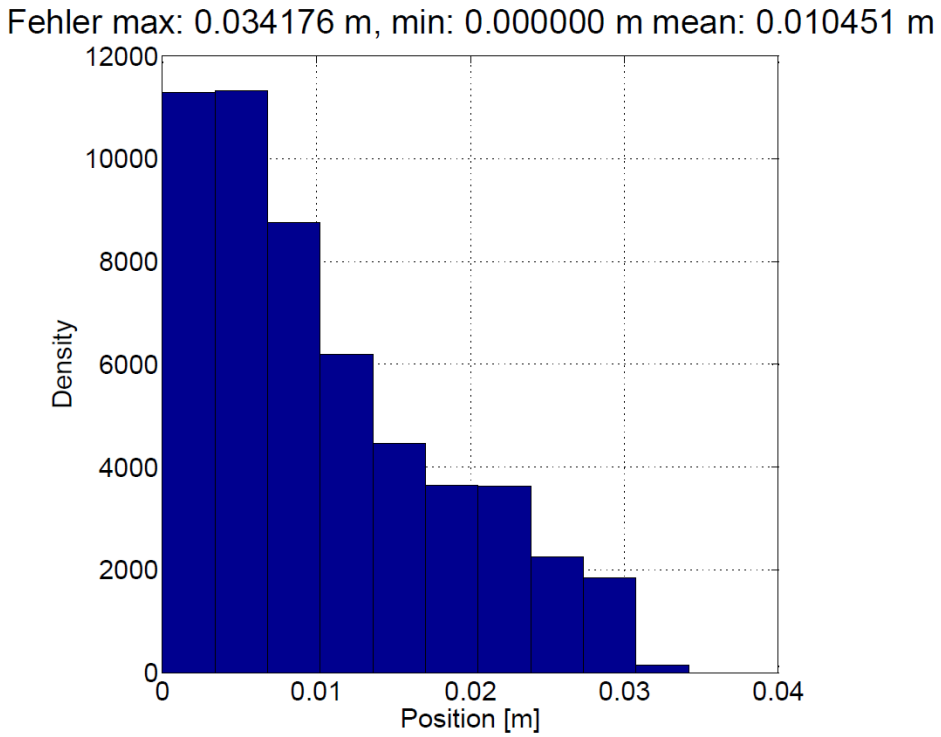


Abbildung 7-24 Verteilung des Fehlers (v = 200mm/s)

Eine Verteilung des Fehlers bei der Geschwindigkeit von 200 mm/s ist in der Abbildung 7-24 dargestellt. Die meiste Anzahl der gemessenen Werte liegt bei 0 und 5 mm, dennoch beträgt der Mittelwert der gesamten Messung 1 cm.

7.3.5. Ergebnisse der Testfahrt mit v = 1000 mm/s

Die Testergebnisse mit einer Geschwindigkeit von 1000 mm/s haben gezeigt, dass bei einer mittleren Dynamik die Referenzposition von der errechneten Position abweicht. Die Abbildung 7-22 bestätigt, dass die gemessene Geschwindigkeit mit der Vorgabe übereinstimmt.

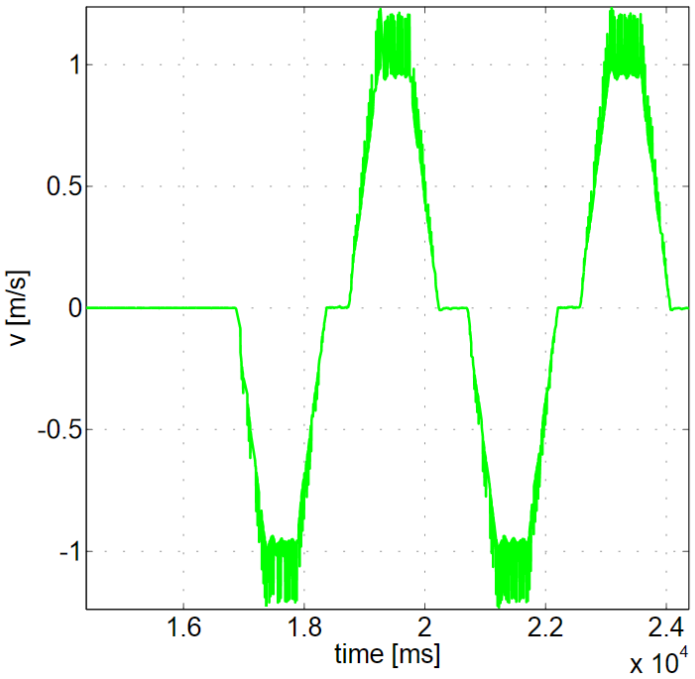


Abbildung 7-25 Geschwindigkeit-Zeit-Diagramm (v = 1000mm/s)

Abbildung 7-23 verdeutlicht, dass ein Unterschied zwischen den Positionen besteht. Wenn man die Differenz zwischen den abgetasteten Werten bildet, erkennt man, dass der Mittelwert des Fehlers über 50 Fahrten mit der gleichen Geschwindigkeit ca. 3 cm beträgt.

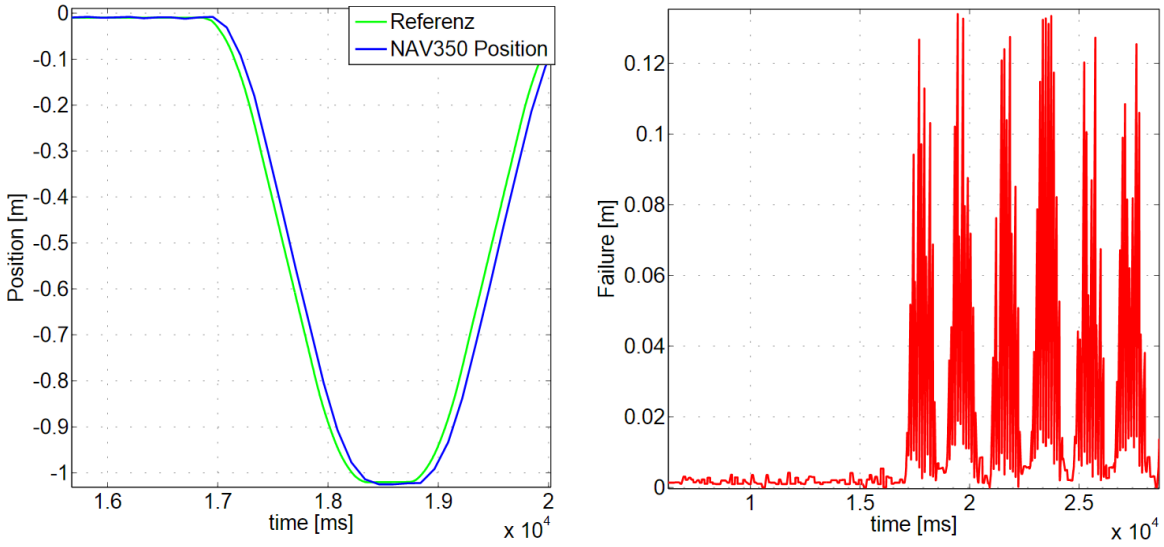


Abbildung 7-26 NAV350 <-> Referenz Vergleich (v = 1000mm/s)

Der maximale Unterschied zwischen der NAV350 Position und der Referenz beträgt 13,9 cm, der minimale Fehler ist weiterhin 0.

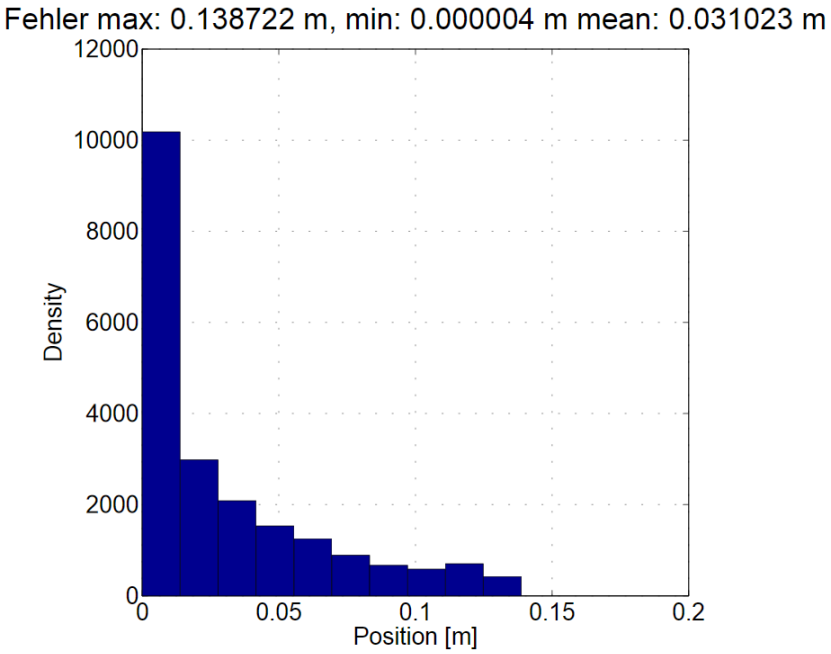


Abbildung 7-27 Verteilung des Fehlers (v = 1000mm/s)

Eine Verteilung des Fehlers bei der Geschwindigkeit von 1000 mm/s ist in der Abbildung 7-27 dargestellt. Die meiste Anzahl der gemessenen Werte liegt bei 0 mm, dennoch beträgt der Mittelwert der gesamten Messung 3 cm.

8. Fazit

8.1. Zusammenfassung

Im Rahmen der Masterarbeit wurden mehrere wissenschaftliche Arbeiten im Bereich des Genauigkeitsnachweises von Laserscanner analysiert. Alle untersuchten Arbeiten beinhalteten keine Information über den Nachweis der Genauigkeit im dynamischen Betreib.

Eine Entwicklung des Konzeptes für einen Positioniertisch ist nach einem V-Modell durchgeführt. Das Konzept ist durch eine genaue Untersuchung und Auswahl von Komponenten realisiert worden.

Als Nächstes wurde eine Steuerung mit dem Controller Development System V3.5 mit der SoftMotion Bibliothek entwickelt. Der Anwender kann über eine bedienfreundliche grafische Oberfläche beliebige Trajektorie definieren und anschließend auf dem Positioniertisch laufen lassen.

Da die entwickelte Steuerung lediglich auf einer virtuellen Ebene ausgeführt wird, konnten die Messungen mit einem Laserscanner nur simuliert werden. Die Auswertung der simulierten Daten wurde mithilfe des SLAM-Algorithmus durchgeführt. Die Auswertung der simulierten Daten hat ergeben, dass der Fehler außerhalb der Spezifikation liegt. Der Fehler nimmt mit der Steigerung der Geschwindigkeit zu.

Neben den gestellten Aufgaben – und zwar Entwicklung und Simulation der Steuerung eines dreiachsigen Positioniertisches für den Genauigkeitsnachweis von Laserscannern im dynamischen Betrieb – wurde die Realisierung eines Aufbaus in Form einer linearen Achse vorgenommen. Weitere Vergleiche der Laserscanner-Position mit der Referenz konnten mithilfe des Aufbaus stattfinden. Die Ergebnisse sind mit der Simulation vergleichbar. Man konnte ähnliche Zusammenhänge des Fehlerverhaltens ermitteln.

8.2. Ausblick

Die entwickelte Steuerung wurde auf einer virtuellen Ebene mit CoDeSys3.5 erstellt. Sie verwendet Funktionen und Funktionsbausteine der SoftMotion-Bibliothek, die mit virtuellen Antrieben arbeiten. Möchte man die Steuerung an der realen Ablage anwenden, müssen wenige Anpassungen durchgeführt werden. Die Anpassungen können sehr einfach implementiert werden.

Eine jener Anpassungen, welche bei der Realisierung beachtet werden soll, ist die Anbindung an die Steuerung eines CANopen-Masters, damit der Anschluss sowie die Ansteuerung der reellen Antriebe stattfinden können. Der CANopen-Master ermöglicht eine gleichzeitige Ansteuerung mehrerer Antriebe. Außer der Konfiguration des Datenbusses ist es essenziell, auf die Eigenschaften des Motorkontrollers zu achten. Der Kontroller soll einen linearen interpolierenden Betrieb haben. Mit dem Modus kann eine Bahnsteuerung in einer mehrachsigen Anwendung des Reglers realisiert werden. Dazu werden in einem festen Zeitraster Lagesollwerte von einer übergeordneten Steuerung vorgegeben. Wenn das Zeitraster der Lagesollwerte größer ist als die interne Lageregler-Zykluszeit des Motorcontrollers, interpoliert der Regler selbstständig die Datenwerte zwischen zwei vorgegebenen Lagesollwerten.

Die Masterarbeit hat gezeigt, dass man auch mit einer einfachen Simulation interessante Ergebnisse erzielen kann. Mit der Simulation erübrigte sich die Realisierung des Positioniertisches. Einer der Nachteile der Simulation ist die Laufzeit. Außer der Laufzeit, die in großer Ordnung nur nebenläufig ist, wurden während der Masterarbeit leichte Abweichungen im Zeitstempel entdeckt.

Literaturverzeichnis

Kapitel 1

[1] SICK AG: *LD Laser Scanners* Prospekt, 2004

[2] SICK, AG: Marketingmaterial der DIV08. 2013

[3] Gattringer, H.: *Starr-elastische Robotersystem,* Springer 2011

Kapitel 2

[4] Ye, C. und Borenstein, J.: *Characterization of a 2-D Laser Scanner for Mobile Robot Obstacle Negotiation,* 2002

[5] Boehler, W. und Marbs, A.: *Investigating Laser Scanner Accuracy,* 2002

[6] Bang, K. I., Habib, A. F., Müller, M.: *LIDAR System Calibration using overlapping strips,* 2009

Kapitel 3

[7] Bender, K.: *Enbedded System-qualitätsorientierte Entwicklung,* Springer 2005.

[8] Papula, Lothar: *Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler*.Wiesbaden : Vieweg-Verlag, 2006. – ISBN 3–8348–0156–9

[9] Wikipedia: *Arkustangens und Arkuskotangens*. Website. http://de.wikipedia.org/wiki/Arkustangens_und_Arkuskotangens. Version: Oktober 2012. – Abgerufen am 16.11.2012

[10] 3S: *Smart Software Solutions GmbH, Technische Dokumente und Broschüre*.2009

[11] Vogel-Heuser, Birgit ; Wannagat, Andreas: *Modulares Engineering und Wiederverwendung mit CoDeSys V3. Für Automatisierungslösungen mit objektorientiertem Ansatz.* München : Oldenburger Industrieverlag, 2009. –ISBN 978-3-8356-3105-2

[12] Seitz, Matthias: *Speicherprogrammierbare Steuerungen. System- und Programmentwurf für die Fabrik- und Prozessautomatisierung, vertikale Integration.* München.2. Auflage : Carl Hanser Verlag, 2008. – ISBN 978-3-446-41431-0

[13] Neier, J. H.: *Automatische Erstellung von hochgenauen Umgebungskarten für fahrerlose Transportfahrzeuge,* Fachhochschule Münster, Masterthesis 2012

[14] SICK AG (Hrsg.): *Betriebsanleitung, Laser-Positioniersensor NAV350*. http://www.sick.de: SICK AG, 2011

[15] Kiel, E.: *Antriebslösungen. Mechatronik für Produktion und Logistik,* Springer 2007

[16] Wellenreuther G., Zastrow D.: *Automatisieren mit SPS – Theorie und Praxis,* Vieweg + Teubner 2011

[17] Petry, J.: *IEC 61131-3 mit CoDeSys v3: Ein Praxisbuch für SPS-Programmierer,* 3S-SmartSoftwareSolution GmbH 2011

Kapitel 7

[18] Tieben, C.: *Scandatensimulation aus einer 3D-Scenerie mittels Raytracing,* Hochschule für angewandte Wissenschaften Hamburg, Bachelorthesis 2013

Anhang

A. Steuerungsprogramm CoDeSys V3.5

1. Strukturen

Um die Trajektorien im G-Code-Format verarbeiten und in einem bestimmten Verzeichnis speichern zu können, wurden zwei Strukturen angelegt, die wie folgt definiert sind:

```
TYPE Directory :
    STRUCT
          iNumberOfEntries : INT ; (*Quantity of entries *)
          asFiles : ARRAY [ 0 .. 63 ] OF STRING ; (*Array with all defined trajectories*)
          iSelectedEntry : INT ; (*pointer to each trajectory*)
          xSelectionValid : BOOL ; (*detection if the selected cnc-file is opened or closed*)
    END_STRUCT
END_TYPE
```

Die “Directory”-Struktur ist zuständig, um den Speicherort der einzelnen CNC-Datei aufzurufen und außerdem alle gespeicherten Dateien anzuzeigen.

Weitere Struktur ist die „TexteditorControl“. Die Struktur ist allein für die Erstellung, die Verarbeitung, das Öffnen, das Schließen und das Speichern der einzelnen CNC-Datei zuständig.

```
TYPE TexteditorControl :
    STRUCT
       sFilename : STRING ;
       diSelectLine : DINT ;
       xOpen : BOOL ;
       xSave : BOOL ;
       xNew : BOOL ;
       xClose : BOOL ;
       xSelectLine : BOOL ;
    END_STRUCT
END_TYPE
```

2. Globale Variablen

Eine globale Instanziierung der Funktionen und Deklaration der Variablen sind nötig, um die Kommunikation und Datenaustausch zwischen den einzelnen Programmen zu ermöglichen.

```
VAR_GLOBAL
(*Global variables for Devices*)
lrMaxLengthX : LREAL := 4000 ;      (*Max Length X Drive*)
```

```
4  lrMaxLengthY : LREAL := 6000 ;      (*Max Length Y Drive*)
5  lrMaxAngleW : LREAL := 360 ;  (*Max Length W Drive*)
6
7  lrPositionX : LREAL ;  (*Actual position of X drive*)
8  lrPositionY : LREAL ;  (*Actual position of Y drive*)
9  lrPositionW : LREAL ;  (*Actual position of W drive*)
10
11 bEmergencyStop : BOOL ;
12
13 _sGenerateNewFile : STRING ;
14
15 _dirData : Directory ;
16 _xUpdateDir : BOOL := TRUE ;
17
18 _tecData , _tecData2 : TexteditorControl ;  (*_tecData is available for
programming mode and _tecData2 for execution mode*)
19
20 _xStartExecution , _xWaitExecution , _xAbortExecution : BOOL ;
21 _xRunningExecution , _xWaitingExecution , _xErrorExecution : BOOL ;
22
23 _ipo : SMC_Interpolator ;
24 _rncfBuffer : ARRAY [ 0 .. 99 ] OF SMC_GCODE_WORD ;
25 _decBuffer : ARRAY [ 0 .. 99 ] OF SMC_GEOINFO ;
26
27 _pt : SMC_PositionTracker ;
28 _trafo :  SMC_TRAFO_GantryCutter2 ;
29 _trafof :       SMC_TRAFOF_GantryCutter2 ;
30 _bufferTracker : ARRAY [ 0 .. 199 ] OF VisuStruct3DPathPoint ;
31
32 _iStatus : INT ;
33
34 _vc : VisuStruct3DControl ;
35 _vc2 : VisuStruct3DControl ;
36
37 _bufferPath : ARRAY [ 0 .. 3999 ] OF VisuStruct3DPathPoint ;
38 _vs3dt : VisuStruct3DTrack ;
39 _bufferPath2 : ARRAY [ 0 .. 3999 ] OF VisuStruct3DPathPoint ;
40 _vs3dt2 : VisuStruct3DTrack ;
41
42 _diActLine : DINT ;
43
44 // instances for motion control
45 _mcpX , _mcpY , _mcpW : MC_Power ;
46 _mchDeceleration : LREAL := 10000 ;
47 _mchX , _mchY , _mchW : MC_Halt := ( Deceleration := 1000 ) ;
48 _maX , _maY , _maW : MC_MoveAbsolute ;
49 _bInStartPosition : BOOL ;
50 _bInStartPosition_blink : BOOL ;
51
52 // instances for trace handling
53  bStartTracePosCommand  ,  bStartTraceVelCommand  ,  bStartTraceAccCommand
:BOOL := FALSE;
54 bWriteTracePositionCommand : BOOL := FALSE ;
55 bStartTrace , bStartTraceWCommand : BOOL := FALSE ;
56 lrPosition , _lrPositionX , _lrPositionY : LREAL := 0 ;
57 lrVelocity , lrVelocityX , lrVelocityY : LREAL := 0 ;
58 lrAccelerationX , lrAccelerationY : LREAL := 0 ;
59
60
61
62 // instances for file handling
63
64 _CNCfileDirectory : STRING := './_cnc/' ;
```

```
65 _xGeneratePath : BOOL ;
66 _pcf : SMC_PathCopierFile ;
67
68 // Velocity Check
69 lrMaxVelocity , lrMaxVelocity_new : LREAL := 0 ;
   70 END_VAR
```

3. Funktionen und Funktionsblöcke

Alle definierte Funktionen und Funktionsblöcke werden im Kapitel mit deren Variablendeklarationen und Eigenschaften beschrieben.

GenerateNewFile-Funktion

Beim Aufruf der Funktion wird eine neue CNC-Datei generiert.

```
1 FUNCTION GenerateNewFile
2 VAR
3       h : RTS_IEC_HANDLE ;
4       result : RTS_IEC_RESULT ;
5 END_VAR
```

Das Programm hat den Zugriff auf die globale Variable _tecData. Somit wird eine neue Datei unter dem vorgegebenen Pfad(_CNCfileDirectory).

```
1 _tecData . sFilename := concat ( _CNCfileDirectory , _sGenerateNewFile ) ;
2 _tecData . sFilename := concat ( _tecData . sFilename , '.cnc' ) ;
3
4 h := SysFileOpen ( _tecData . sFilename , SysFile . AM_READ , ADR ( result ) ) ;
5 IF h <> RTS_INVALID_HANDLE THEN
6       SysFileClose ( h ) ;
7       h := 0 ;
8       _tecData . sFilename := '' ;
9
10 ELSE
11 h := SysFileOpen ( _tecData . sFilename , SysFile . AM_WRITE , ADR ( result) ) ;
12      IF h <> RTS_INVALID_HANDLE THEN
13              SysFileClose ( h ) ;
14              h := 0 ;
15              _tecData . xNew := TRUE ;
16      ELSE
17              _tecData . sFilename := '' ;
18
19      END_IF
20 END_IF
21
22 _tecData . xClose := TRUE ;
23 _xUpdateDir := TRUE ;
24 _xGeneratePath := TRUE ;
```

GeneratePathExec- und GeneratePathProg-Funktionen

Die GeneratePathExec- und GeneratePathProg-Funktionen sind identisch und unterscheiden sich nur in der Stelle des Aufrufes und im Zugriff auf unterschiedlichen Variablen. Die beiden Funktionen machen möglich die erstellte Trajektorie zu analysieren. Die Trajektorie wird im kartesischen Koordinatensystem gezeichnet(siehe Kapitel B.3 Visualisierung Automatic Mode).

```
1 _pcf (
2         bExecute := FALSE ) ;
3
4 WHILE NOT ( _pcf . bDone OR _pcf . bError ) DO
5 _pcf (
6         bExecute := TRUE ,
7         udiNumberOfPointsInArray := SIZEOF ( _bufferPath2 ) / SIZEOF (_bufferPath2 [ 0 ] ) ,
8         pBuffer := ADR ( _bufferPath2 ) ,
9 pvl := ,
10        piStartPosition := ,
11        bDone => ,
12        bError => ,
13        iErrorID => ,
14        vs3dt => ) ;
15 END_WHILE
16
17 _vs3dt2 := _pcf . vs3dt ;
```

ASCII-Funktion

Funktion ASCII ist zuständig die Systemzeit und das Systemdatum, die im ASCII-Format sind, in die Zahlen im String-Format umzuwandeln.

Deklaration der Variablen:

```
1 FUNCTION ASCII : STRING
2 VAR_INPUT
3         _byte : BYTE ;
4 END_VAR
5 VAR_OUTPUT
6         _sSign : STRING ;
7 END_VAR
8
9 VAR
10
11END_VAR
```

Die eigentliche Umwandlung des einzelnen ASCII-Zeichens ist mit einer CASE-Anweisung realisiert.

```
1 CASE _byte OF
2 48 :
3         _sSign := '0' ;
4 49 :
5         _sSign := '1' ;
```

```
50 :
        _sSign := '2' ;
51 :
        _sSign := '3' ;
52 :
        _sSign := '4' ;
53 :
        _sSign := '5' ;
54 :
        _sSign := '6' ;
55 :
        _sSign := '7' ;
56 :
        _sSign := '8' ;
57 :
        _sSign := '9' ;

END_CASE
```

ROUND-Funktion

Um das Aufrunden der der beliebiegen Werte zu ermöglichen, exisiert eine ROUND-Funktion.

```
FUNCTION ROUND : LREAL
VAR_INPUT
        _number : LREAL ;
        _nks : DINT ;
END_VAR
VAR
        _inumber : UDINT ;
END_VAR
VAR_OUTPUT
        _rnumber : LREAL ;
END_VAR
```

Funktion hat als Eingänge der LREAL-Wert und die DINT-Stelle. Die Variable _number ist die Zahl, die aufgerundet werden soll und die Variable _nks ist die Anzahl der gewünschten Nachkommastellen.

```
_number := _number * _nks ;
_inumber := REAL_TO_UDINT ( _number ) ;
_rnumber := UDINT_TO_REAL ( _inumber ) ;
_rnumber := _rnumber / _nks ;
```

SizeOfString-Funktion

```
FUNCTION SizeOfString
VAR_INPUT
        _sString : STRING ;
END_VAR
VAR_OUTPUT
        Size : UDINT ;
```

```
END_VAR
VAR
        bByte : BYTE ;
        i : UDINT ;
END_VAR
```

Der Eingang der Funktion ist ein String. Die Länge des Strings wird mit Hilfe der Funktion errechnet.

```
FOR i := 0 TO 40 BY 1 DO
        bByte := _sString [ i ] ;
        IF bByte = 10 THEN // 10 ASCII dec for LF
                Size := i + 1 ; (*Size of String plus NewLine*)
                i := 40 ;
        END_IF
END_FOR
```

DateToString-Funktionsblock

Der Funktionsblock dient zur einfachen Konvertierung einer Variablen im DT-Format(Date-Format) in einen String.

```
FUNCTION_BLOCK DateToString
VAR_INPUT
        _date : DT ;
        _bExecute : BOOL ;
END_VAR
VAR
        _sDate : STRING ;
        _sMinus : STRING := '-' ;
        _sASCII : STRING ;
END_VAR
VAR_OUTPUT
        _sFileDate : STRING := '' ;
END_VAR
```

Programm:

```
_sDate := DT_TO_STRING ( _date ) ;
IF _bExecute THEN
        //year
        ASCII ( _byte := _sDate [ 5 ] , _sSign => _sASCII ) ;
        _sFileDate := concat ( _sFileDate , _sASCII ) ;
        ASCII ( _byte := _sDate [ 6 ] , _sSign => _sASCII ) ;
        _sFileDate := concat ( _sFileDate , _sASCII ) ;
        //month
        ASCII ( _byte := _sDate [ 8 ] , _sSign => _sASCII ) ;
        _sFileDate := concat ( _sFileDate , _sASCII ) ;
        ASCII ( _byte := _sDate [ 9 ] , _sSign => _sASCII ) ;
        _sFileDate := concat ( _sFileDate , _sASCII ) ;
        //day
        ASCII ( _byte := _sDate [ 11 ] , _sSign => _sASCII ) ;
        _sFileDate := concat ( _sFileDate , _sASCII ) ;
```

```
16        ASCII ( _byte := _sDate [ 12 ] , _sSign => _sASCII ) ;
17        _sFileDate := concat ( _sFileDate , _sASCII ) ;
18        _sFileDate := concat ( _sFileDate , _sMinus ) ;
19        //time
20        //hour
21        ASCII ( _byte := _sDate [ 14 ] , _sSign => _sASCII ) ;
22        _sFileDate := concat ( _sFileDate , _sASCII ) ;
23        ASCII ( _byte := _sDate [ 15 ] , _sSign => _sASCII ) ;
24        _sFileDate := concat ( _sFileDate , _sASCII ) ;
25        //minutes
26        ASCII ( _byte := _sDate [ 17 ] , _sSign => _sASCII ) ;
27        _sFileDate := concat ( _sFileDate , _sASCII ) ;
28        ASCII ( _byte := _sDate [ 18 ] , _sSign => _sASCII ) ;
29        _sFileDate := concat ( _sFileDate , _sASCII ) ;
30        //seconds
31        ASCII ( _byte := _sDate [ 20 ] , _sSign => _sASCII ) ;
32        _sFileDate := concat ( _sFileDate , _sASCII ) ;
33        ASCII ( _byte := _sDate [ 21 ] , _sSign => _sASCII ) ;
34        _sFileDate := concat ( _sFileDate , _sASCII ) ;
35        _sFileDate := concat ( _sFileDate , '_' ) ;
36  END_IF
37  IF NOT  _bExecute THEN
38        _sFileDate := '' ;
39  END_IF
```

ConcatActPos-Funktionsblock

Der Funktionsblock „ConcatActPos“ ist für die Konkatenation (Verkettung) der aktuellen Position und Orientierung in einen String zuständig.

Deklaration der Variablen:

```
1  FUNCTION_BLOCK  ConcatActPos
2  VAR_INPUT
3     Enable : BOOL  ;
4     _lrX , _lrY , _lrW : LREAL  ;
5     _time : STRING  ;
6  END_VAR
7  VAR
8     _sSpace : STRING  := '$T' ;
9     _sNewLine : STRING  := '$R$N' ;
10    _roundi : DINT  := 1000 ;
(*round to X decimal places=> 10 : X=1 ; 100 : X=2...*)
11 END_VAR
12 VAR_OUTPUT
13     _sPos : STRING  ;
14    Size : UDINT  ;
15 END_VAR
```

Der Funktionsblock hat als Eingang die aktuellen Positionen der einzelnen Antriebe und die Orientierung als LREAL-Wert. Der Werte werden als nächstes in die Strings konvertiert und

anschließend werden die einzelne Strings in einem neuen String miteinander mit Hilfe einer concat-Funktion[8] verknüpft.

```
IF Enable THEN
    ROUND ( _number := _lrX , _nks := _roundi , _rnumber => _lrX ) ;
    ROUND ( _number := _lrY , _nks := _roundi , _rnumber => _lrY ) ;
    ROUND ( _number := _lrW , _nks := _roundi , _rnumber => _lrW ) ;
    _sPos := REAL_TO_STRING ( _lrX ) ;
    _sPos := concat ( _sPos , _sSpace ) ;
    _sPos := concat ( _sPos , REAL_TO_STRING ( _lrY ) ) ;
    _sPos := concat ( _sPos , _sSpace ) ;
    _sPos := concat ( _sPos , REAL_TO_STRING ( _lrW ) ) ;
    _sPos := concat ( _sPos , _sSpace ) ;
    _sPos := concat ( _sPos , _time ) ;
    _sPos := concat ( _sPos , _sNewLine ) ;
    SizeOfString (
                    _sString := _sPos ,
                    Size => Size ) ;
END_IF
```

4. ManualMode

Das Programm ist für die manuelle Bedienung der Antriebe zuständig.

```
PROGRAM ManualControl
VAR
    (*W Drive*)
    bMoveWpos : BOOL := FALSE ;
    bMoveWneg : BOOL := FALSE ;
    bExecuteWneg : BOOL ;
    bExecuteWpos : BOOL ;
    mvW : MC_MoveVelocity ;
    amcDirW : ARRAY [ 1 .. 2 ] OF MC_Direction := [ Positive , Negative ] ;
    lrPosW : LREAL := 0 ;
    lrManualMoveVelW : LREAL := 10 ;

    (*X Drive*)
    lrPosX : LREAL ;
    lrManualMoveVelX : LREAL := 10 ;
    bMoveXpos , bMoveXneg : BOOL := FALSE ;
    bExecuteXneg : BOOL ;
    bExecuteXpos : BOOL ;

    (*Y Drive*)
    lrPosY : LREAL ;
    lrManualMoveVelY : LREAL := 10 ;
    bMoveYpos , bMoveYneg : BOOL := FALSE ;
    bExecuteYneg : BOOL ;
    bExecuteYpos : BOOL ;

    (**)
    bExecute : BOOL ;
```

[8] concat-Funktion ermöglicht die Konkatenation (von lateinisch catena „Kette“) oder Verknüpfung zweier Stings in einem neuen String

```
29      bFolPosVel : BOOL ;
30      bMoveXYWDone : BOOL ;
31      bMoveXYW : BOOL ;
32      bFolPosVelDone : BOOL ;
33 END_VAR
```

Der Anwender ist mit dem Programm in der Lage die interessierenden Punkte mit gewünschter Geschwindigkeit anzufahren. Das Programm ist in KOP(Kontaktplan) erstellt. Die Struktur gibt die Sicherheit, dass während dem manuellen Betrieb keine CNC-Trajektorie verfahren wird.

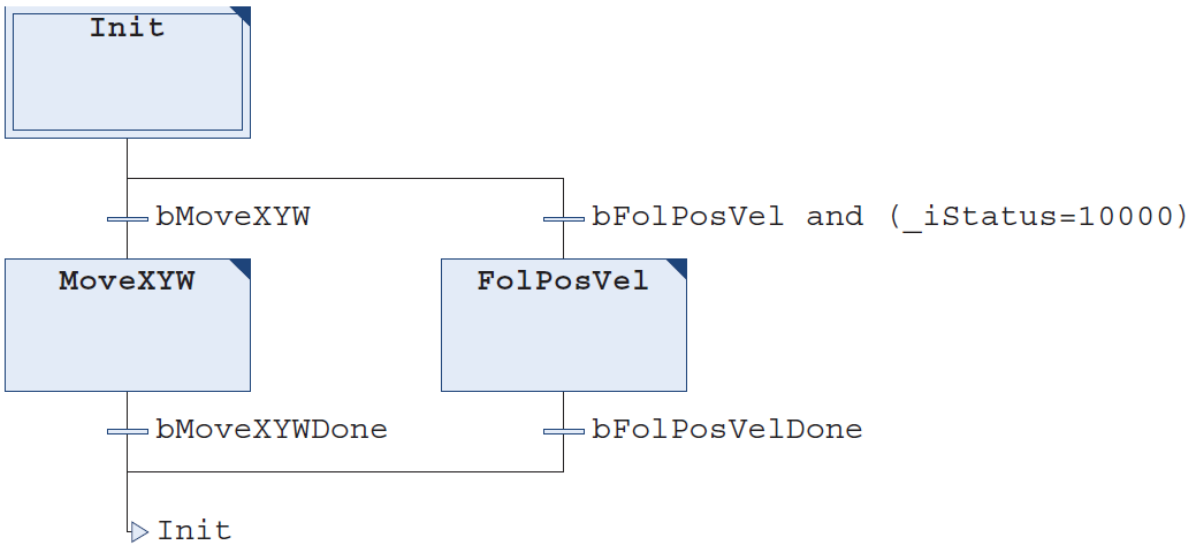


Abbildung 8-1 KOP Programm des manuellen Betriebes

Abbildung 8-1 stellt das Programm des manuellen Betriebes grafisch dar.

5. Automatic Mode

Es ist möglich im automatischen Betrieb die Trajektorien zu definieren und zu verfahren.

```
1  PROGRAM  AutomaticMode
2  VAR_INPUT
3          _timeToGetSample : BOOL ;
4  END_VAR
5  VAR
6  //instances for motion
7          _cbpvX , _cbpvY , _cbpvW : SMC_ControlAxisByPosVel ;
8          _cbpW : SMC_ControlAxisByPos ;
9          _bWconstant : BOOL  := FALSE ;
10         _lrWposition : LREAL  := 0 ;
11         _diLineInFile : DINT  ;
12        lrGapVelocity : LREAL  := 1000 ;
13        lrGapAcceleration : LREAL  := 2000 ;
14        _VelMode : SMC_INT_VELMODE := 0 ;
15
16        _lrVelToStartPosition : LREAL  := 2000 ;
17        _lrAccToStartPosition : LREAL  := 3500 ;
18        _lrDecToStartPosition : LREAL  := 3500 ;
```

```
19        LimitSafety : LREAL := 0 ; (*mm safety distance from end of axis*)
20
21        _lrStartPosition : LREAL := 0 ;
22
23        (*Timer*)
24        _bSaveFile : BOOL ;
25        _tp1 , _tp2 : TP ;
26        _SamplingTime : TIME := T#125MS ;
27        _sSamplingTime : STRING ;
28        _modTime : TIME ;
29        triger : BOOL ;
30        SamplePosX : ARRAY [ 0 .. 50 ] OF LREAL ;
31        SamplePosY : ARRAY [ 0 .. 50 ] OF LREAL ;
32        SamplePosW : ARRAY [ 0 .. 50 ] OF LREAL ;
33        SampleTime : ARRAY [ 0 .. 50 ] OF TIME ;
34        _SizeOfMem : UINT := 50 ;
35        SamplePointer : UINT := 0 ;
36        _bFirstSample : BOOL := TRUE ;
37        _sample : BOOL := TRUE ;
38
39 END_VAR
```

Der Ablauf des Programms ist in Form einer CASE-Anweisung realisiert.

```
1 //power
2 _mcpX (
3        Enable := TRUE ,
4        bRegulatorOn := TRUE ,
5        bDriveStart := TRUE ,
6        Axis := X_Axis ,
7        Status => ,
8        bRegulatorRealState => ,
9        bDriveStartRealState => ,
10       Busy => ,
11       Error => ,
12       ErrorID => ) ;
13 _mcpY (
14       Enable := TRUE ,
15       bRegulatorOn := TRUE ,
16       bDriveStart := TRUE ,
17       Axis := Y_Axis ,
18       Status => ,
19       bRegulatorRealState => ,
20       bDriveStartRealState => ,
21       Busy => ,
22       Error => ,
23       ErrorID => ) ;
24 _mcpW (
25       Enable := TRUE ,
26       bRegulatorOn := TRUE ,
27       bDriveStart := TRUE ,
28       Axis := W_Axis ,
29       Status => ,
30       bRegulatorRealState => ,
31       bDriveStartRealState => ,
32       Busy => ,
```

```
33      Error => ,
34      ErrorID => ) ;
35 //end power
36
37 CASE _iStatus OF
38 0 : //idle
39      _sSamplingTime := TIME_TO_STRING ( _SamplingTime ) ;
40
41      IF ( X_Axis . fSetPosition > 0 ) OR ( Y_Axis . fSetPosition > 0 ) OR (
W_Axis . fSetPosition > 0 ) THEN
42              _iStatus := 1001 ;
43      END_IF
44      IF _xStartExecution AND NOT _bInStartPosition THEN
45              _xStartExecution := FALSE ;
46              _bInStartPosition := FALSE ;
47
48              _ipo ( bExecute := FALSE ) ;
49
50              _pt (
51                      bEnable := FALSE ,
52                      bClear := TRUE ) ;
53
54              _xErrorExecution := FALSE ;
55
56              //controlaxisbyposition
57              _cbpvX (
58                      iStatus := _ipo . iStatus ,
59                      bEnable := FALSE ,
60                      bAvoidGaps := TRUE ,
61                      fSetPosition := _trafo . dx ,
62                      fSetVelocity := _ipo . dVel ,
63                      fGapVelocity := lrGapVelocity ,
64                      fGapAcceleration := lrGapAcceleration ,
65                      fGapDeceleration := lrGapAcceleration ,
66                      Axis := X_Axis ,
67                      bBusy => ,
68                      bCommandAborted => ,
69                      bError => ,
70                      iErrorID => ,
71                      bStop_ipo => ) ;
72              _cbpvY (
73                      iStatus := _ipo . iStatus ,
74                      bEnable := FALSE ,
75                      bAvoidGaps := TRUE ,
76                      fSetPosition := _trafo . dy ,
77                      fSetVelocity := _ipo . dVel ,
78                      fGapVelocity := lrGapVelocity ,
79                      fGapAcceleration := lrGapAcceleration ,
80                      fGapDeceleration := lrGapAcceleration ,
81                      Axis := Y_Axis ,
82                      bBusy => ,
83                      bCommandAborted => ,
84                      bError => ,
85                      iErrorID => ,
86                      bStop_ipo => ) ;
87              _cbpW (
```

```
88                      iStatus := _ipo . iStatus ,
89                      bEnable := FALSE ,
90                      bAvoidGaps := FALSE ,
91                      fSetPosition := _trafo . dr ,
92                      fGapVelocity := 500 ,
93                      fGapAcceleration := 360 ,
94                      fGapDeceleration := 360 ,
95                      Axis := W_Axis ,
96                      bBusy => ,
97                      bCommandAborted => ,
98                      bError => ,
99                      iErrorID => ,
100                     bStop_ipo => ) ;
101             //end controlaxisbypos
102
103             _iStatus := 1 ;
104     END_IF
105
106 10 : //normal operation
107     _ipo (
108             bExecute := TRUE ,
109             poqDataIn := InitCNC . _cv . poqDataOut ,
110             bSlow_Stop := _xWaitExecution ,
111             bEmergency_Stop := ,
112             bWaitAtNextStop := ,
113             iVelMode := _VelMode ,
114             dwipoTime := LREAL_TO_DWORD ( X_Axis . fTaskCycle * 1E6 ) ,
115             dLastWayPos := ,
116             bAbort := ,
117             bSingleStep := ,
118             bAcknM := ,
119             bQuick_Stop := ,
120             dQuick_deceleration := ,
121             bDone => ,
122             bBusy => ,
123             bError => ,
124             wErrorID => ,
125             piSetPosition => ,
126             _iStatus => ,
127             bWorking => ,
128             iActObjectSourceNo => ,
129             dVel => ,
130             vecActTangent => ,
131             iLastSwitch => ,
132             dwSwitches => ,
133             dWayPos => ,
134             wM => ,
135             Vel_Start => ) ;
136
137     _xErrorExecution := _xErrorExecution OR _ipo . bError ;
138
139     _pt (
140             bEnable := TRUE ,
141             bClear := FALSE ,
142             dX := _trafo . dx ,
143             dY := _trafo . dy ,
```

```
144             udiNumberOfPointsInArray := SIZEOF ( _bufferTracker ) / SIZEOF
(_bufferTracker [ 0 ] ) ,
145             pBuffer := ADR ( _bufferTracker ) ,
146             _vs3dt => ) ;
147     _trafo (
148             pi := _ipo . piSetPosition ,
149             v := _ipo . vecActTangent ) ;
150     _trafof (
151             DriveR := W_axis ,
152             DriveX := X_Axis ,
153             DriveY := Y_Axis ,
154             minX := 0 ,
155             maxX := lrMaxLengthX ,
156             minY := 0 ,
157             maxY := lrMaxLengthY ,
158             dx => lrPositionX ,
159              dy => lrPositionY ,
160             dr => lrPositionW ) ;
161
162     _xErrorExecution := _xErrorExecution OR _cbpvX . bError OR _cbpvY .bError OR
_cbpW . bError ;
163
164
165     // control if the must positon from trafo higher as max length of axis
166     IF ( _trafo . dx > lrMaxLengthX - LimitSafety ) THEN
167             _cbpvX (
168             iStatus := _ipo . iStatus ,
169             bEnable := ( _iStatus = 10 ) ,
170             bAvoidGaps := TRUE ,
171             fSetPosition := lrMaxLengthX - LimitSafety ,
172              fSetVelocity := _ipo . dVel ,
173              fGapVelocity := lrGapVelocity ,
174              fGapAcceleration := lrGapAcceleration ,
175             fGapDeceleration := lrGapAcceleration ,
176             Axis := X_Axis ,
177             bBusy => ,
178             bCommandAborted => ,
179             bError => ,
180             iErrorID => ,
181             bStop_ipo => ) ;
182      END_IF
183
184     // control if the must positon from trafo higher as max length of axis
185     IF ( _trafo . dy > lrMaxLengthY - LimitSafety ) THEN
186             _cbpvY (
187                     iStatus := _ipo . iStatus ,
188                     bEnable := ( _iStatus = 10 ) ,
189                     bAvoidGaps := TRUE ,
190                      fSetPosition := lrMaxLengthY - LimitSafety ,
191                     fSetVelocity := _ipo . dVel ,
192                     fGapVelocity := lrGapVelocity ,
193                     fGapAcceleration := lrGapAcceleration ,
194                     fGapDeceleration := lrGapAcceleration ,
195                     Axis := Y_Axis ,
196                     bBusy => ,
197                     bCommandAborted => ,
```

```
198                     bError => ,
199                     iErrorID => ,
200                     bStop_ipo => ) ;
201     END_IF
202
203     IF _ipo . ACT_OBJECT <> 0 THEN
204             _diLineInFile := _ipo . ACT_OBJECT ^ . iSourceLine_No ;
205     END_IF
206     IF NOT _tecData2 . xSelectLine AND _tecData2 . diSelectLine <>_diLineInFile THEN
207             _tecData2 . diSelectLine := _diLineInFile ;
208             _tecData2 . xSelectLine := TRUE ;
209     END_IF
210
211     //Sampling of act. position
212     IF _bSaveFile THEN
213             _tp1 ( //Timer for completelly motion
214                     IN := TRUE ,
215                     PT := T#7D ) ;
216             _tp2 ( //Tiimer for user defined sampling time
217                     IN := TRUE ,
218                     PT := _SamplingTime ,
219                     ET => ) ;
220             IF _tp2 . ET = _tp2 . PT AND _bFirstSample THEN
221                     _bFirstSample := FALSE ;
222                     _timeToGetSample := FALSE ;
223                     _tp2 ( IN := FALSE , PT := T#0MS ) ;
224                     SamplePosX [ SamplePointer ] := X_Axis . fActPosition ;
225                     SamplePosY [ SamplePointer ] := Y_Axis . fActPosition ;
226                     SamplePosW [ SamplePointer ] := W_Axis . fActPosition ;
227                     SampleTime [ SamplePointer ] := _tp1 . ET ;
228                     _timeToGetSample := TRUE ;
229                     SamplePointer := SamplePointer + 1 ;
230             END_IF

231             IF _tp2 . ET >= _tp2 . PT - T#1MS AND NOT _bFirstSample AND _sample
THEN

232                     _sample := FALSE ;
233                     _timeToGetSample := FALSE ;
234                     _tp2 ( IN := FALSE , PT := T#0MS ) ;
235                     SamplePosX [ SamplePointer ] := X_Axis . fActPosition ;
236                     SamplePosY [ SamplePointer ] := Y_Axis . fActPosition ;
237                     SamplePosW [ SamplePointer ] := W_Axis . fActPosition ;
238                     SampleTime [ SamplePointer ] := _tp1 . ET ;
239                     _timeToGetSample := TRUE ;
240                     SamplePointer := SamplePointer + 1 ;
241             END_IF
242
243             IF SamplePointer > _SizeOfMem THEN
244                     SamplePointer := 0 ;
245             END_IF
246             _sample := TRUE ;
247     END_IF
248     //end of sampling
249
250     IF _ipo . bDone OR _xErrorExecution OR _xAbortExecution OR bEmergencyStop
THEN
```

```
251             _ipo ( bAbort := TRUE ) ;
252             _ipo . bAbort := FALSE ;
253             _iStatus := 100 ;
254             _tp1 ( IN := FALSE , PT := T#0MS ) ;
255             _tp2 ( IN := FALSE , PT := T#0MS ) ;
256             SamplePointer := 0 ;
257             _bFirstSample := TRUE ;
258     END_IF
259 100 :
260     IF NOT  bEmergencyStop OR NOT  _xAbortExecution THEN
261             _iStatus := 1001 ;
262     END_IF
263
264 1001 : //move to start position
265     _bInStartPosition := TRUE ;
266     _maX (
267             Execute := TRUE ,
268             Position := _lrStartPosition ,
269             Velocity := _lrVelToStartPosition ,
270             Acceleration := _lrAccToStartPosition ,
271             Deceleration := _lrDecToStartPosition ,
272             Axis := X_Axis ) ;
273     _maY (
274             Execute := TRUE ,
275             Position := _lrStartPosition ,
276             Velocity := _lrVelToStartPosition ,
277             Acceleration := _lrAccToStartPosition ,
278             Deceleration := _lrDecToStartPosition ,
279             Axis := Y_Axis ) ;
280     _maW (
281             Execute := TRUE ,
282             Position := _lrStartPosition ,
283             Velocity := 100 ,
284             Acceleration := 120 ,
285             Deceleration := 120 ,
286             Direction := shortest , Axis := W_Axis ) ;
287     _mchX (
288             Execute := ( bEmergencyStop OR  _xAbortExecution ) ,
289             Deceleration := _mchDeceleration ,
290             Axis := X_Axis ) ;
291     _mchY (
292             Execute := ( bEmergencyStop OR  _xAbortExecution ) ,
293             Deceleration := _mchDeceleration ,
294             Axis := Y_Axis ) ;
295     _mchW (
296             Execute := ( bEmergencyStop OR  _xAbortExecution ) ,
297             Deceleration := 720 ,
298             Axis := W_Axis ) ;
299
300     IF  ( _mchX . Done AND  _mchY . Done AND  _mchW . Done ) THEN
301             _bInStartPosition := FALSE ;
302             _mchX . Execute := FALSE ;
303             _mchY . Execute := FALSE ;
304             _mchW . Execute := FALSE ;
305             _maX ( Execute := FALSE , Axis := X_Axis ) ;
306             _maY ( Execute := FALSE , Axis := Y_Axis ) ;
```

```
307             _maW ( Execute := FALSE , Axis := W_Axis ) ;
308             _iStatus := 100 ;
309     END_IF
310
311     IF _maX . Done AND _maY . Done AND _maW . Done THEN
312             _bInStartPosition := FALSE ;
313             _maX ( Execute := FALSE , Axis := X_Axis ) ;
314             _maY ( Execute := FALSE , Axis := Y_Axis ) ;
315             _maW ( Execute := FALSE , Axis := W_Axis ) ;
316             _iStatus := 1002 ;
317     END_IF
318
319     //refresh position tracker during movement to start position
320     _pt (
321             bEnable := TRUE ,
322             bClear := FALSE ,
323             dX := X_Axis . fActPosition ,
324             dY := Y_Axis . fActPosition ,
325             udiNumberOfPointsInArray := SIZEOF ( _bufferTracker ) / SIZEOF
(_bufferTracker [ 0 ] ) ,
326             pBuffer := ADR ( _bufferTracker ) ,
327             _vs3dt => ) ;
328     // end movement to start position
329
330 1002 : //stop the axis
331      _mchX . Execute := TRUE ;
332      _mchY . Execute := TRUE ;
333      _mchW . Execute := TRUE ;
334
335     IF _xErrorExecution OR ( _mchX . Done AND _mchY . Done AND _mchW . Done )
THEN
336             _mchX . Execute := FALSE ;
337             _mchY . Execute := FALSE ;
338             _mchW . Execute := FALSE ;
339             _iStatus := 0 ;
340     END_IF
341     //end stop of axis
342
343 END_CASE
344
345 //controlaxisbypositionvelocity
346 _cbpvX (
347     iStatus := _ipo . iStatus ,
348     bEnable := ( _iStatus = 10 ) ,
349     bAvoidGaps := TRUE ,
350     fSetPosition := _trafo . dx ,
351     fSetVelocity := _ipo . dVel ,
352     fGapVelocity := lrGapVelocity ,
353     fGapAcceleration := lrGapAcceleration ,
354     fGapDeceleration := lrGapAcceleration ,
355     Axis := X_Axis ,
356     bBusy => ,
357     bCommandAborted => ,
358      bError => ,
359      iErrorID => ,
360     bStop_ipo => ) ;
```

```
361 _cbpvY (
362     iStatus := _ipo . iStatus ,
363     bEnable := ( _iStatus = 10 ) ,
364     bAvoidGaps := TRUE ,
365      fSetPosition := _trafo . dy ,
366      fSetVelocity := _ipo . dVel ,
367      fGapVelocity := lrGapVelocity ,
368     fGapAcceleration := lrGapAcceleration ,
369     fGapDeceleration := lrGapAcceleration ,
370     Axis := Y_Axis ,
371     bBusy => ,
372     bCommandAborted => ,
373     bError => ,
374     iErrorID => ,
375     bStop_ipo => ) ;
376 // control for w-achsis in thre case of trafo cutter
377 IF  _bWconstant THEN
378     _cbpvW (
379             iStatus := _ipo . iStatus ,
380             bEnable := ( _iStatus = 10 ) ,
381             bAvoidGaps := FALSE ,
382             fSetPosition := _lrWposition ,
383             fGapVelocity := ,
384             fGapAcceleration := ,
385             fGap_deceleration := ,
386             Axis := W_Axis ,
387             bBusy => ,
388             bCommandAborted => ,
389             bError => ,
390             iErrorID => ,
391             bStop_ipo => ) ;
392     ELSE
393             _cbpvW (
394                     iStatus := _ipo . iStatus ,
395                     bEnable := ( _iStatus = 10 ) ,
396                     bAvoidGaps := FALSE ,
397                     fSetPosition := _trafo . dr ,
398                     fGapVelocity := ,
399                     fGapAcceleration := ,
400                     fGap_deceleration := ,
401                     Axis := W_Axis ,
402                     bBusy => ,
403                     bCommandAborted => ,
404                     bError => ,
405                     iErrorID => ,
406                     bStop_ipo => ) ;
407     END_IF
408
409
410 //end
411
412 //stop
413 _mchX (
414      Axis := X_Axis ) ;
415 _mchY (
416      Axis := Y_Axis ) ;
```

```
417 _mchW (
418       Axis := W_Axis ) ;
419 //end stop
420
421 _xRunningExecution := _ipo . bBusy AND NOT  _xErrorExecution ;//for visualization only
422 _xWaitingExecution := _xWaitExecution AND  ( _ipo . iStatus = IPO_WAIT ) ;//for visualiza-
tion only
423
424 //Calculation of the values for trace
425 _lrPositionX := X_Axis . fActPosition / 1000 ;
426 _lrPositionY := Y_Axis . fActPosition / 1000 ;
427
428 lrVelocityX := X_Axis . fActVelocity / 1000 ;
429 lrVelocityY := Y_Axis . fActVelocity / 1000 ;
430 lrVelocity := SQRT  ( lrVelocityX * lrVelocityX + lrVelocityY * lrVelocityY ) ;
431
432 lrAccelerationX := X_Axis . fSetAcceleration / 1000 ;
433 lrAccelerationY := Y_Axis . fSetAcceleration / 1000 ;
434 lrMaxVelocity_new := lrVelocity ;
435
436 IF  ( lrMaxVelocity < lrMaxVelocity_new ) THEN
437     lrMaxVelocity := lrMaxVelocity_new ;
438 END_IF
439 IF  _ipo . bDone THEN
440     lrMaxVelocity := 0 ;
441 END_IF
442
443 //power
```

6. InitCNC

Das Programm ist verantwortlich für den Zugriff auf die CNC-Dateien, deren Verarbeitung und das wichtigste das Programm generiert dem gewählten G-Code die Datenstruktur(CNC-Data). Die Datenstruktur ist Eingang für den Interpolator, der die einzelnen Achsen ansteuert.

```
1 PROGRAM  InitCNC
2 VAR
3         _xLoadSaveActive : BOOL  ;
4         _xGeneratePathDelayed : BOOL  ;
5
6         _xLoadCommand : BOOL  ;
7         _xSaveCommand : BOOL  ;
8         _xNewCommand : BOOL  ;
9         _xDeleteCommand : BOOL  ;
10        _xDeletePossible : BOOL  ;
11
12        _xUnloadCommand : BOOL  ;
13
14        _iOpenEntry : INT  := - 1 ;
15
16        _iProgramSelectSpin : INT  := 0 ;
17        _sProgram : STRING  ;
18        _sCurrentProgram : STRING  ;
19
```

```
20      _xWaitingExecution_blink , _xErrorExecution_blink : BOOL ;
21      _bInStartPosition : BOOL ;
22      _bInStartPosition_blink : BOOL ;
23      _blink : Blink := ( Enable := TRUE , timelow := T#1S , timehigh := T#1S) ;
24
25      _rncf : SMC_ReadNCFile ;
26      _dec : SMC_NCDecoder ;
27      _cv : SMC_CheckVelocities ;
28
29      _sFilename : STRING ;
30      _iOldProgramSelectSpin : INT := - 1 ;
31
32      _iOldSelectedEntry : INT := - 1 ;
33      _xOpenProgNext : BOOL ;
34 END_VAR
```

Das Programm wird jede 10 ms aufgerufen, um die CNC-Daten zu aktualisieren und an den Eingang vom Interpolator zu geben.

```
1  _blink ( ) ;
2
3  IF _xUpdateDir THEN
4       _xUpdateDir := FALSE ;
5       _iOldProgramSelectSpin := - 1 ; //repaint
6
7       _xLoadSaveActive := TRUE ;
8       UpdateDirectory ( ) ;
9       _xLoadSaveActive := FALSE ;
10 END_IF
11
12 _dirData . iSelectedEntry := LIMIT ( 0 , _dirData . iSelectedEntry , _dirData .iNumberOfEntries - 1 ) ;
13
14 IF _dirData . iSelectedEntry <> _iOldSelectedEntry AND _iOpenEntry = - 1 THEN
15      _xGeneratePath := TRUE ;
16      _sFilename := concat ( _CNCfileDirectory , _dirData . asFiles [ _dirData .iSelectedEntry ] ) ;
17 END_IF
18 _iOldSelectedEntry := _dirData . iSelectedEntry ;
19
20 IF _xLoadCommand THEN
21      _xLoadCommand := FALSE ;
22      _rncf ( bExecute := FALSE ) ;
23
24      _iOpenEntry := _dirData . iSelectedEntry ;
25      _sCurrentProgram := _dirData . asFiles [ _iOpenEntry ] ;
26      _tecData . sFilename := concat ( _CNCfileDirectory , _dirData . asFiles [_iOpenEntry ] ) ;
27      _tecData . xOpen := TRUE ;
28      _xGeneratePath := TRUE ;
29 END_IF
30
31 IF _xUnloadCommand THEN
32      _xUnloadCommand := FALSE ;
33
34      _iOpenEntry := - 1 ;
35      _tecData . xClose := TRUE ;
36 END_IF
37
38 IF _xSaveCommand THEN
39      _xSaveCommand := FALSE ;
40      _iOpenEntry := - 1 ;
41      _tecData . xSave := TRUE ;
```

```
42        _xGeneratePathDelayed := TRUE ;
43  END_IF
44
45  _xDeletePossible := ( _dirData . iSelectedEntry <> _iOpenEntry ) ;
46  IF  _xDeleteCommand THEN
47        _xDeleteCommand := FALSE ;
48
49        _sFilename := concat ( _CNCfileDirectory , _dirData . asFiles [ _dirData .iSelectedEntry ] ) ;
50        SysFileDelete ( _sFilename ) ;
51         _xUpdateDir := TRUE ;
52  END_IF
53
54  IF  _sGenerateNewFile <> '' THEN
55        GenerateNewFile ( ) ;
56        _sGenerateNewFile := '' ;
57  END_IF
58
59  IF  _xGeneratePath OR  ( _xGeneratePathDelayed AND  NOT  _tecData . xSave ) THEN
60        _xGeneratePath := FALSE ;
61        _xGeneratePathDelayed := FALSE ;
62
63         _xLoadSaveActive := TRUE ;
64
65        _pcf . sFilename := _sFilename ;
66        GeneratePathProg ( ) ;
67
68        _xLoadSaveActive := FALSE ;
69  END_IF
70
71  _sProgram := _dirData . asFiles [ _iProgramSelectSpin ] ;
72
73  _iProgramSelectSpin := LIMIT  ( 0 , _iProgramSelectSpin , _dirData .iNumberOfEntries - 1 ) ;
74  IF  _iProgramSelectSpin <> _iOldProgramSelectSpin THEN
75        _pcf . sFilename := concat ( _CNCfileDirectory , _sProgram ) ;
76        GeneratePathExec ( ) ;
77
78        _tecData2 . xClose := ( _iOldProgramSelectSpin > - 1 ) ;
79        _xOpenProgNext := TRUE ;
80  END_IF
81  IF  NOT  _tecData2 . xClose AND  _xOpenProgNext THEN
82        _xOpenProgNext := FALSE ;
83        _tecData2 . sFilename := concat ( _CNCfileDirectory , _sProgram ) ;
84        _tecData2 . xOpen := TRUE ;
85  END_IF
86  _iOldProgramSelectSpin := _iProgramSelectSpin ;
87
88  IF  bStartTrace THEN
89        bStartTracePosCommand := TRUE ;
90        bStartTraceVelCommand := TRUE ;
91        bStartTraceAccCommand := TRUE ;
92  ELSE
93        bStartTracePosCommand := FALSE ;
94        bStartTraceVelCommand := FALSE ;
95        bStartTraceAccCommand := FALSE ;
96  END_IF
97
98  CASE  _iStatus OF
99  1 : //restart FBs
100       _xLoadSaveActive := TRUE ;
101
102       _rncf ( bExecute := FALSE ) ;
```

```
103       _dec ( bExecute := TRUE , bAbort := TRUE , ncprog := _rncf . ncprog ) ;
104       _dec ( bExecute := FALSE , bAbort := FALSE , ncprog := _rncf . ncprog )
;
105       _cv ( bExecute := FALSE ) ;
106       _iStatus := 2 ;
107 2 : //first call
108       _rncf ( bExecute := TRUE , sFilename := concat ( _CNCfileDirectory ,_sProgram ) , pBuffer :=
ADR ( _rncfBuffer ) , dwBufferSize := SIZEOF (_rncfBuffer ) ) ;
109       _dec ( bExecute := _rncf . bExecuteDecoder , ncprog := _rncf . ncprog ,pbyBufferOutQueue :=
ADR ( _decBuffer ) , nSizeOutQueue := SIZEOF (_decBuffer ) ) ;
110 _cv ( bExecute := _rncf . bExecuteDecoder , poqDataIn := _dec .poqDataOut ) ;
111
112 _iStatus := 10 ;
113
114 10 : //normal operation
115       WHILE NOT _cv . poqDataOut ^ . bFull AND NOT _cv . poqDataOut ^ .bEndOfList AND NOT
_xErrorExecution AND NOT _xAbortExecution AND NOT bEmergencyStop DO
116                 _rncf ( ) ;
117                 _dec ( bExecute := _rncf . bExecuteDecoder , ncprog := _rncf . ncprog) ;
118                 _cv ( bExecute := _rncf . bExecuteDecoder ) ;
119
120                 _xErrorExecution := _xErrorExecution OR _rncf.bError OR _dec .bError OR _cv.bError ;
121       END_WHILE
122       IF _dec . bDone THEN
123                 _xLoadSaveActive := FALSE ;
124       END_IF
125 100 :
126       _xLoadSaveActive := FALSE ;
127 END_CASE
128
129 _xWaitingExecution_blink := _xWaitExecution AND _blink . OUT ;
130 _xErrorExecution_blink := _xErrorExecution AND _blink . OUT ;
131 _bInStartPosition_blink := _bInStartPosition AND _blink . OUT ;
```

7. PositionSampling

Das Programm macht eine Abtastung der aktuellen Position in der Orientierung mit einer gewünschten Frequenz möglich. Der maximale Wert der möglichen Abtastfrequenz liegt bei 25 Hz. Bei den höheren Frequenzen gehen manche Abtastwerte verloren, da das Schreiben des abgetasteten Wertes ca. 30 ms dauert.

```
1 PROGRAM PositionSampling
2 VAR
3
4       _sFileDir : STRING := '.\_savedTrajectory\' ;
5       _sFileName : STRING := 'trajectory' ;
6       _iFileFormat : UINT := 1 ;
7       _sFileFormat : ARRAY [ 1 .. 3 ] OF STRING := [ '.txt' , '.xls' , '.h' ] ;
8       _sFileDirName , _sFileDirNameFormat , _sTimeStamp : STRING ;
9       _date : STRING ;
10      dts : DateToString ;
11      getDateAndTime : DTU . GetDateAndTime ;// function block for getting system date and time
12
13      h : RTS_IEC_HANDLE ;
14      result : RTS_IEC_RESULT ;
15
16      open : BOOL := TRUE ;
```

```
17      _iWrite : INT := 0 ; (*case status*)
18      _sHeader : STRING := 'X[mm]$TY[mm]$TW[°]$Ttime[ms]$r$n' ;
19
20      //_soh:SizeOfHeader;
21      sizeHeader : UDINT ;
22      _cap : ConcatActPos ; (*FB for string constuction*)
23      WritePointer : UINT := 0 ;
24      _sPos : STRING ;
25      sizeInt : UDINT ; (*Size of string*)
26 END_VAR
```

Die Variable _sHeader zeigt wie aktuelles Format der Abtastwerte aussieht und in die Datei geschrieben wird. Das Format ist veränderbar mit der Veränderung der Anordnung im ConcatActPos-Funktionsblock.

```
1 // get system time and date
2 getDateAndTime ( xExecute := TRUE ) ;
3
4 IF _iFileFormat > 3 THEN
5       _iFileFormat := 0 ;
6 END_IF
7 IF _iFileFormat < 1 THEN
8       _iFileFormat := 3 ;
9 END_IF
10
11 IF _iStatus = 10 AND AutomaticMode . _bSaveFile THEN
12
13      IF open THEN
14              open := FALSE ;
15              dts (   _bExecute := TRUE , _date := getDateAndTime . dtDateAndTime ) ;
16                      _date := dts . _sFileDate ;
17                      _sFileName := concat ( _date , _sFileName ) ;
18                      _sFileDirName := concat ( _sFileDir , _sFileName ) ;
19                      _sFileDirNameFormat := concat ( _sFileDirName , _sFileFormat [_iFileFormat ]
) ;
20                      h := SysFileOpen ( _sFileDirNameFormat , SysFile . AM_WRITE , ADR (result )
) ;
21
22              SizeOfString ( // Get size of the string
23                              _sString := _sHeader ,
24                              Size => sizeHeader ) ;
25
26              SysFileWrite ( //Write header data into a file
27                      hFile := h ,
28                      pbyBuffer := ADR ( _sHeader ) ,
29                      ulSize := sizeHeader ,
30                      pResult := ADR ( result ) ) ;
31      END_IF
32      IF AutomaticMode . _timeToGetSample THEN
33              AutomaticMode . _timeToGetSample := FALSE ;
34              _sTimeStamp := TIME_TO_STRING ( AutomaticMode . SampleTime [WritePointer ] ) ;
35              _cap ( //Concatenation Actual Position of the measurement data
36                      Enable := TRUE ,
37                      _lrX := AutomaticMode . SamplePosX [ WritePointer ] ,
38                      _lrY := AutomaticMode . SamplePosY [ WritePointer ] ,
39                      _lrW := AutomaticMode . SamplePosW [ WritePointer ] ,
40                      _time := _sTimeStamp ,
41                      _sPos => _sPos ,
42                      Size => sizeInt ) ;
43
```

```
44                  SysFileWrite ( //Write sampled data into a file
45                          hFile := h ,
46                          pbyBuffer := ADR ( _sPos ) ,
47                          ulSize := sizeInt ,
48                          pResult := ADR ( result ) ) ;
49                          WritePointer := WritePointer + 1 ;
50                  IF WritePointer > AutomaticMode . _SizeOfMem THEN
51                          WritePointer := 0 ;
52                  END_IF
53          END_IF
54  END_IF
55  IF _ipo . bDone THEN
56          SysFileClose ( h ) ;
57          h := 0 ;
58          WritePointer := 0 ;
59          open := TRUE ;
60          _sFileName := 'trajectory' ;
61          getDateAndTime ( xExecute := FALSE ) ;
62          dts ( _bExecute := FALSE ) ;
63  END_IF
```

B. Visualisierung

1. Home Visualisierung

Nach dem Starten des Programms gelangt man auf die HOME-Seite. Man hat die Möglichkeit von der Startseite zwischen dem manuellen und automatischen Modus wählen. Desweiteren wird auf der HOME-Seite das aktuelle Datum und die Uhrzeit vom System angezeigt. Die Visualisierung der HOME-Seite ist auf der Abbildung 8-2 angezeigt.

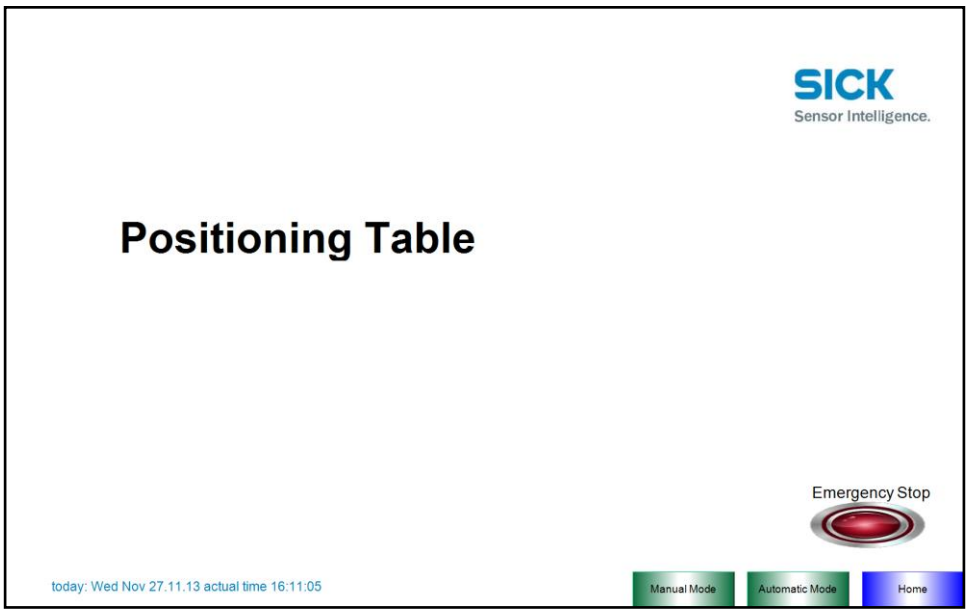


Abbildung 8-2 Home Visualisierung

2. Manual Mode Visualisierung

Abbildung 8-3 zeigt die Visualisierung des manuellen Betriebes. Der Anwender kann beliebigen Punkt des Arbeitsraumes durch die Bewegung

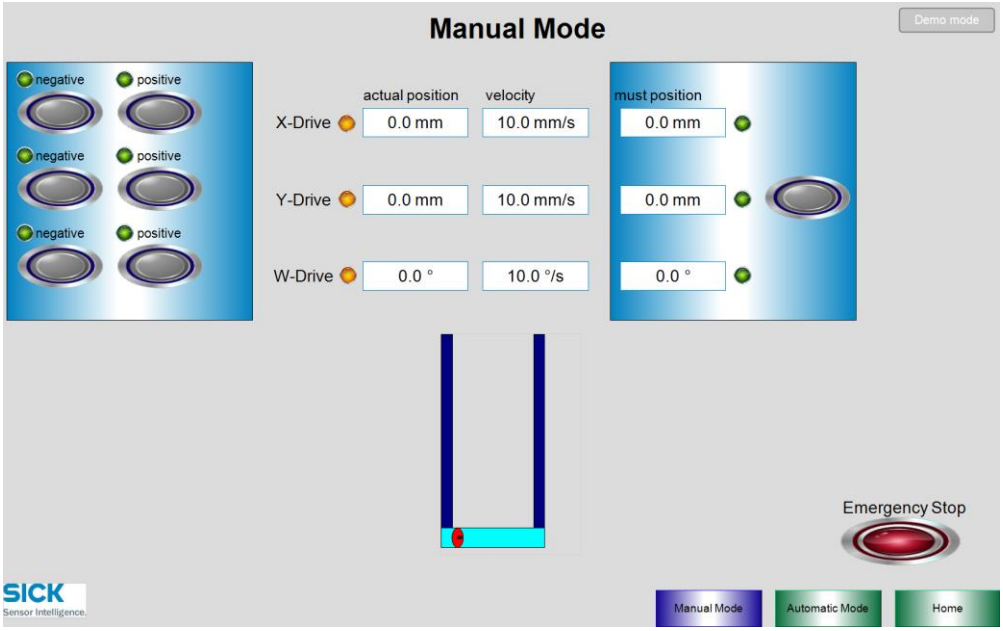


Abbildung 8-3 Visualisierung von Manual Mode

3. Automatic Mode

Automatischer Modus ist wie bereits im Unterkapitel 6.2.2 beschrieben in drei Bereiche aufgeteilt:

- Programmierung
- Ausführung
- Analyse

Die Visualisierung des Programmiermodus ist auf der Abbildung 8-4 dargestellt.

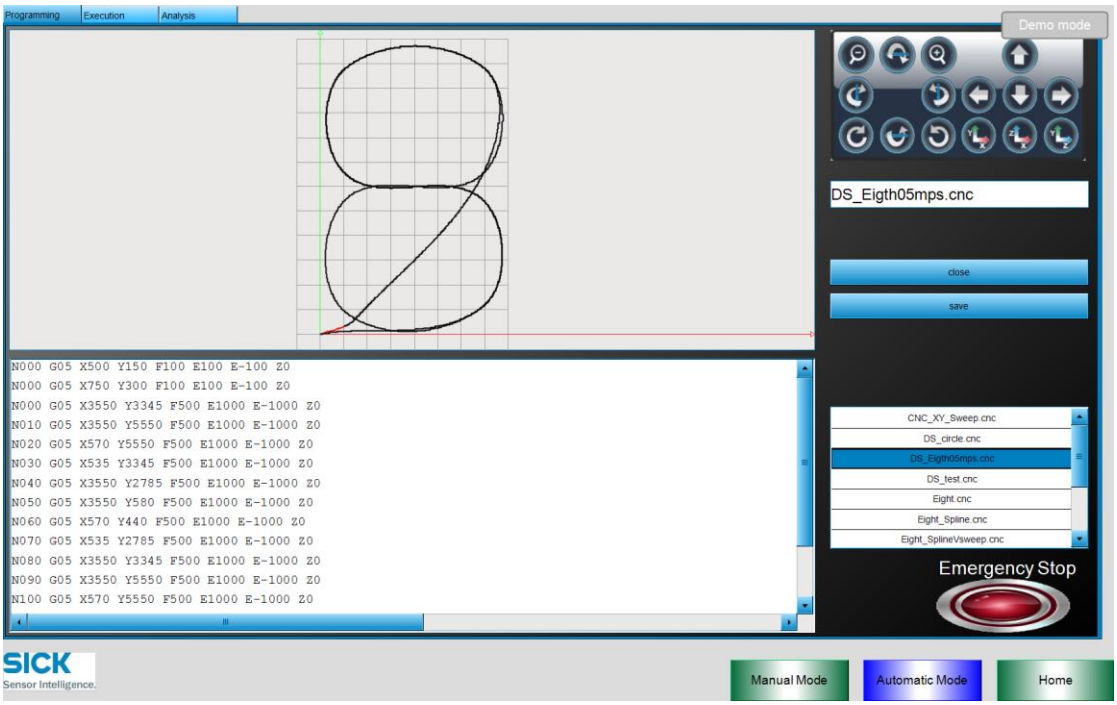


Abbildung 8-4 Visualisierung der Programmierung-Seite

Die Trajektorien können im Programmiermodus aufgerufen, erstellt, verändert und abgespeichert werden. Die grafische Darstellung im kartesischen Koordinatensystem gibt dem Anwender die Möglichkeit die Trajektorie an zu schauen. Ein Bedienpanel in der oberen rechten Ecke der Visualisierung dient zur Drehung, zum Vergrößern bzw. Verkleinern oder zur Verschiebung des Grafen.

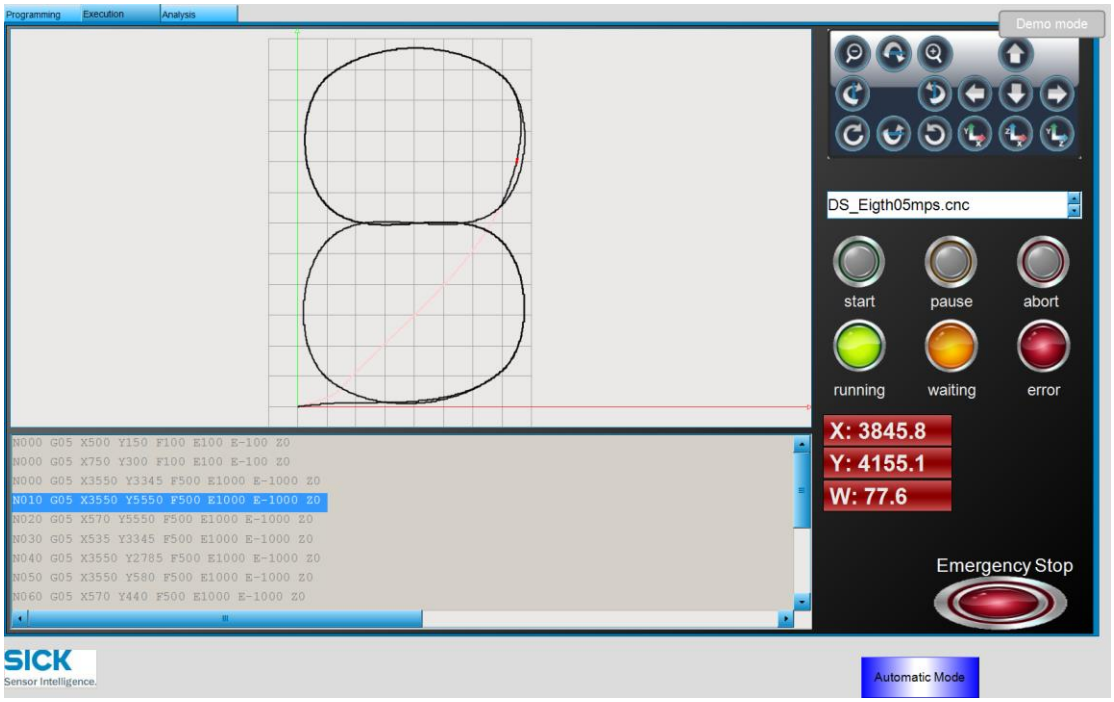


Abbildung 8-5 Visualisierung der Ausführung-Seite

Die bereits erstellten Trajektorien können im Ausführungsmodus ausschließlich angezeigt werden. Die Verarbeitung ist im Modus nicht möglich. Der Anwender kann die Trajektorien verfahren, für eine gewünschte Zeit anhalten oder auch im Notfall unmittelbar stoppen.

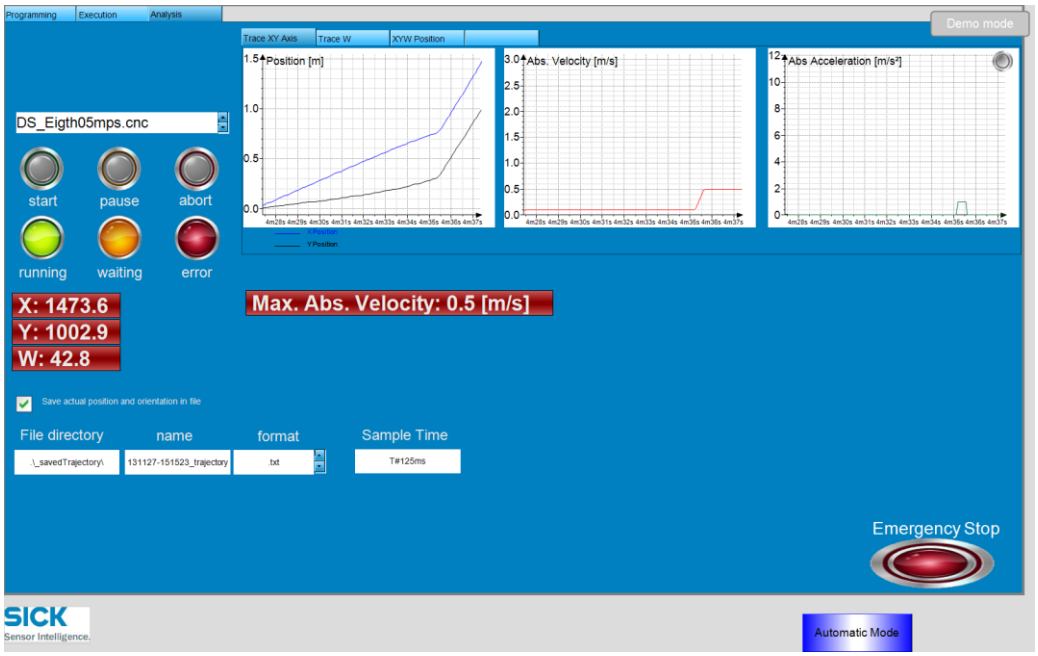


Abbildung 8-6 Visualisierung der Analyse-Seite

Dem Anwender wird beim Umschalten der Visualisierung auf die Analyse-Seite möglich die Trace-Aufzeichnungen an zu schauen. Die Aufzeichnungen sind frei definierbar. Man kann außerdem das Speichern der aktuellen Position und Orientierung des Schlittens in einer Datei abspeichern. Die Abtastfrequenzbereich ist bereits im Kapitel A.7 PositionSampling beschrieben.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur eingegebene Hilfsmittel benutzt habe.

Datum, Ort	Unterschrift