Institut für Parallele und
Verteilte Systeme
Abteilung Parallele Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3222

Eigenspannungsmessung an Hochbelastbaren, Keramischen Beschichtungen

Jinxu Wu

Studiengang:	Informatik
Prüfer:	Prof. Dr. Sven Simon
Betreuer:	Lars Rockstroh
begonnen am:	25. Juli, 2011
beendet am:	20. Februar, 2012
CR-Klassifikation:	D.2.3, F.3.1, H.5.2

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation und Aufgabenstellung

Die thermische und mechanische Belastbarkeit einer Vielzahl von Bauteilen wird heute durch das Auftragen keramischer Schichten erheblich erhöht. Die Haltbarkeit einer Beschichtung ist im Wesentlichen wieder abhängig von den Eigenspannungen, die beim Herstellungsprozess zwischen Schicht und Bauteil sowie in der Beschichtung entstehen. Um den Eigenspannungszustand eines Bauteils zu beurteilen, sollen zuerst in ihm vorhandene Eigenspannungen gemessen werden.

Zur Messung von Eigenspannungen wird häufig das inkrementelle Bohrlochverfahren verwendet. Die durch das Bohren an der Oberfläche des Bauteils ausgelösten Dehnungen werden mit Hilfe von Dehnnungsmessstreifen gemessen. Die Eigenspannungen, die vor dem Bohren in der Umgebung des Loches vorhanden waren, können dann aus den gemessenen Dehnungen abgeleitet werden. Es existiert am Institut für Fertigungstechnologie keramischer Bauteile der Universität Stuttgart ein solcher Aufbau zur Eigenspannungsmessung mit dem Bohrlochverfahren.

Das Ziel dieser Diplomarbeit ist es, die elektrischen Komponenten des existierenden Aufbaus auszutauschen und den mechanischen Teil dieses Aufbaus weiter zu verwenden. Hierzu soll ein Anwendungsprogramm erstellt werden, welches die folgenden Aufgaben erledigt:

- Erfassen von Dehnungen an der Oberfläche des Bauteils mit Hilfe von Dehnnungsmessstreifen und dem USB-Messgerät Vishay Modell P3. Die Messwerte sollen vom Messgerät in den Computer eingelesen, in einer CSV-Datei gespeichert und in der Benutzerschnittstelle angezeigt werden.
- Steuern drei Schrittmotoren für X-, Y- und Z-Achse über die USB-Steuerkarte Coptonix USBMotion3xII.
- Erzeugen einen Satz Befehle, die über die USB-Steuerkarte die Schrittmotoren für X- und Y-Achse steuern, damit eine Kreisbahn mit dem vom Benutzer eingegebenen Radius gefräst wird. Wird die Kreisbahn in inkrementellen Schritten vertieft, so soll das Programm in der Lage sein, zu erkennen, ob nach jeder Vertiefung die Messwerte von den eingesetzten Messkanälen stabil sind.

Das Anwendungsprogramm soll unter gängigen Windows Betriebssystemen wie Windows 7 oder Windows XP lauffähig sein und über eine Benutzerschnittstelle verfügen, über die der Benutzer den Fräser positionieren, den Radius der Kreisbahn eingeben und den Messvorgang starten kann. Die aktuelle Position des Fräsers und die Koordinaten der für das Fräsen der Kreisbahn ausgewählten Punkte sollen auch in der Benutzerschnittstelle angezeigt werden.

1.2. Aufbau der Arbeit

Diese Arbeit gliedert sich in sieben Kapitel:

In Kapitel 2 wird zunächst ein Überblick über die Entstehung von Eigenspannungen in einem Bauteil gegeben. Bevor das Eigenspannugsmessverfahren Bohrlochverfahren vorgestellt wird, werden die bei diesem Verfahren verwendeten Dehnungsmessstreifen (DMS) und DMS-Brückenschaltungen vorgestellt.

In Kapitel 3 werden das bei der Durchführung des Bohrlochverfahrens einzusetzende USB-Messgerät Vishay Modell P3 und die ebenfalls dabei einzusetzende USB-Steuerkarte USBMotion3xII vorgestellt.

In Kapitel 4 werden die dieser Arbeit zugrunde liegenden Softwaretechnologien vorgestellt. Es betrifft vor allem Dynamic Link Library (DLL), ActiveX-Steuerelement, .Net Framework und C++/CLI-Schnittstelle.

In Kapitel 5 wird zunächst der Arbeitsablauf des zu erstellenden Anwendungsprogrammes bei einem Messvorgang beschrieben. Anschließend werden für zwei wichtige Aufgaben, die bei einem Messvorgang erledigt werden müssen, geeignete Lösungen erarbeitet.

In Kapitel 6 wird zunächst die eingerichtete Entwicklungsumgebung beschrieben. Anschließend werden die implementierten Klassen und deren Methoden näher beschrieben. Schließlich werden die Benutzerschnittstelle und die Testergebnisse des Anwendungsprogrammes vorgestellt.

Eine Zusammenfassung der wichtigen Grundlagen und der geleisteten Arbeit sowie ein Ausblick auf mögliche Verbesserungen werden in Kapitel 7 gegeben.

In Anhang A werden die vom ActiveX-Steuerelement VMMP3Control zur Verfügung gestellten Eigenschaften und Methoden vorgestellt. In Anhang B werden die Funktionen der DLL USBM3x32 vorgestellt. Der Quelltext des Anwendungsprogrammes wird in Anhang C aufgelistet.

2. Grundlagen der Eigenspannungsmessung

2.1. Eigenspannungen

Unter *Eigenspannungen* versteht man mechanische Spannungen in einem Bauteil bzw. einem abeschlossenen System, auf das keine äußeren Kräfte und Momente wirken und das sich in einem räumlich und zeitlich konstanten Temperaturfeld befindet. Die mit den Eigenspannungen verbundenen inneren Kräfte und Momente sind im mechanischen Gleichgewicht, das Bauteil bzw. das System weist einen sog. Eigenspannungszustand auf.

Eigenspannungen entstehen immer durch inhomogen verteilte elastische und elastisch-plastische Verformungen des Bauteils. Bei der Herstellung sind z.B. Gießen, Schweißen, Bearbeiten, Beschichten oder Wärmebehandeln dafür verantwortlich. Im Betrieb sind z.B. Beanspruchungen und Temperaturfelder. Für die metall-keramischen Schichtverbundbauteile sind v.a. die unterschiedlichen elastschen Eigenschaften der einzelnen Werkstoffschichten die Entstehungsursache von Eigenspannungen.

Eigenspannungen werden nach ihrer Reichweite im Werkstoff in Eigenspannungen 1., 2. und 3. Art eingeteilt [3]:

- Die Eigenspannungen 1. Art sind wirksam über größere Werkstoffbereiche bzw. über mehrere Körner. Sie verursachen bei der Störung des mechanischen Gleichgwichts immer makroskopische Maßänderungen.
- Die Eigenspannungen 2. Art sind wirksam über kleine Werkstoffbereiche (Kornbereiche, d.h. von Korn zu Korn). Sie können bei der Störung des mechanischen Gleichgwichts makroskopische Maßänderungen verursachen.
- Die Eigenspannungen 3. Art sind wirksam über kleinste Werkstoffbereiche (mehrere Atomabstände). Sie verursachen bei der Störung des mechanischen Gleichgwichts keine makroskopischen Maßänderungen.

In einem Bauteil überlagern sich alle drei Eigenspannungsarten [14]. Von technischer Bedeutung sind in erster Linie die Eigenspannungen 1. Art, da sie immer makroskopische Maßänderungen verursachen, die die Lebensdauer, die Formstabilität und das Bruchverhalten eines Bauteils maßgeblich bestimmen.

Die Überlagerung der Eigenspannungen mit den Lastspannungen beeinflusst das Festigkeitsverhalten der Bauteile sowohl im positiven als auch im negativen Sinne, je nachdem, ob die beiden Spannungsarten in entgegengesetzter Richtung oder in gleicher Richtung wirken.

In den nächsten Abschnitten wird erklärt, wie die Eigenspannungen beim Bohrlochverfahren mit Hilfe von Dehnungsmessstreifen und Wheatstonesche Brückenschaltungen gemessen werden können.

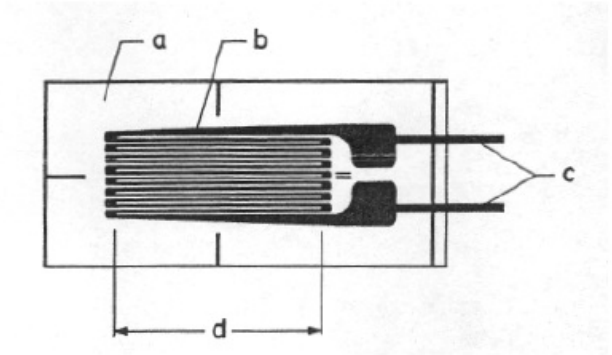
2.2. Dehnungsmessstreifen (DMS)

Dehnungsmessstreifen (englisch: *strain gauge*) sind Messinstrumente zur Dehnungsmessung. Sie verändern bei Dehnungen oder Stauchungen (negativen Dehnungen) ihren elektrischen Widerstand und werden zur experimentellen Bestimmung von Eigenspannungen des Bauteils eingesetzt.

2.2.1. Aufbau und Formen

DMS sind meist aus Metall oder Halbleiter hergestellt, haben folgende Bauformen[4]:

- Draht-DMS: bestehen aus einem elektrisch isolierenden Trägermaterial und einem sich darauf befindenden Messgitter bzw.einem flach gewickelten, dünnen, metallischen Widerstandsdraht.
- Folien-DMS: bestehen aus einem elektrisch isolierenden Trägermaterial und einem Messgitter bzw. einer metallischen Widerstandsfolie (siehe Abb. 2.1). Mittels der Photo-Ätztechnik kann die Form von Widerstandsfolie vielfältig gestaltet werden.
- Halbleiter-DMS: bestehen aus einem Trägermaterial und einem Halbleiter-Messgitter (Silizium). Ihre Empfindlichkeit sind vielfach höher als Metall-DMS. Ihre Temperaturstabilität sind aber schlechter.



a Trägerfolie
b Messgitter
c Anschlüsse
d wirksame Messgitterlänge

Abbildung 2.1.: Charakteristische Bauform von Folie-DMS

Wie in der Abb. 2.1 gezeigt, ist die Leiterbahn mäanderförmig ausgeführt. DMS können damit einen ausreichend hohen Widerstandswert erreichen. Die vielen parallelen Leiterstücke verstärken auch den Effekt der Widerstandsänderung durch Dehnung. An den Enden des Messgitters sind 2 Anschlussdrähte befestigt.

Der Nennwiderstand eines DMS ist der Widerstand zwischen seinem beiden Anschlüssen im unbelasteten Fall. DMS werden mit verschiedenen Nennwiderstandswerten hergestellt. Typische Werte sind 120, 350, 700 und 1000 Ohm.

Werden auf einem Träger mehrere Messgitter in verschiedener Richtung nebeneinander oder übereinander angeordnet, so entsteht eine DMS-Rosette. Für mehrachsige Messungen kommen die DMS-Rosetten zum Einsatz.

2.2.2. Funktionsweise

Die Dehnungsmessung mit DMS beruht auf dem von Wheatstone und Thomson gefundenen Dehnungs-Widerstands-Effekt elektrischer Leiter. Der Widerstand eines unbelasteten metallischen Leiters kann man bestimmen durch:

$$R = \rho \frac{l}{A} = \rho \frac{4 \cdot l}{\pi \cdot d^2} \tag{2.1}$$

wobei ρ den spezifischen Widerstand des Leiers, l die Leiterlänge und d den Durchmesser des Leiters bezeichnet.

Die Widerstandsänderung des Leiters durch Zug, Druck, Biegung oder Torsion ist dargestellt durch:

$$\Delta R = \frac{\partial R}{\partial \rho} \cdot \Delta \rho + \frac{\partial R}{\partial l} \cdot \Delta l + \frac{\partial R}{\partial d} \cdot \Delta d \tag{2.2}$$

Dividiert man diese Gleichung durch R, so erhält man die relative Widerstandsänderung:

$$\frac{\Delta R}{R} = \frac{\Delta \rho}{\rho} + \frac{\Delta l}{l} - \frac{2 \cdot \Delta d}{d} \tag{2.3}$$

Die relative Widerstandsänderung ist also von der Längs- und der Querdehnung abhängig. Seien

$$\epsilon = \frac{\Delta l}{l} \qquad \text{und} \qquad \epsilon_q = \frac{\Delta d}{d} = -\mu \cdot \epsilon \tag{2.4}$$

Somit erhält man:

$$\frac{\Delta R}{R} = \frac{\Delta \rho}{\rho} + \epsilon + 2 \cdot \mu \cdot \epsilon = k \cdot \epsilon \tag{2.5}$$

wobei k, der sogenannte k-Faktor, die Dehnungsempfindlichkeit darstellt:

$$k = \frac{\Delta \rho}{\rho \cdot \epsilon} + 1 + 2 \cdot \mu \tag{2.6}$$

mit
ϵ : relative Längenänderung
ϵ_q : relative Querschnittsänderung
μ : materialspezifischer Zusammenhang zwischen ϵ und ϵ_q

Der k-Faktor gibt also das Verhältnis der relativen Widerstandsänderung $\frac{\Delta R}{R}$ zur Dehnung ϵ an. Der Wert von k-Faktor wird experimentell ermittelt.

Der Widerstand von Metallischen DMS wird also durch ihre Längen- und Querschnittsänderung verändert. Wird ein DMS gedehnt, so nimmt sein Widerstand zu. Wird er gestaucht, so nimmt sein Widerstand ab.

DMS werden mit einem spezielen Klebstoff auf dem Messobjekt befestigt. Die Dehnungen an der Oberfläche des Messobjekts werden über den Klebstoff und das Trägermaterial auf die Messgitter übertragen und führen dazu, dass DMS ihren Widerstand verändern.

2.3. DMS-Brückenschaltungen

DMS werden üblicherweise in Wheatstonesche Brückenschaltungen eingebaut, damit ihre kleinen Widerstandsänderungen in die leicht zu messenden Spannungsänderungen umgewandelt werden. Man kann einen, zwei oder alle vier Zweige einer Brückenschaltung mit DMS besetzen und bezeichnet diese Brückenschaltung dann auch als Viertel-, Halb- oder Vollbrücke.

Die in Abb.2.2 dargestellte Brückenschaltung besteht aus vier Widerständen R1 bis

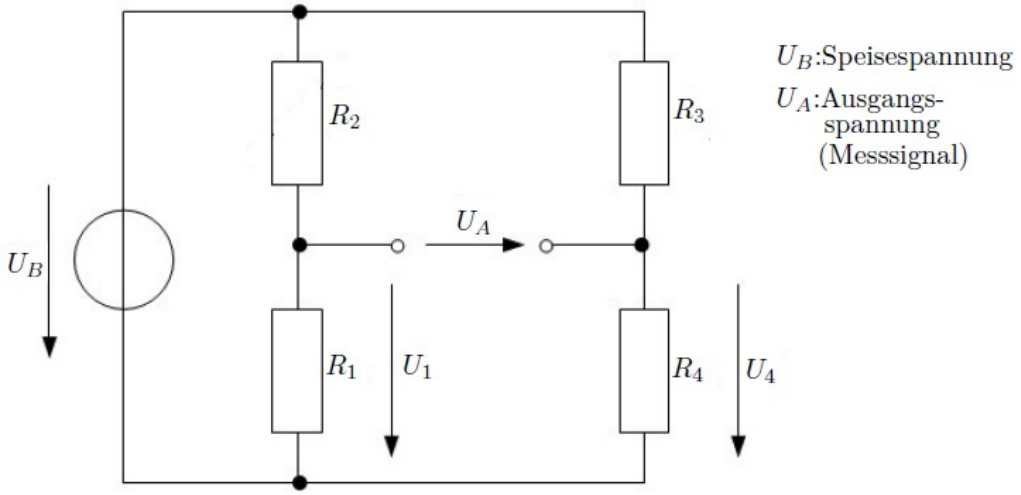


Abbildung 2.2.: Wheatstonesche Brückenschaltung

R4 und ist von der Gleichspannungsquelle U_B gespeist. Für die beiden Spannungsteiler R_1, R_2 und R_3, R_4 gilt:

$$U_1 = U_B \cdot \frac{R_1}{R_1 + R_2} \qquad \text{und} \qquad U_4 = U_B \cdot \frac{R_4}{R_3 + R_4} \tag{2.7}$$

Die Differenz der Spannungen U_1 und U_4 ist die Messspannung U_A bei der hochohmigen, stromlosen Messung:

$$U_A = U_1 - U_4 = U_B \cdot \left(\frac{R_1}{R_1 + R_2} - \frac{R_4}{R_3 + R_4} \right) \tag{2.8}$$

Sind R_1, R_2, R_3 und R_4 DMS mit gleichem Nennwiderstand R, dann gilt für R_i, $i = 1..4$, bei Dehnungen:

$$R_i = R + \Delta R_i = R \cdot \left(1 + \frac{\Delta R_i}{R} \right) = R \cdot (1 + r_i) \tag{2.9}$$

wobei $r_i = \frac{\Delta R_i}{R} = k\epsilon$, so erhält man:

$$U_A = U_B \cdot \left(\frac{1 + r_1}{2 + r_1 + r_2} - \frac{1 + r_4}{2 + r_3 + r_4} \right) \tag{2.10}$$

Da die relative Widerstandsänderungen von metallischen DMS sehr klein sind, gilt die Näherung $\frac{1}{1+r_i} \approx 1 - r_i$. Damit kann die Formel (3.10) vereinfacht werden zu:

$$U_A = \frac{U_B}{4} \cdot (r_1 - r_2 + r_3 - r_4) = \frac{U_B}{4} \cdot k \cdot (\epsilon_1 - \epsilon_2 + \epsilon_3 - \epsilon_4) \tag{2.11}$$

Die Formel (2.11) weist folgende Vorteile der DMS-Brückenschaltung bei der Dehnungsmessung auf:

- Im unbelasteten Fall ist die Ausgangsspannung U_A (Messsignal) gleich Null.
- Die temperaturbedingte Widerstandsänderungen in der Brückenschaltung lassen sich kompensieren.
- Die Ausgangsspannung U_A ist proportional zu den gemessenen Dehnungen an der Oberfläche des Messobjekts.

Bei der Viertelbrücke sind $\epsilon_2 = \epsilon_3 = \epsilon_4 = 0$, bei der Halbbrücke sind $\epsilon_3 = \epsilon_4 = 0$. Bei der Vollbrücke gilt $U_A = U_B \cdot k \cdot \epsilon$, falls ϵ_1 und ϵ_3 durch Dehnung entstehen, ϵ_2 und ϵ_4 durch Stauchung und sie alle betragsgleich sind.

Die Festwiderstände R_3, R_4 und bei der Viertelbrücke auch R_2 sowie die Spannungsquelle U_B sind schon im in dieser Arbeit verwendeten Dehnungsmessgerät Vishay P3 enthalten, so dass in der Praxis die am Messobjekt befindlichen DMS direkt an dieses Messgerät angeschlossen werden können.

Nachdem die Ausgangsspannung U_A auf eine ausreichende Höhe verstärkt und digitalisiert wurde, kann sie in einen Rechner eingelesen und weiterverarbeitet werden.

Wie eine DMS-Viertelbrücke in der Praxis zum Einsaz kommt, ist in [12] vorgestellt.

2.4. Eigenspannungsmessverfahren

Zur Eigenspannungsmessung gibt es verschiedene Verfahren, die sich nach dem Zerstörungsgrad des Bauteils durch das jeweilige Messverfahren in zerstörende, teilzerstörende und zerstörungsfreie Verfahren einteilen lassen[3].

In der Industrie und der Forschung wird zur Messung von Eigenspannungen 1. Art häufig das Bohrlochverfahren verwendet. Es ist ein teilzerstörendes Messverfahren, genormt und relativ einfach durchzuführen. Im folgenden werden das Messprinzip und die Durchführung dieses Verfahren beschrieben.

Wird an einer Messstelle eines Bauteils durch das Bohren etwas eigenspannungsbehaftetes Material entfernt, wird dadurch das innere mechanische Gleichgewicht gestört, was an der Oberfläche des Bauteils um das Bohrloch Dehnungen auslösen wird. Mit Hilfe der gemessenen Dehnungen kann der zuvor an der Messstelle vorhandene Eigenspannungszustand berechnet werden.

Erfolgt das Bohren inkrementell bzw. schrittweise, so können die Dehnungsänderungen als Funktion der Bohrtiefe gebildet werden. Mit Hilfe dieser Funktion kann die Eigenspannungstiefenverteilung berechnet werden [7].

Bei der Durchführung dieses Verfahrens geht man folgendermaßen vor: Zuerst wird eine DMS-Rosette aus drei Messgitter an einer interessierenden Stelle eines zu untersuchenden Bauteils installiert, die Messgitter werden an eine Dehnungsmessbrücke angeschlossen. Dann wird eine kleine Bohrung an dieser Stelle, zentrisch zur DMS-Rosette, eingebracht. Dazu wird ein rotierende Fräser eingesetzt, dessen Vorschub normalerweise durch den Schrittmotor erfolgt. Die beim schrittweisen Einbringen der Bohrung im Bauteil ausgelösten Dehnungsänderungen werden mit der DMS-Rosette in drei Richtungen erfasst.

Eine ausführliche Beschreibung dieses Verfahrens ist in [3] und [13] gegeben.

In der Praxis, um Dehnungsänderungen auszulösen, kann übrigens in der Mitte der DMS-Rosette eine Kreisbahn gefräst und diese schrittweise vertieft werden. Diesen Vorgang zu steuern, ist eine Aufgabe des zu erstellenden Anwendungsprogrammes.

3. Hardware

In diesem Kapitel werden das Dehnungsmessgerät Vishay Modell P3 und die Schrittmotorsteuerkarte USBMotion3xII vorgestellt. Die beiden sollen bei der Durchführung des Bohrlochverfahrens eingesetzt werden.

3.1. Vishay Modell P3

Das Modell P3 Strain Indicator and Recorder der Firma Vishay Micro-Measurements [1] ist ein tragbares, batteriegespeistes Prazisionsinstrument zur Messwertaufnahme auf der DMS-Basis. Es besitzt u.a. folgende Eigenschaften [11]:

- Vier Eingangskanäle, an die Viertel-, Halb- und Vollbrückenschaltungen angeschlossen werden können.
- Eingebaute Ergänzungswiderstände für 60Ω- bis 2000Ω-Halbbrückenschaltungen sowie 120Ω-, 350Ω- und 1000Ω-Viertelbrückenschaltungen.
- Ein Speicherkarteneinschub zur Aufnahme von Multimedia-Karte (MMC). Die erfassten Messwerte können auf der MMC mit einer Aufzeichnungsrate von 1/Sekunde bis 1/Stunde gespeichert werden.
- Eine USB Schnittstelle, über die die Messwerte von den vier Eingangskanälen sowie auf der MMC in den Computer eingelesen werden können.
- Nullabgleich und Kalibrierung automatisch oder manuell.
- Stromversorgung über Batterie, USB Schnittstelle oder AC-Adapter.
- Messbereich: $\pm 31.000 \mu m/m$ bei K-Faktor $= 2,000$ mit $\pm 1 \mu m/m$ Auflösung (Statt $\mu m/m$ wird auf dem Bildschirm μe gezeigt)

Die Abbildung 3.1 zeigt die Frontplatte des Gerätes. Es verfügt noch über eine LCD-Anzeige, eine Softkeytastatur und einen Analogausgang. Die vier Eingangskanäle sind mit Klemmanschlüssen bestückt, die einen einfachen und schnellen Drahtanschluss ermöglichen. Die Abbildung 3.2 zeigt die Anschlussschemata für Viertel-, Halb- und Vollbrückenschaltung. Für Viertelbrückenschaltung ist beim Anschluss eine von drei Brückenergänzungsklemmen (D_{120}, D_{350} und D_{1K}), welche den Nominalwiderständen des DMS entsprechen, zu wählen.

Zum Lieferumfang des Gerätes gehört auch ein ActiveX-Steuerelement [10], welches in unserem Anwendungsprogramm eingesetzt werden soll, um v.a. den Messwert des gewählten Eingangskanals in Echtzeit in den Computer einzulesen.

[1] http://www.vishaypg.com/micro-measurements/instruments/p3-list/

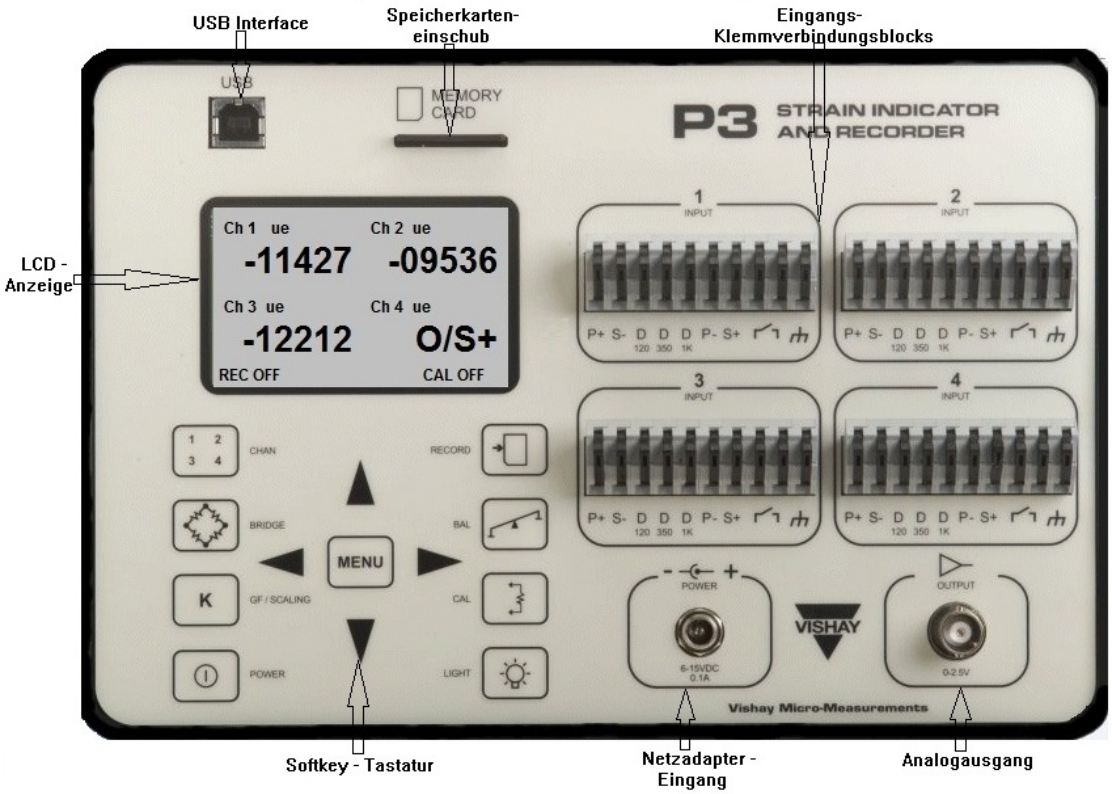


Abbildung 3.1.: Frontplatte von Vishay Modell P3

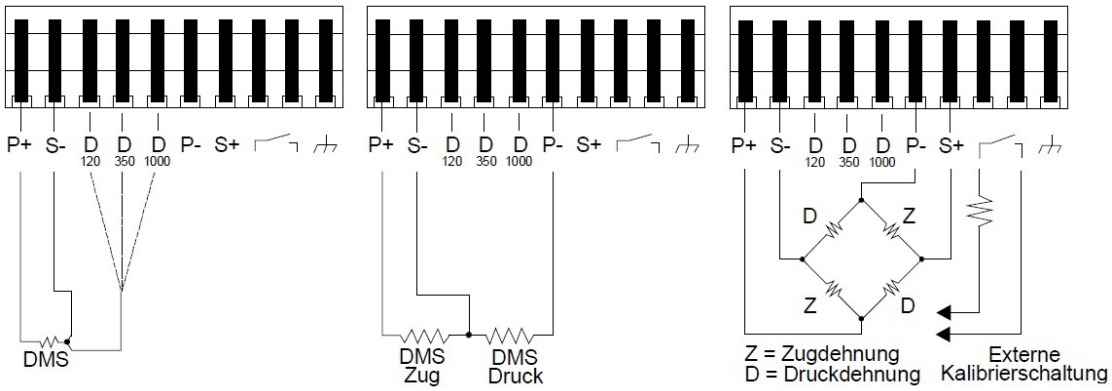


Abbildung 3.2.: Anschlussschemata für Viertel-, Halb- und Vollbrückenschaltung

Abbildung 3.3.: Schrittmotorsteuerkarte USBMotion3XII

3.2. Steuerkarte USBMotion3xII

Der beim Bohrlochverfahren eingesetzte Fräser wird durch drei Schrittmotoren für die X-, Y- und Z-Achse angetrieben. Die Steuerung der Schrittmotoren soll dabei über die USB-Steuerkarte USB Motion 3XII von der Firma Coptonix [2] erfolgen. Diese Karte verfügt über drei Schnittstellen zum Anschluss des Schrittmotors und eine I2C-Schnittstelle, über die weitere elektronische Komponenten angeschlossen werden können (Abb. 3.3). Der Hersteller hat für diese Karte eine DLL angeboten, die die Funktionen zur Verwaltung der angeschlossenen Komponenten und zur Steuerung der Schrittmotoren bereitstellt. Damit muss kein Low-Level Treiber mehr für die Steuerung der Schrittmotoren entwickelt werden [2].

Die am häufigsten verwendeten Funktionen zur Schrittmotorsteuerung sind im folgenden kurz beschrieben. Eine Auflistung der verfügbaren Funktionen und die ausführliche Beschreibung befinden sich im Anhang A.2.

`SetXtarget`	legt die Schrittzahl fest, die der Schrittmotor 0, 1 oder 2 machen soll.
`SetXYZtarget`	legt die Schrittzahlen fest, die die Schrittmotoren 0, 1 und/oder 2 machen sollen.
`SetVmax`	legt die maximale Geschwindigkeit vom Schrittmotor 0, 1 oder 2 fest.
`GetXtarget`	gibt die Zielposition eines Schrittmotors zurück.
`SetVmax`	gibt die maximale Geschwindigkeit eines Schrittmotors zurück.

Beim Festlegen der Schrittzahlen spricht man auch von den Zielpositionen der Schrittmotoren. Schreibt die Funktion `SetXtarget` die Schrittzahl ins Zielposition-Register vom

[2] http://www.coptonix.com/index.html. Statt USB Motion 3XII ist USB Motion 3XIII im aktuellen Angebot.

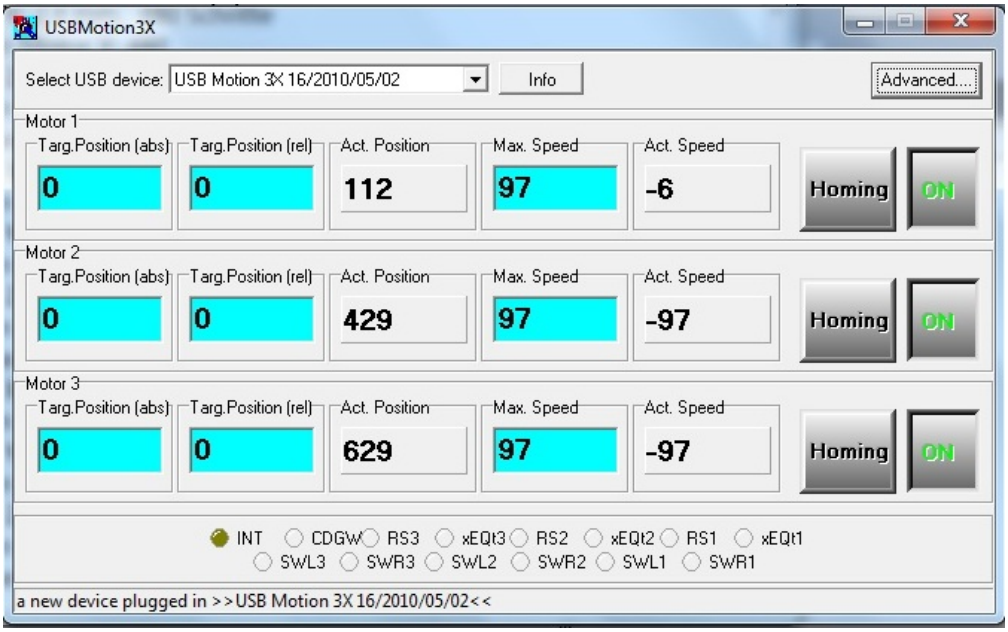


Abbildung 3.4.: USBMotion3X II Benutzerschnittstelle

Schrittmotor für die X-Achse, so bewegt sich der Schrittmotor in der Ziel-Richtung solange, bis die Differenz zwischen der Zielposition und der aktuellen Position gleich Null ist. Der Fräser bewegt sich dabei in positiver x-Richtung, falls die Schrittzahl positiv ist, sonst in negativer x-Richtung.

Zur Schrittmotorsteuerung hat der Hersteller auch eine Benutzerschnittstelle angeboten (Abb.5.4), über die der Benutzer die Zielposition, die maximale Geschwindigkeit sowie andere Parameter wie Betriebsart, Schrittauflösung, die maximale Beschleunigung usw. festlegen kann. Für den Fall, dass die Zielpositionen mit Hilfe der DLL festgelegt werden, können die aktuellen Positionen und Geschwindigkeiten von drei Schrittmotoren in dieser Benutzerschnittstelle angezeigt werden. Das ist besonders hilfreich beim Test von unserem Anwendungsprogramm.

4. Basistechnologien

In diesem Kapitel werden zuerst die dieser Arbeit zugrunde liegenden Softwaretechnologien vorgestellt. Das sind vor allem Dynamic Link Library (DLL), ActiveX-Steuerelement und das .Net Framework. Anschließend werden einige Besonderheiten der C++/CLI-Schnittstelle, die in dieser Arbeit verwendet werden, beschrieben.

4.1. Softwarekomponenten in Form von DLLs

Softwarekomponenten sind Softwarebauteile, welche in binärer Form vorliegen und ihre Dienste ausschließlich über vordefinierte Schnittstellen nach außen zur Verfügung stellen. Sie können auch die Dienste von anderen Komponenten in Anspruch nehmen. Es ergibt sich damit die Möglichkeit, neue Komponenten oder Anwendungen aus vorhandenen Komponenten zusammenzusetzen.

Die ActiveX-Technologie von Microsoft ist neben JavaBeans von JavaSoft auch eine weitverbreitete Technologie zur Erstellung von Softwarekomponenten. Die aufgrund dieser Technologie erstellten Komponenten, ActiveX-Steuerelemente, werden im nächsten Abschnitt näher betrachtet.

Softwarekomponenten können in Form von DLLs vorliegen. DLL ist die Abkürzung für Dynamic Link Library, also eine Programmbibliothek, die erst zur Laufzeit in eine Anwendung eingebunden wird. Unter Windows gibt es zwei Arten von DLLs [1]:

- Einsprungs-DLLs, die Prozeduren und Funktionen enthalten. Die in dieser Arbeit zur Schrittmotorsteuerung verwendete DLL (USBM3x32.dll) gehört zu dieser Art.
- ActiveX-DLLs, die Klassen enthalten, deren Dateiendung auch `OCX` sein kann. Die in dieser Arbeit zur Messdatenbearbeitung verwendete DLL (VMMP3Control.dll) gehört zu dieser Art.

Es gibt zwei unterschiedliche Arten, DLLs in eine Anwendung einzubinden[5]:

Implicit Run-Time Linking: Wenn es zu einer DLL eine Importbibliothek gibt und eine Verknüpfung mit dieser Bibliothek hergestellt ist, so wird diese DLL zur Laufzeit automatisch geladen. Die benötigten DLL-Funktionen werden jeweils mit der Funktion `__declspec(dllimport)` importiert. Ein explizites Abfragen von Funktionseinsprungadresse ist nicht erforderlich.

Explicit Run-Time Linking: Erst zur Laufzeit werden die Windows-API-Funktionen `LoadLibrary` und `GetProcAddress` aufgerufen, um das DLL-Handle und die einzelnen Funktionspointer zu bekommen, über die dann die DLL-Funktionen aufgerufen werden können. Die geladene DLL kann später mit dem Aufruf der Windows-API-Funktion `FreeLibrary` wieder aus dem Arbeitsspeicher entfernt werden. Die Importbibliothek ist

[1] http://de.wikipedia.org/wiki/Programmbibliothek

in diesem Fall nicht nötig. Die DLL-Datei muss aber in einem der Anwendung zugänglichen Ordner liegen.

Dlls auf die erste Art einzubinden, ist relativ einfach zu handhaben. Der Nachteil besteht darin, dass beim Programmstart alle auf diese Art eingebundenen DLLs in den Speicher automatisch geladen werden, auch wenn sie gegebenenfalls nicht benötigt werden. Der Programmstart wird dadurch verlangsamt. Das Programm braucht auch mehr Speicherplatz. Ist eine DLL oder Funktion nicht vorhanden, kann das Programm bereits den Start verweigern.

Dlls auf die zweite Art einzubinden, ist flexibel und braucht weniger Speicherplatz, weil DLLs nur bei Bedarf in den Arbeitsspeicher geladen werden. Der Programmierer muss aber sich selbst um das Laden und das Entfernen der DLLs sowie das Abfragen der Einsprungadressen kümmern. Da dafür Windows-API-Funktionen verwendet werden, bekommt das Programm auch hilfreiche Rückmeldungen, wie ob eine DLL erfolgreich geladen wurde, ob die DLL-Funktionen galaden werden konnte, so dass während der Laufzeit das Programm auf nicht vorhandene DLLs reagieren kann.

Die in dieser Arbeit verwendete DLL USBM3x32.dll kann nur auf die zweite Art in eine Anwendung eingebungden werden. Im Anhang B.2 befindet sich ein Beispielprogramm, das den Umgang mit USBM3x32.dll demonstiert.

4.2. ActiveX-Steuerelemente (ActiveX Controls)

Ein Microsoft ActiveX-Steuerelement ist eine wiederverwendbare Softwarekomponente, die man in eigenständige Anwendungen und Web-Seiten einbauen kann. Der Programmierer kann ActiveX-Steuerelemente gleichermaßen in verschiedenen Programmierumgebungen wie Visual Basic, Delphi oder C++ einsetzen, ganz unabhängig davon, in welcher Programmiersprache sie geschrieben worden sind.

Die ActiveX-Technologie basiert auf der ebenfalls von Microsoft entwickelten COM-Technologie (Component Object Model), welche zur Herstellung der standardisierten Softwarekomponenten eingeführt wurde[1]. Ein ActiveX-Steuerelement muss mindestens die von COM definierte Basisschnittstelle *IUnknown* implementieren. Die weiteren von COM sowie vom Entwickler selbst definierten Schnittstellen müssen von *IUnknown* ableitbar sein.

ActiveX-Steuerelemente können nicht eigenständig laufen. Sie brauchen einen so genannten ActiveX-Container. Dies kann eine Visual C++- oder Visual Basic-Anwendung sein, wenn in ihr ActiveX-Steuerelemente eingesetzt werden. ActiveX-Steuerelemente kommunizieren dann mit ihrem Container ausschließlich über die Schnittstellen. Ein ActiveX-Steuerelement kann über folgende Elemente verfügen:

- Methoden, in denen sein Verhalten abgebildet sind. Der ActiveX-Container kann diese Methoden aufrufen.
- Ereignisse, die den ActiveX-Container benachrichtigen, dass bestimmte Ereignisse eingetreten sind.
- Eigenschaften, die den Zustand des Objekts beschreiben und sich durch den ActiveX-Container modifizieren lassen.

Liegt ein ActiveX-Steuerelement in Form von DLL oder OCX vor, so kann es von der Entwicklungsumgebung wie Visual C++ referenziert werden. Die von COM definierten Schnittstellen ITypeLib und ITypeInfo werden benutzt, um dem Entwickler zur Entwurfszeit die Typbibliothek, die die Informationen über Methoden, Eigenschaften und Ereignisse des Steuerelementes bietet, zur Verfügung zu Stellen[5]. Dieses ActiveX-Steuerelement kann dann in der Entwicklungsumgebung wie normale Steuerelemente behandelt werden.

ActiveX-Steuerelemente gibt es übrigens nur für die Betriebssystemfamilie Windows.

4.2.1. ActiveX-Steuerelement registrieren

Nach der Definition von Microsoft [2] ist ein ActiveX-Steuerelement im wesentlichen ein COM-Objekt, das die *IUnknown* Schnittstelle implementiert hat und sich selbst für Windows registrieren kann. Die Registrierung für Windows kann auf zwei Wegen erfolgen. Man soll zuerst versuchen, das Setup-Programm des Steuerelementes auszuführen. In dieser Arbeit ist es die von der Firma Vishay Micro-Measurements angebotene Installationsroutine. Die Registrierung wird in der Regel bei der Installation der Software vorgenommen. Man kann das Steuerelement auch über das Hilfsprogramm *regsvr32.exe* manuell registrieren. Der Vorgang sieht so aus:

- Öffne die MS-DOS-Eingabeaufforderung.
- Wechsele in das Verzeichnis, wo sich das ActiveX-Steuerelement VMMP3Control.dll befindet. Führe dann den Befehl regsvr32 aus:
  ```
  C:\Program Files (x86)\Vishay Micro-Measurements\Model P3 Strain
     Indicator and Recorder>regsvr32 VMMP3Control.dll
  ```

4.2.2. Verweis auf ActiveX-Steuerelement einrichten

Es muss noch ein Verweis auf dieses Steuerelement im Visual C++-Projekt eingerichtet werden. Dafür müssen folgende Schritte unternommen werden:

- Rufe den Menübefehl *Projekt/..Eigenschaften* auf, klicke im aufspringenden Dialogfeld auf den Schalter *Neuen Verweis hinzufügen* auf der Seite *Allgemeine Eigenschaften*. Gebe dann im daraufhin erscheinende Dialogfeld *Verweis hinzufügen* den Pfad des Speicherorts von ActiveX-Steuerelement VMMP3Control.dll ein und klicke auf *OK*. Die Abbildung 4.1 zeigt die Eigenschaften des neuen eingerichteten Verweises auf dieses Steuerelement.

Die in der VMMP3Control Library enthaltenen Eigenschaften und Methoden kann man folgendermaßen auflisten:

- Rufe den Menübefehl *Projekt/Vorhandenes Element hinzfügen* auf, gebe dann im Dialogfeld *Vorhandenes Element hinzfügen* den Pfad des Speicherorts von diesem Steuerelement ein, markiere die Datei VMMP3Control.dll und klicke auf *hinzufügen*. Ein neuer Knoten namens VMMP3Control.dll erscheint im Projektmappen-Explorer. Klicke doppelt auf ihn. Die VMMP3Control Library mit den zugehörigen Eigenschaften und Methoden wird dann in einem neuen Fenster namens Objektkatalog im Arbeitsbereich gezeigt (siehe Abbildung 4.2).

[2] http://msdn.microsoft.com/en-us/library/aa751972(v=vs.85).aspx

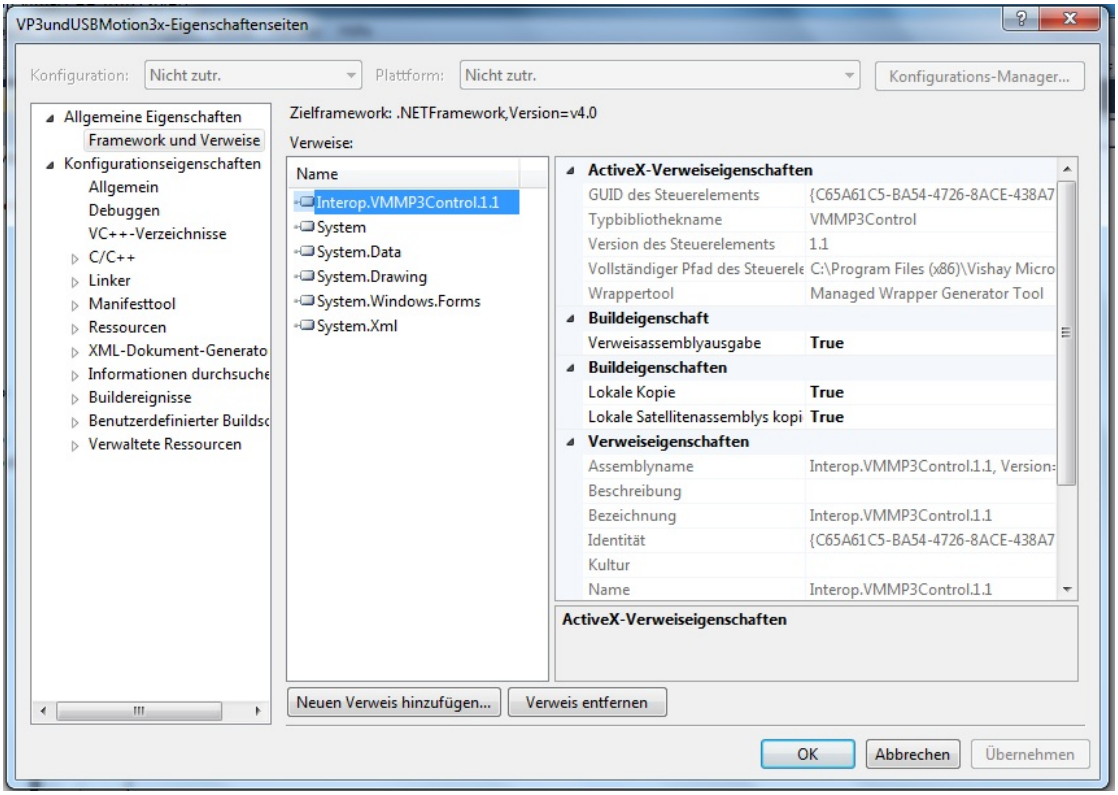


Abbildung 4.1.: ActiveX-Verweiseigenschaften

Visual Studio erzeugt beim Einrichten des Verweises eine Interop-Assembly namens Interop.VMMP3Control.1.1 automatisch. Die Typmetadaten in Form eines Manifest werden der in der Datei VMMP3Control.dll enthaltenen Typbibliothek entsprechend erstellt und dieser Assembly hinzugefügt. Das bedeutet, dass dieses ActiveX-Steuerelement in einer Visual C++-Anwendung wie übliche Steuerelemente benutzt werden kann[3]. Visual Studio speichert diese Assembly im neu angelegten Ordner interop und eine Kopie im Ordner Debug, wo sich auch die ausführbare Datei der Anwendung befindet.

Die Eigenschaften dieses Steuerelements können aber nicht dem Eigenschaftsfenster seines Containers hinzugefügt werden. Das bedeutet, dass sie im Programmcode abgefragt oder gesetzt, aber nicht beim Programmentwurf voreingestellt werden können.

4.2.3. Eigenschaften abfragen und setzen

Das in dieser Arbeit verwendete ActiveX-Steuerelement VMMP3Control.dll verfügt über keine Benutzeroberfläche und kann auch nicht über Ereignisse mit seinem Container interagieren. Es verfügt über zahlreiche Eingenschaften und Methoden. Diese sind im Anhang A aufgelistet.

In einer Visual C++-Anwendung können die Eigenschaften von VMMP3Control.dll folgendermaßen abgefragt bzw. gesetzt werden:

[3] http://msdn.microsoft.com/de-de/library/xwzy44e4(v=VS.100).aspx

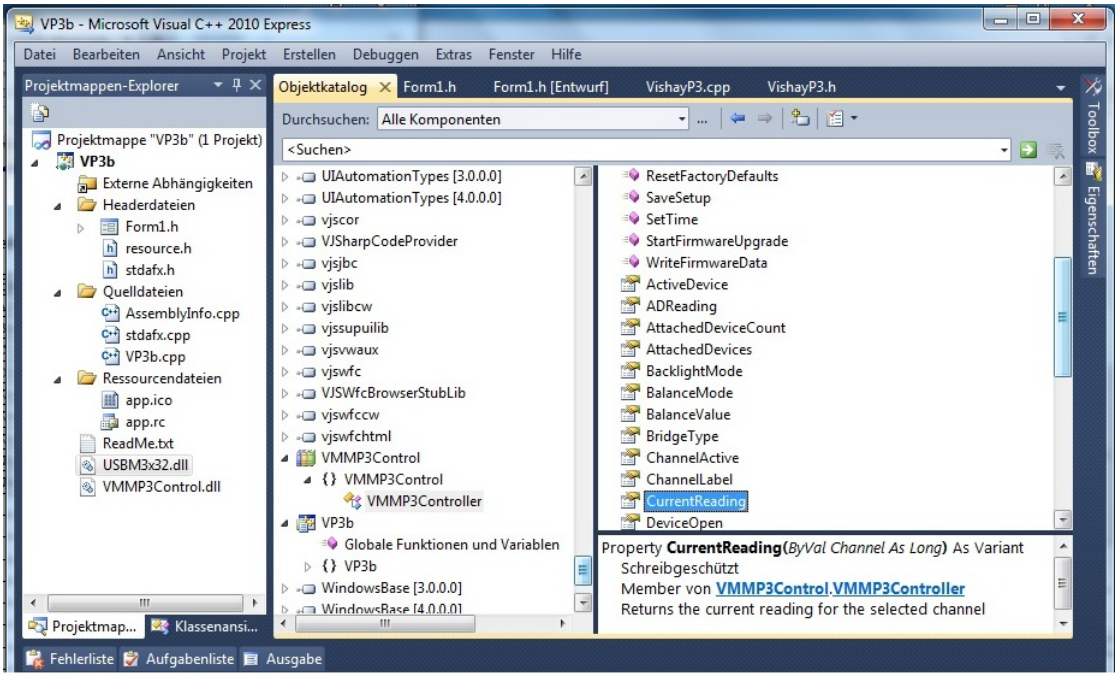


Abbildung 4.2.: VMMP3Control Library

Es soll zuerst eine Variable oder Membervariable einer Klasse, in der dieses Steuerelement eingesetzt werden soll, definiert werden. Ein Objekt vom Typ VMMP3controller wird erzeugt und dieser Variablen zugewiesen.

```
VMMP3Control::VMMP3Controller ^NewVP3 = gcnew VMMP3Controller();
```

Die Eigenschaften von VMMP3Control.dll können dann über die Membervariable `NewVP3` angesprochen werden:

```
NewVP3->DeviceOpen = false;

if (NewVP3->DeviceOpen)
{
  double kanal1 = Convert::ToDouble(NewVP3->CurrentReading[1]);
} else NewVP3->DeviceOpen = true;
```

Die Eigenschaft `DeviceOpen` muss auf `true` gesetzt werden, befor das Programm auf andere Eingenschaften zugreift. Die Nur-Lesen-Eigenschaft `CurrentReading[i]` enthält die aktuellen Messwerte vom Kanal `i` mit `i = 1,2,3,4`. Das Programm kann jede Zeit diese Eigenschaft abfragen.

4.3. .NET Framework

Die in diesem Abschnitt verwendeten Materialien zum Tehma „.NET Framework“ sind [5] und [9]. In beiden sind die hier benötigen Grunglagen beschrieben.

Das .NET-Framework ist eine Zielplattform für die Anwendungen, die mit den sogenannten .NET-Sprachen geschrieben wurden, und besteht aus zwei Hauptkomponenten:

der Common Language Runtime (CLR) und der .Net Framework-Klassenbibliothek.

Die CLR ist eine Laufzeitumgebung, die der Java Virtual Machine ähnlich, einen prozessorunspezifischen Zwischencode, den sogenannten Intermediate Language Code (IL-Code) ausführen kann. Wird ein in C++ oder anderen .NET-Sprachen geschriebenes Programm für das .NET-Framework kompiliert, übersetzt der Compiler den Quelltext in den IL-Code. Dieser IL-Code wird dann als verwalteter Code unter der Kontrolle der CLR ausgeführt, d.h., die CLR lädt das Programm nach dem Programmstart, lässt den IL-Code von ihrem Just-In-Time Compiler (JIT) bei Bedarf modulweise in einen prozessorspezifischen Code übersetzen, wobei die Module in der Regel den Methoden einer Anwendung entsprechen.

Die CLR übernimmt gleichzeitig noch die Aufgaben wie die Speicherverwaltung (Reservierung eines Speicherblocks für den von ihr verwalteten Heap, Garbage Collection etc.), die Überwachung der Code-Ausführung und die Durchsetzung von Sicherheitsfeatures (z.B. Von wem oder von wo aus eine Assembly aufgerufen werden darf und wann sie das Recht auf die Registry zuzugreifen bekommt).

Die .Net Framework-Klassenbibliothek, die von allen .NET-Sprachen benutzt werden kann, umfasst mehrere Tausend Klassen. Diese sind in unterschiedliche Namespaces organisiert. Unter dem Root-Namespace `System` sind Sub-Namespaces, die Klassen für bestimmte Funktionalitäten enthalen. z.B.:

`System::IO`	Klassen für die Ein- und Ausgebe und den Zugriff auf Dateien
`System::XML`	Klassen für die Arbeit mit XML-Daten
`System::Data`	Klassen für den Zugriff auf Datenbank
`System::Windows::Forms`	Klassen für GUI-Anwendungen (Windows Forms-Anwendungen)

Das .Net Framework schreibt allen .NET-Sprachen nicht nur eine gemeinsame Zielsprache (IL) vor, sondern auch ein einheitliches und verbindliches Typsystem, nämlich das Common Type System (CTS), und ermöglicht damit die Sprachinteroperabilität in .NET-Sprachen. Das CTS ist objektorientiert und unterstützt alle schon vorher allgemein anerkannten OOP-Konzepte wie Klassen, Schnittstellen, Einfachvererbung für Klassen, Mehrfachvererbung für Schnittstellen, Polymorphie, virtuelle Methoden etc.. Darüber hinaus bietet das CTS noch neue Konzepte wie Eigenschaften (Properties), Indexer (eine besondere Form von Eigenschaften) und Delegates (typisierte Verweise auf Methoden). Alle Typen des CTS sind von der Wurzelklasse `System.Object` abgeleitet. Das gilt auch für Werttypen wie ganze Zahlen (`System::Int32`) oder logische Werte (`System::Boolean`).

Das CTS ermöglicht also es den Programmierern, in einem Programm Klassenbibliotheken, die in anderen .NET-Sprachen geschrieben sind, problemlos zu verwenden. Außerdem wird das .Net Framework von den meisten der bisherigen Windows-Betriebssysteme und allen zukünftigen Windows-Betriebssystemen unterstützt. Das heißt, ein für das .NET-Framework entwickeltes Programm kann auf jedem Computer, auf dem beispielsweise Windows 7, Windows XP oder ein zukünftiges Windows-Betriebssystem installiert wird, ausgeführt werden.

Die mit Hilfe vom .NET-Framework erstellten softwarekomponenten (EXE- oder DLL-Datei) sind zwar ebenfalls sprachunabhängig, liegen aber nicht mehr in Binärform, son-

dern in IL-Code vor. Eine oder mehrere Klassen werden zusammen mit den zugehörigen Metadaten zu einem Modul (Komponente) zusammengefasst, und ein oder mehrere Module können eine Assembly bilden. Assemblies enthalten neben den Modulen mit deren Code und Metadaten auch eigene Metadaten in einem Manifest. Eine .NET-Komponente ist also keine Assembly, sie wird vielmehr in einer Assembly ausgeliefert. Das Assembly-Manifest enthält alle für die Verwendung der Assembly notwendigen Informationen. Das NET-Sicherheitskonzept, die NET-Vesionsverwaltung und die NET-Sprachunabhäigkeit basieren auf diesen Informationen.

4.4. Visual C++

Das Softwarepaket Visual C++ ist ein Produkt von Microsoft und steht für die Erstellung von Anwendungen mit der Programmiersprache C++. Dieses Paket enthält u.a. eine integrierte Entwicklungsumgebung, zahlreiche Bibliotheken sowie Build-Werkzeuge. Die aktuelle Version von Visual C++ unterstützt die Syntaxerweiterungen C++/CLI, die die Schnittstelle von ANSI C++ zum .NET Framework definieren. Visual C++-Anwendungen können damit die Klassen und Typen aus der .NET-Bibliothek verwenden und von den Diensten der CLR profitieren.

C++/CLI erfüllt den Standard Common Language Infrastructure (CLI). Das .Net Framework ist nämlich eine Implementierung dieses Standards. In diesem Abschnitt werden einige Besonderheiten der C++/CLI-Schnittstelle, die für diese Arbeit nützlich sind, vorgestellt.

4.4.1. Verwalteter Klassentyp

Nach der Definition der Schnittstelle wird ein verwalteter Klassentyp mit dem Schlüsselwort `ref` definiert:

```
public ref class ManUSBMotion3x
{
// ...
};
```

Damit die Objekte (dieser Klasse) auf dem von der CLR verwalteten Heap angelegt werden, müssen noch zwei Bedingungen erfüllt sein:

- Die Typdefinitionen der jeweiligen Objekte müssen in die verwalteten Codes, d.h. mit dem Compiler-Schalter `/clr` kompiliert werden.
- Die Objekte müssen mit `gcnew` erzeugt und den mit `^` definierten Trackinghandles zugewiesen werden.

```
int main(array<System::String ^> ^args)
{
  ManUSBMotion3x^ NewMotion3x = gcnew ManUSBMotion3x(); // ...
  return 0;
}
```

Die CLR kann ein Objekt an einen neuen Speicherort innerhalb des verwalteten Heaps verschieben, wenn es Vorteile bringt. Im Gegensatz zu den mit `*` definierten Zeigern

kann ein Trackinghandle das Verschieben des Objekts, auf das es verweist, mit verfolgen und verweist damit stets auf dieses Objekt.

Das „gc“ in `gcnew` steht für Garbage Collection. Die Speicherverwaltung für die Objekte auf dem verwalteten Heap wird von der CLR übernommen, so dass sich der Programmierer nicht mehr darum kümmern muss. Das Nichtaufräumen von Speicherressourcen ist eine häufige Fehlerquelle in nicht verwalteten C++-Programmen.

4.4.2. Verwalteter und nicht verwalteter Code

In der .NET-Terminologie werden alle Programme, die für das .NET Framework geschrieben sind und unter der Kontrolle der CLR ausgeführt werden, als verwaltet (managed) und alle anderen, insbesondere ältere Programme, als nicht verwaltet (unmanaged) bezeichnet.

Man kann aus verwaltetem Code heraus nicht verwalteten, nativen Code aufrufen, sofern dieser in C++ geschrieben wurde. Man kann auch innerhalb eines C++-Quelltextes festlegen, welche Funktionen bzw. Klassen in nativen und welche in verwalteten Code kompiliert werden sollen. Dazu werden die Präprozessor-Direktiven `#pragma managed` und `#pragma unmanaged` verwendet. Die Einstellung, dass das Programm mit dem Compiler-Schalter `/clr` kompiliert wird, erzeugt IL-Code, erlaubt auch die Kombination mit nicht verwaltetem Code.

Das folgende Codestück zeigt, wie die DLL USBM3x32, die in dieser Arbeit zur Schrittmotorsteuerung verwendet wird, mit Hilfe der beiden Präprozessor-Direktiven in einer Windows Forms-Anwendung eingesetzt wird.

```
#pragma once
#include <windows.h>

// ...
#pragma unmanaged
  // Definition des Types der DLL-Funktion, die verwendet werden soll
  typedef unsigned char (__stdcall *LPGETXTARGET)(unsigned char,
                                                  unsigned char&, long&);

  HINSTANCE husbm3x32Dll;
  unsigned char ucstatus;
  // DLL Datei laden
  husbm3x32Dll=(HINSTANCE)LoadLibrary(L"USBM3x32.dll");

  int SetXtarget(unsigned char motorIndex, long lXtarget)
  {
    LPSETXTARGET            lpSetXtarget;
    // Die Einsprungadresse abfragen
    lpSetXtarget=(LPSETXTARGET)GetProcAddress(husbm3x32Dll, "SetXtarget");
    return lpSetXtarget(motorIndex, lXtarget, ucstatus);
  }
```

```
  // ...
#pragma managed
  public ref class Form1 : public System::Windows::Forms::Form
  {
  // ...
  // Zielposition von Motor 0 setzen
  private: System::Void textBox2_KeyPress(System::Object^  sender,
                   System::Windows::Forms::KeyPressEventArgs^  e)
    {
      if ( e->KeyChar == (char)13 )     //Die Taste ENTER ist gepresst
      {
        if (SetXtarget(0, Convert::ToInt64(this->textBox2->Text)) == 5)
        {
          this->textBox1->AppendText("USB Gerät nicht verfügbar! \n");
        } else this->textBox1->AppendText(GetXtarget(0) + "\n");
      } // GetXtarget(i) gibt die akt. Position von Motor i zurück
    }
  };
```

Die Funktion `SetXtarget` und die Klasse `Form1` können auf dieselbe Weise zusammen arbeiten, wenn sie in separaten Dateien definiert werden.

4.4.3. Indizierte Eigenschaften

In verwalteten Klassen kann man die Get- und Set-Methoden durch die mit dem Schlüsselwort `property` deklarierten Eigenschaften ersetzen. Visual C++ verfügt über eine besondere Form von Eigenschaften, nämlich indizierte Eigenschaften, die es erlauben, über den Index auf die Felder einer Klasse zuzugreifen.

Indizierte Eigenschaften kann man mit dem Schlüsselwort `default` oder statt dessen mit eigenen Eienschaftennamen definieren. Der Indextyp wird in eckigen Klammern angegeben. Der indizierte Zugriff auf Felder erfolgt dann über die Objektnamen oder über die Eienschaftennamen.

```
  public ref class Messdaten
  {
    array<double> ^currentReading;

  public:
    Messdaten()
    {
      // ein verwaltetes Array mit 4 Elementen des Datentyps double
      currentReading = gcnew array<double>(4);
    }

    property double CurrentReading[int]
    {
      double get (int i)
      {
```

```
      return currentReading[i];
    }

    void set (int i, double wert)
    {
      currentReading[i] = wert;
    }
  }
};
```

Die Eigenschaft `CurrentReading[i]` mit `i =0,1,2,3` kann folgendermaßen angesprochen werden. Auf gleiche Weise werden die Messwerte vom Messgerät Vishay P3 in den Computer eingelesen (siehe Abschnitt 4.2.3).

```
int main(array<System::String ^> ^args)
{
  Messdaten ^daten = gcnew Messdaten();

  daten->CurrentReading[0] = Convert::ToDouble(Console::ReadLine());
  Console::WriteLine("CurrentReading: " + daten->CurrentReading[0]);
  return 0;
}
```

4.4.4. Generische Auflistungsklasse List<T>

Zur Implementierung dynamischer Datenstrukturen stellt die .Net Framework-Bibliothek im Namensraum `System::Collections::Generic` einen Satz von generischen Auflistungsklassen bereit. Dazu gehört die Klasse `List<T>`. Der Datentyp-Platzhalter `T` in spitzen Klammern bedeutet, dass diese Klasse typisiert ist. Bei der Deklaration und Instanzierung muss statt `T` ein konkreter Datentyp angegeben werden:

```
List<double> ^listMesswert = gcnew List<double>(5);
```

In runden Klammern ist die Anfangskapazität der Auflistung `listMesswert` angegeben. Die Kapazität einer Auflistung wird automatisch erhöht, wenn die Anzahl der Elemente die aktuelle Kapazität übersteigt. Das vereinfacht die Verwendung von Auflistungen, kann aber ihre Leistung negativ beeinflussen, weil die Elemente in den neu zugewiesenen Speicher kopiert werden müssen [4].

Das Hinzufügen und Entfernen von Elementen können durch Aufruf der Methoden `void Add(T item)` und `void RemoveAt(int Index)` erfolgen:

```
listMesswert->Add(CurrentReading[0]);
listMesswert->RemoveAt(0);
```

Der double-Wert in `CurrentReading[0]` wird am Ende der Auflistung `listMesswert` hinzugefügt. Das Element an der Position `0` dieser Auflistung wird entfernt und alle Elemente dahinter werden nach vorne geschoben.

Die Anzahl der Elemente, die sich zurzeit in einer `List`-Auflistung befinden, kann über

[4] http://msdn.microsoft.com/de-de/library/akyhke97.aspx

die Eigenschaft `Count` ermittelt werden. Werden z.B. alle Elemente einer Auflistung durch Aufruf der Methode `Clear()` entfernt, so gibt die Eigenschaft `Count` bei Abfrage den Wert Null zurück.

Im gegensatz zu den `Queue<T>`-Auflistungen erlauben die `List<T>`-Auflistungen auch den indizierten Zugriff auf ihre Elemente:

```
double durchschnitt = 0, summe = 0;
for (int i; i < listMesswert->Count; ++i )
    summe += listMesswert[i];

durchschnitt = summe/listMesswert->Count;
```

5. Entwurf

Das zu erstellende Anwendungsprogramm soll zwei Hauptaufgaben erledigen, nämlich die Schrittmotoren zu steuern und die Messwerte vom Messgerät einzulesen. Das Programm soll wie folgt vorgehen:

1. Fragen nach dem Radius und der Tiefe der Kreisbahn.
2. Steuern der Schrittmotoren über die Steuerkarte, damit die Kreisbahn mit dem vom Benutzer eingegebenen Radius und der ebenfalls vom Benutzer eingegebene Tiefe gefräst wird.
3. Einlesen der Messwerte in bestimmten Zeitabständen vom Messgerät Vishay P3 in den Computer. Warten bis sich die Messwerte von allen eingesetzten Messkanälen kaum verändern und speichern für jeden Kanal einen aktuellen Messwert in einer CSV-Datei. Beginnen dann wieder bei 1.

In den folgenden Abschnitten werden für zwei wichtige Aufgaben, die bei einem Messvorgang erledigt werden müssen, geeignete Lösungen erarbeitet.

5.1. Aufnehmen der Messwerte

Um festzustellen, dass sich die Messwerte nach jeder Vertiefung der Kreisbahn nicht mehr verändern, muss während des gesamten Messvorgangs für jeden Messkanal eine bestimmte Anzahl der am neuesten eingelesenen Messwerte immer vorhanden sein.

Die Messwerte werden in bestimmten Zeitabständen gleichzeitig von drei Messkanälen des Messgerätes eingelesen und entsprechend in drei Auflistungen gespeichert, die die gleiche Kapazität haben. Erreicht die Anzahl der Messwerte in allen drei Auflisungen die gegebene Kapazität, so wird vor dem Hinzufügen eines neuen Messwertes am Ende einer der drei Auflistungen ein Messwert am Anang dieser Auflistung gelöscht. Nach jedem Hinzufügen werden für jede Auflistung die in ihr befindlichen Messwerte miteinander verglichen. Jedes Mal, wenn sich die Messwerte in allen drei Auflistungen nicht mehr verändern, wird für jede Auflistung jeweils ein Messwert in einer CSV-Datei gespeichert und auf dem Bildschirm gezeigt.

Auf diese Weise werden die stabilen Messwerte für jede Vertiefung der Kreisbahn aufgenommen. Die Messwerte, die sich in einer CSV-Datei befinden, können von anderen Anwendungsprogrammen wie Microsoft Excel komfortabel verwendet werden.

Für diese Aufgabe soll die `List<double>`-Auflistung eingesetzt werden, mit deren Hilfe die Messwerte bequem und effizient hinzugefügt, gelöscht und miteinander verglichen werden können. Die `Queue<double>`-Auflistung, d.h. die FIFO-Auflistung ist nicht geeignet, da sie keinen indizierten Zugriff auf ihre Elemente erlaubt.

Das Einlesen und Vergleichen der Messwerte sollen nach dem Start der Messung regelmäßig stattfinden. Dies kann auf zwei Wegen erfolgen. Der eine besteht darin, ein

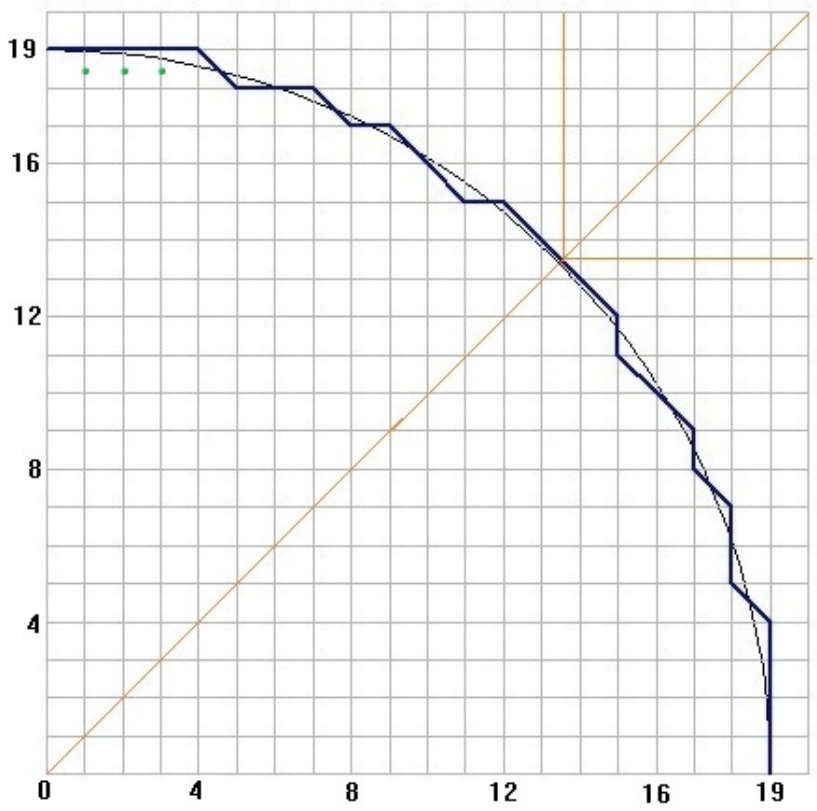


Abbildung 5.1.: Viertelkreisbahn

`timer`-Steuerelement in der Anwendung einzusetzen, das sein Tick-Ereignis in bestimmten Zeitabständen auslöst. Die Tick-Ereignisbehandlungsmethode sorgt dann dafür, für jeden Messkanal jeweils einen Messwert in den Computer einzulesen und mit den anderen Messwerten zu vergleichen. Es kann auch ein Thread-Objekt erzeugt werden, das im Hintergrund diese Aufgabe erledigt.

5.2. Bestimmen der Punkte auf der Kreisbahn

Ein Punkt (x, y) auf der Kreisbahn kann über die Sinus- und Cosinus-Funktion wie folgt bestimmt werden:

$$(x, y) = (r \cdot \cos(\alpha), r \cdot \sin(\alpha)) \tag{5.1}$$

wobei sich der Mittelpunkt der Kreisbahn am Koordinatenursprung $(0, 0)$ befindet und r der Radius der Kreisbahn ist. Der Winkel α ist derjenige, der von der x-Achse und der Verbindungsstrecke zwischen dem Ursprung und dem Punkt (x, y) eingeschlossen wird.

Der Radius r soll hier in Schrittzahl des Schrittmotors umgerechnet werden, indem der vom Benutzer eingegebene Betrag des Radius (in mm) durch die Schrittauflösung (in mm/Schritt) des Schrittmotors dividiert wird. Der Fräser, der die gegebene Kreisbahn abfahren soll, wird von den Schrittmotoren für die X- und Y-Achse angetrieben und kann sich daher nur in der X-, Y-Richtung oder der Diagonalrichtung bewegen. Wie in der Abbildung 5.1 gezeigt, bewegt sich der Fräser tatsächlich entlang der blauen Linie.

Die Anzahl N von den Schritten, die entweder einer der beiden Schrittmotoren oder die beiden gleichzeitig fahren müssen, damit die komplette Kreisbahn gefräst wird, kann dann mit Hilfe der Formel (5.1) und der Symmetrieeigenschaft der Kreisbahn wie folgt abgeschätzt werden:

$$N = 4 \cdot (r \cdot \cos(\alpha) + r \cdot \sin(\alpha)) = 8 \cdot r \cdot \cos(\pi/4) \leq 6 \cdot r \tag{5.2}$$

Werden die berechneten Koordinaten der einzelnen Punkte, die der Fräser nacheinander anfahren soll, in einer `List`-Auflistung gespeichert, so muss die Kapazität dieser Auflistung nicht größer als die Zahl N sein.

5.2.1. Bestimmen der Punkte aufgrund der Symmetrie

Die Kreisbahn ist eine achsensymmetrische Figur. Jede Gerade durch ihren Mittelpunkt ist eine Symmetrieachse. Wird ein Punkt (x, y) für das Fräsen der Kreisbahn bestimmt, so lässt sich z.B aufgrund der X-Achsensymmetrie feststellen, dass auch der Punkt $(x, -y)$ dafür nötig sein kann.

Die Spiegelung eines Punktes an der X-, Y-Achse und den beiden Diagonalen kann von

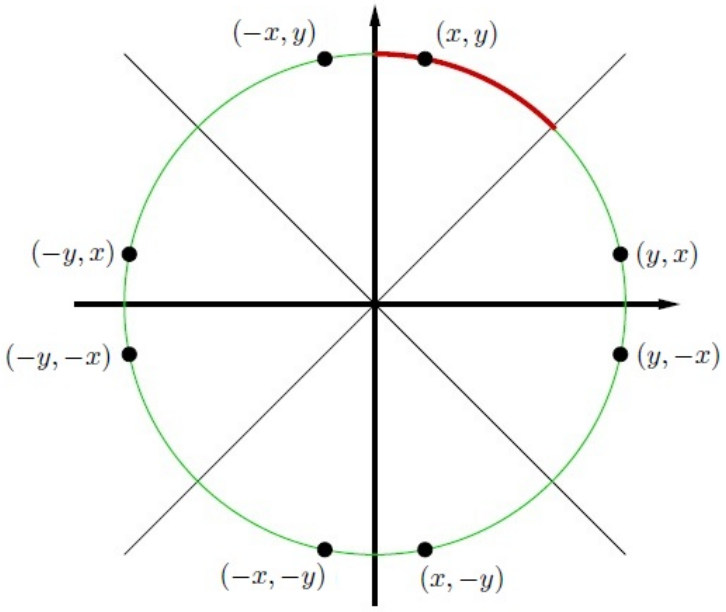


Abbildung 5.2.: Symmetrie der Kreisbahn

einem Programm quasi ohne Rechenaufwand erledigt werden. Um die Rechenzeit beim Erzeugen der Kreisbahn-Daten zu reduzieren, wird die Symmetrieeigenschaft der Kreisbahn von unserem Programm wie folgt genutzt:

- Das Programm berechnet zuerst nur die Koordinaten derjenigen Punkte, die für das Fräsen einer Achtelkreisbahn benötigt sind, also z.B. wie in der Abbildung 5.2 gezeigt, die Punkte im Bereich zwischen 12:00 und 13:30 Uhr.

- Dann leitet das Programm für das Fräsen einer weiteren Achtelkreisbahn die Koordinaten der Punkte ab, indem es die für das Fräsen der vorherigen Achtelkreisbahn bestimmten Punkte an der X-, Y-Achse oder den beiden Diagonalen spiegelt. Dies wiederholt solange, bis die Punkte für das Fräsen der lezten Achtelkreisbahn bestimmt werden.

Wird also ein Punkt (x, y) für das Fräsen der Kreisbahn bestimmt, so werden auch die Punkte (y, x), $(y, -x)$, $(x, -y)$, $(-x, -y)$, $(-y, -x)$, $(-y, x)$ und $(-x, y)$ für das Fräsen der Kreisbahn bestimmt.

5.2.2. Berechnen der Koordinaten der Punkte

Zur Berechnung der Punkte auf der Kreisbahn kann der Bresenham Algorithmus[6][8] zum Einsatz kommen. Dieser Algorithmus wird vor allem in der Computergrafik häufig verwendet, weil er ohne Multiplikation, ohne Auswertung der Wurzel und der Sinus-, Cosinus-Ausdrücke auskommt, und somit bei Bildern mit sehr vielen Kreisen die Rechenzeit erheblich einspart.

Nach diesem Algorithmus soll das Programm den Fräser über zwei Schrittmotoren wie folgt steuern: Der Fräser wird am Anfang an den Punkt $(0, r)$ positioniert und dann nach unten rechts bis zum Winkel von 45° fortgesetzt. Also, wenn sich der Fräser in diesem Achtel der Kreisbahn am Punkt (x, y) befindet, dann muss der nächste Punkt, den er in einem Schritt erreichen soll, entweder $(x + 1, y)$ oder $(x + 1, y - 1)$ sein. Das Programm trifft die Entscheidung, indem es überprüft, welcher der beiden Punkte näher am Kreis (die schwarze Linie in Abb. 5.1) liegt.

Sei für einen Punkt (x, y) die Funktion $F(x, y) = x^2 + y^2 - r^2$ gegeben, dann gilt:

$$\begin{aligned} F(x, y) &= 0, \text{ falls } (x, y) \text{ auf dem Kreis} \\ F(x, y) &< 0, \text{ falls } (x, y) \text{ innerhalb des Kreises} \\ F(x, y) &> 0, \text{ falls } (x, y) \text{ außerhalb des Kreises} \end{aligned}$$

Das Programm berechnet $d = F(x + 1, y - 1/2)$, also den F-Wert des Punktes M (wie grüne Punkte in Abb. 5.1), der in der Mitte zwischen den beiden Punkten $(x+1, y)$ und $(x + 1, y - 1)$ liegt. Die Entscheidung wird wie folgt getroffen:

Fall 1: $d < 0 \Rightarrow$ M ist innerhalb des Kreises $\Rightarrow$ Punkt $(x + 1, y)$ ist näher am Kreis
$\Rightarrow$ Aus (x, y) wird $(x + 1, y)$
Fall 2: $d \geq 0 \Rightarrow$ M ist außerhalb des Kreises $\Rightarrow$ Punkt $(x + 1, y - 1)$ ist näher am Kreis
$\Rightarrow$ Aus (x, y) wird $(x + 1, y - 1)$

Da das Fräsen am Punkt $(0, r)$ beginnt, wird d initialisiert mit

$$d = F(0 + 1, r - \frac{1}{2}) = 1 + (r^2 - r + \frac{1}{4}) - r^2 = \frac{5}{4} - r$$

Sei

$$d_{alt} = F(x + 1, y - \frac{1}{2}) = (x + 1)^2 + (y - \frac{1}{2})^2 - r^2$$

gegeben. Wenn $d_{alt} < 0$ ist, dann wird der Punkt $(x+1, y)$ ausgewählt. Der neue Wert von d ergibt sich als

$$\begin{aligned} d_{neu} &= F(x+2, y-\frac{1}{2}) = (x+2)^2 + (y-\frac{1}{2})^2 - r^2 \\ &= d_{alt} + 2x + 3 = d_{alt} + \Delta_x \end{aligned}$$

Wenn $d_{alt} \geq 0$ ist, dann wird der Punkt $(x+1, y-1)$ ausgewählt. Der neue Wert von d ergibt sich als

$$\begin{aligned} d_{neu} &= F(x+2, y-\frac{3}{2}) = (x+2)^2 + (y-\frac{3}{2})^2 - r^2 \\ &= d_{alt} + 2x - 2y + 5 = d_{alt} + \Delta_{xy} \end{aligned}$$

wobei zu beachten ist, dass die Inkremente Δ_x und Δ_{xy} ständig anwachsen. Es handelt sich bei $d = F(x+1, y-1/2)$ um eine quadratische Gleichung, ihre ersten Ableitungen sind deswegen nicht konstant. Die beiden werden also in jedem Schritt um 2 erhöht, falls der Punkt $(x+1, y)$ als der nächste ausgewählt wird. Sonst werden Δ_x um 2 und Δ_{xy} um 4 erhöht. Außerdem sollen Δ_x und Δ_{xy} jeweils mit $x = 0$ und $y = r$ initialisiert werden, d.h., zum Beginn des Fräsens sind $\Delta_x = 3$ und $\Delta_{xy} = -2r + 5$.

Da d mit dem Wert $5/4 - r$ initialisiert und dann nur ganzzahlig inkrementiert wird, muss d in der Menge $\{..., -\frac{3}{4}, \frac{1}{4}, \frac{5}{4}, ...\}$ liegen. Es muss also gelten:

$$d < 0 \Leftrightarrow d \leq -\frac{3}{4} \Leftrightarrow d - \frac{1}{4} \leq -1 \Leftrightarrow d - \frac{1}{4} < 0$$

Es kann daher d auch mit $(5/4 - r) - 1/4 = 1 - r$ initialisiert werden. Im Programm wird d dann nur ganzzahlige Werte annehmen.

Nachdem die Punkte auf der ersten Achtelkreisbahn berechnet wurden, können die anderen Punkte auf der Kreisbahn durch die Spiegelungen, wie im letzten Abschnitt beschrieben, abgeleitet werden.

6. Implementierung

Vor der Implementierung müssen zwei Entscheidungen getroffen werden. Das betrifft zunächst das Betriebssystem. Weil das Anwendungsprogramm zwei DLLs verwenden muss, so soll es unter Windows implementiert werden. Damit das Anwendungsprogramm auch unabhängig von der Betriebssystemversion eingesetzt werden kann, soll es für das .NET-Framework geschrieben werden. Eine weitere Entscheidung betrifft die Programmiersprache. Wenn es sich bei diesem Anwendungsprogramm schon um eine .NET Framework basierende Anwendung handelt, kann es im Prinzip in beliebigen .NET-Sprachen implementiert werden, z.B. in C++/CLI, C# oder VB.NET. Hier wird C++/CLI ausgewählt, nur weil der Herseller der DLL USBM3x32 für die Verwendung dieser DLL brauchbaren C++-Quellcode angeboten hat.

6.1. Einrichten der Entwicklungsumgebung

Als Entwicklungsplattform wurde Windows 7 verwendet. Zum Erstellen und Test des Anwendungsprogrammes wurde das Softwarepaket Microsoft Visual C++ 2010 Express installiert. Beim Testen und bei der Fehlersuche war der integrierte Debugger zum Einsatz gekommen.

Das Anwendungsprogramm wird in Visual C++ 2010 Express in Form eines Projektes verwaltet. Die Projektvorlage Windows Forms-Anwendung wurde beim Anlegen des Projektes ausgewählt, damit das Anwendungsprogramm innerhalb des .NET Frameworks ausgeführt wird. Für diese Projektvorlage ist aber per Voreinstellung dem Visual C++-Compiler der Compiler-Schalter */clr:pure* übergeben. Da dieses Programm sowohl verwalteten als auch nicht verwalteten Code enthält, wurde der Compiler-Schalter */clr* wieder ausgewählt (über *Projekt/Eigenschaften/Konfigurationseigenschaften/Allgemein/Common Language Runtime-Unterstützung*).

Das ActiveX-Steuerelement VMMP3Control.dll wurde für Windows registriert und ein Verweis auf dieses im angelegten C++-Projekt eingerichtet (siehe Abschnitt 4.2.1 und 4.2.2). Die DLL USBM3x32 musste nicht registriert werden, musste aber in das selbe Verzeichnis wie die ausführbare Datei des Programmes kopiert werden. Die von ihr zur Verfügung gestellten Funktionen wurden im Programmcode deklariert und mit Hilfe der Windows-API-Funktionen aufgerufen.

Das Messgerät Vishay P3 und die Schrittmotorsteuerkarte USBMotion3XII wurden über USB Kabel mit dem Computer verbunden und vom ihm erkannt. Die Schrittmotorsteuerkarte wurde noch von einer Gleichspannungsquelle (7 – 34 VDC) gespeist.

6.2. Implementierte Klassen

Im Laufe des Implementierungsprozesses ergaben sich folgende Klassen:

- Die Formularklasse `Form1`, die von der Basisklasse `Form` abgeleitet wurde. Das Gerüst dieser Klasse wurde beim Anlegen des Projektes automatisch erzeugt. Diese Klasse besitzt folgende Methoden:

  `void InitializeComponent(void)` ist vom Windows Forms-Designer verwaltet. In dieser Methode werden alle Komponenten der Benutzerschnittstelle deklariert und mit den gewünschten Werten initialisiert. Darüber hinaus werden auch Events registriert.

  `void dateienAnlegen()` legt zur Speicherung der Messwerte und Koordinaten der für das Fräsen der Kreisbahn ausgewählten Punkte zwei Dateien `MessDaten.csv` und `KreisbahnDaten.txt` an.

  `void messvorgang()` speichert jeweils einen Messwert für die eingesetzten Messkanäle in der Datei `MessDaten.csv`, wenn sich alle Messwerte von diesen Kanälen nicht mehr verändern.

  `System::Void button1_Click` wird aufgerufen, wenn der Benutzer den Button Starten in der Benutzerschnittstelle anklickt. Diese Methode stellt die Verbindung zum Messgerät Vishay P3 her.

  `System::Void button2_Click` wird aufgerufen, wenn der Benutzer den Button Beenden in der Benutzerschnittstelle anklickt. Diese Methode löscht die hergestellte Verbindung zum Messgerät Vishay P3.

  `System::Void button3_Click` wird aufgerufen, wenn der Benutzer den Button Homing in der Benutzerschnittstelle anklickt. Diese Methode positioniert den Fräser an den Punkt $(0, 0, 0)$.

  `System::Void timer1_Tick` wird aufgerufen, sofern das im Programm eingesetzte `timer`-Steuerelement sein Tick-Ereignis auslöst. In dieser Methode wird den Zustand des Hintergrundthreads `threadKreisbahn` abgefragt und die Methode `messvorgang` aufgerufen.

  `SchrittmotorSteuerkarteInit()` wird vom Konstruktor der Klasse `Form1` aufgerufen. Diese ruft wieder die Methoden `managed_SchrittmotorDLLInit` und `managed_SetVmax` von der Klasse ManUSBMotion3x auf, damit die Schrittmotorsteuerkarte initialisiert und die maximale Geschwindigkeit von drei Motoren auf einen gegebenen Wert gesetzt wird.

  `System::Void textBox2_KeyPress`,
  `System::Void textBox3_KeyPress` oder
  `System::Void textBox4_KeyPress` wird aufgerufen, wenn der Benutzer einen Wert (in mm) im Eingabefeld für X-, Y- oder Z-Koordinate eingibt und dann die ENTER-Taste drückt. Die aufgerufene Methode nimmt den Wert entgegen, rechnet ihn in Schrittzahl des Schrittmotors um und ruft dann die Methode `managed_SetXtarget` von der Klasse ManUSBMotion3x auf, damit die eingegebene Zielposition angefahren wird.

`System::Void textBox6_KeyPress` wird aufgerufen, wenn der Benutzer einen Wert (in mm) im Eingabefeld für den Radius der Kreisbahn eingibt und dann die ENTER-Taste drückt. Sie nimmt den Wert entgegen, rechnet ihn in Schrittzahl des Schrittmotors um und startet ein Hintergrundthread namens `threadKreisbahn`.

`void achtelKreisbahn` berechnet die Koordinaten der ausgewählten Punkte für die erste Achtelkriesbahn nach dem Bresenham Algorithmus.

`void datenSchickenSpeichern1` lässt den Schrittmotor für die X-Achse x Schritte oder den Schrittmotor für die Y-Achse y Schritte oder die beiden gleichzeitig die angegebenen Schritte machen, indem sie die Methode `managed_SetXYZtarget` von der Klasse ManUSBMotion3x aufruft. Ist der Aufruf erfolgreich, so wird die Koordinaten des zu erreichenden Punktes in der Datei `KreisbahnDaten.txt` gespeichert.

`void datenSchickenSpeichern` entscheidet nach der aktuellen und der nächsten Position des Fräsers, welche Schrittmotoren in Bewegung gesetzt werden sollen, ruft dann die Methode `datenSchickenSpeichern1` auf.

`void kreisbahnFräsen()` wird vom Hintergrundthreads `threadKreisbahn` aufgerufen. Diese ruft zuerst die Methode `achtelKreisbahn` auf, leitet dann die anderen Punkte auf der Kreisbahn durch die Spiegelungen ab. Anschließend ruft diese die Methode `datenSchickenSpeichern` einmal pro Sekunde auf, bei jedem Aufruf werden als Argumente die aktuelle Position und die nächste Position des Fräsers übergeben.

- Die Klasse `USBMotion3x` fasst die Methoden zur Steuerung des Schrittmotors zusammen:

  `int SchrittmotorDLLInit()` wird zur Initialisierung der DLL aufgerufen. Diese ruft die DLL-Funktionen `USBMCCreate` und `USBMCinit` auf.

  `int SchrittmotorDLLEntladen()` wird zum Entladen der DLL aufgerufen. Diese ruft wieder die DLL-Funktion `USBMCDestroy` und die Windows-API-Funktion `FreeLibrary` auf.

  `int SetXtarget` wird zum Festlegen der Zielposition des Schrittmotors für die X-, Y- oder Z-Achse aufgerufen. Diese ruft die DLL-Funktion `SetXtarget` auf.

  `int SetXYZtarget` wird für den Fall, dass die Zielpositionen von mehreren Schrittmotoren gleichzeitig festgelegt werden sollen, aufgerufen. Diese Methode ruft die DLL-Funktion `SetXYZtarget` auf.

  `long GetXtarget` wird zum Abfragen der Zielposition des Schrittmotors aufgerufen. Diese ruft die DLL-Funktion `GetXtarget` auf.

  `int SetVmax` wird zum Festlegen der maximalen Geschwindigkeit des Schrittmotors aufgerufen. Diese ruft die DLL-Funktion `SetVmax` auf.

- Die verwaltete Klasse `ManUSBMotion3x` ist die Wrapper-Klasse der nicht verwalteten Klasse `USBMotion3x`, hat daher die gleiche öffentliche Funktionalität wie die Klasse `USBMotion3x`. In der Klasse wird ein Zeiger namens `UnUSBMotion3x` vom Typ der Klasse `USBMotion3x` mit `*` deklariert. Im Konstruktor `ManUSBMotion3x()` wird ein Objekt der Klasse `USBMotion3x` mit `new` erzuegt und diesem Zeiger zu-

gewiesen. Im Destruktor `~ManUSBMotion3x()` wird das nicht verwaltete Objekt gelöscht. Diese Klasse besitzt folgende methoden:

```
int managed_SchrittmotorDLLInit()
int managed_SchrittmotorDLLEntladen()
int managed_SetXtarget
int managed_SetXYZtarget
long managed_GetXtarget
int managed_SetVmax
```

Diese Methoden tun nichts anderes, als die entsprechende Methode der klasse `USBMotion3x` über den Zeiger `UnUSBMotion3x` aufzurufen. Ein Objekt dieser Klasse wird in der verwalteten Klasse `Form1` erzeugt und dem Trackinghandle `NewMotion3x` zugewiesen. Die DLL-Funktionen können dann über dieses Trackinghandle in der Klasse `Form1` aufgerufen werden.

6.3. Benutzerschnittstelle

Die Abbildung 6.1 zeigt die Benutzerschnittstelle des in dieser Arbeit erstellten Anwendungsprogrammes. Die Benutzerschnittstelle setzt sich aus einem Ausgabefenster, drei Buttons und vier Eingabefeldern zusammen.

Im Ausgabefenster werden folgende Informationen angezeigt:

- die Benutzereingaben für die Zielpositionen von drei Schrittmotoren, den Radius der Kreisbahn
- die aktuellen Positionen von drei Schrittmotoren, die Koordinaten der ausgewählten Punkte auf der ersten Achtelkreisbahn sowie die erfassten Messwerte
- die Meldungen wie „Die Kreisbahndaten wurden erfolgreich an die Steuerkarte geschickt!“ oder “Die Zielposition für Motor X konnte nicht ins Register geschrieben werden!“

Die Benutzer kann den Messvorgang starten, beenden oder den Fräser an den Punkt $(0, 0, 0)$ positionieren, indem er den Button Starten, Beenden oder Homing klickt. Der in einem Eingabefeld eingegebene Wert muss vom Benutzer mit der Enter-Taste bestätigt werden. Der eingegebene Wert für den Radius der Kreisbahn soll nicht größer als $1,0mm$ sein.

6.4. Testergebnisse

Beim Test wurden an drei Messkanäle des Messgerätes normale Widerstände angeschlossen. Die Widerstandsänderungen von DMS wurden durch die Wärmung dieser Widerstände simuliert. Die Abbildung 6.2 zeigt bei einem Test in der Datei `MessDaten.CSV` gespeicherte Messwerte. Dabei passierten die Widerstandsänderungen zuerst beim Messkanal3, dann beim Messkanal2 und schließlich beim Messkanal1. Die durch eine Linie gekennzeichneten Werte wurden gespeichert, weil das Programm davon ausging, dass die Messwerte von allen drei eingesetzten Messkanälen wieder stabil geworden waren.

Die Koordinaten der Punkte, die zum Fräsen einer Kreisbahn mit dem vom Benutzer

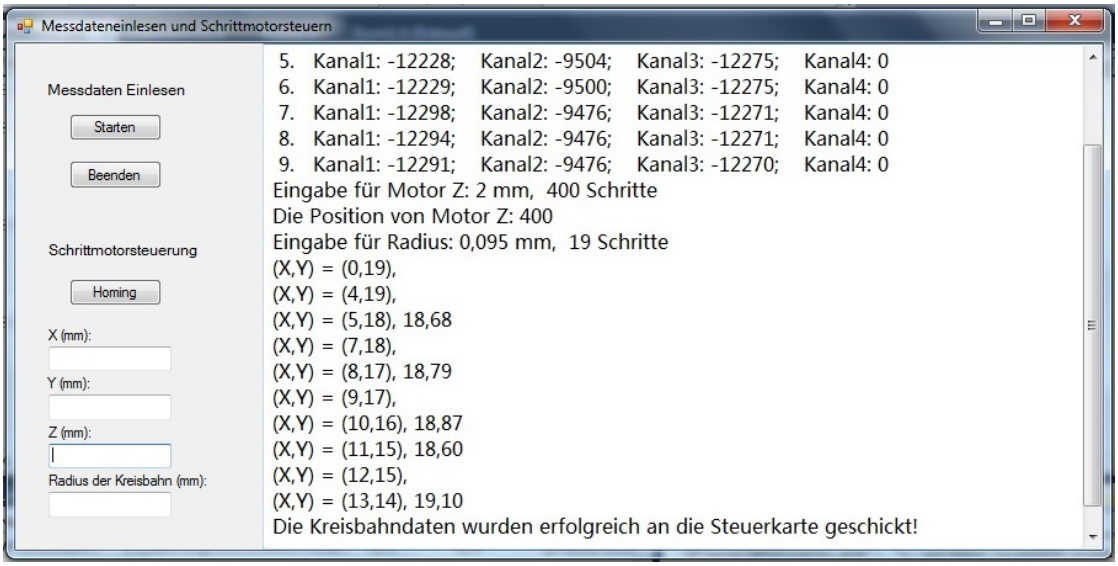


Abbildung 6.1.: Benutzerschnittstelle

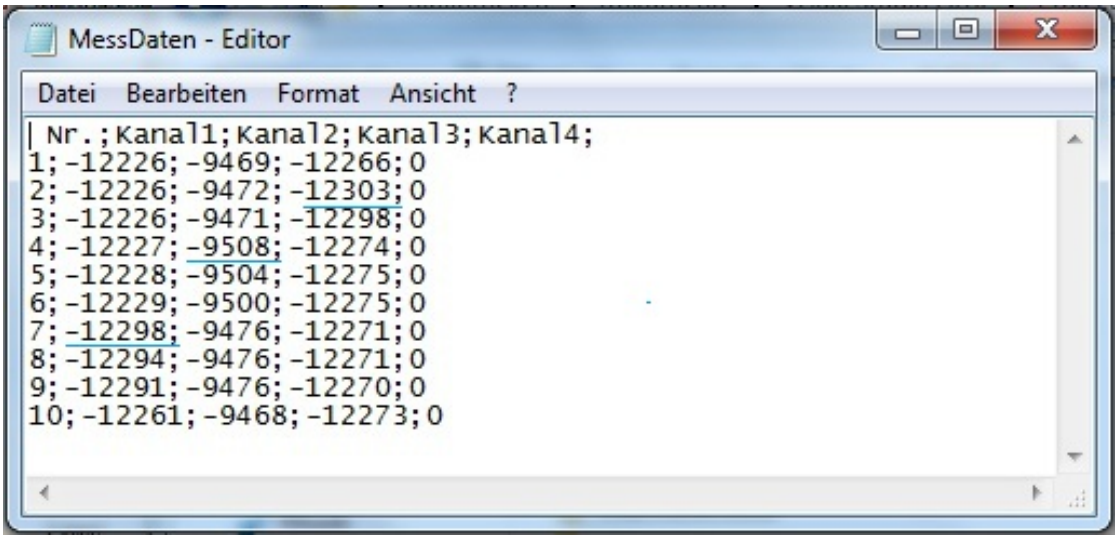


Abbildung 6.2.: Messwerte nach Widerstandsänderungen

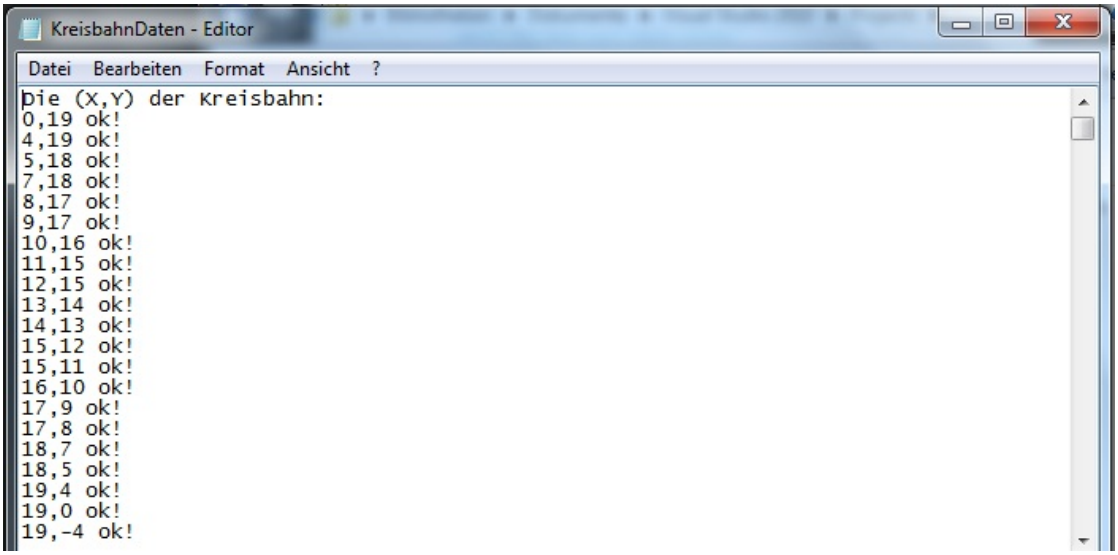


Abbildung 6.3.: Die Koordinaten der ausgewählten Punkte auf der ersten Viertelkreisbahn mit r = 0,095 mm (19 Schritte)

eingegebenen Radius ausgewählt wurden, und die Rückmeldungen, ob die Koordinaten dieser Punkte erfolgreich an die Schrittmotorsteuerkarte geschickt wurden, sind in der Datei `KreisbahnDaten.txt` gespeichert. Die Abbildung 6.3 zeigt einen Teil dieser Datei, welche die Koordinaten der zum Fräsen einer Kreisbahn mit $r = 0,095mm$ ausgewählten Punkte und die Rückmeldungen enthält. Für diesen Test wurde die Schrittauflösung des Schrittmotors auf $0,005mm/Schritt$ eingestellt, so dass der Radius umgerechnet 19 Schritte beträgt. Die gefräste Kreisbahn soll genau wie die blaue Linie in der Abbildung 5.1 aussehen.

7. Zusammenfassung

Es wurden zunächst die dieser Arbeit zu Grunde liegenden Kenntnisse vorgestellt. Dies sind Eigenspannungen und deren Messverfahren Bohrlochverfahren, Dynamic Link Library (DLL), ActiveX-Steuerelement, .Net Framework sowie C++/CLI-Schnittstelle. Das Messgerät Vishay Modell P3 und die USB-Steuerkarte USBMotion3xII wurden in Kapitel 3 vorgestellt.

In Kapitel 5 (Entwurf) wurden zunächst ein Konzept entwickelt, das erkennen kann, ob nach jeder Vertiefung der Kreisbahn die Messwerte von den eingesetzten Messkanälen stabil sind. Anschließend wurde der Bresenham Algorithmus, der für die Bestimmung der Punkte auf der Kreisbahn zum Einsatz kam, vorgestellt.

Die implementierten Klassen und deren Methoden wurden in Kapitel 6 (Implementierung) beschrieben. Im Gegensatz zum ActiveX-Steuerelement VMMP3Control können die Funktionen von der DLL USBM3x32 in einer verwalteten Klasse nicht direkt aufgerufen werden. Daher wurde auch eine Wrapper-Klasse für diese DLL implementiert. Die Benutzerschnittstelle und die Testergebnisse des Anwendungsprogrammes wurden auch in diesem Kapitel vorgestellt.

Die Testergebnisse haben gezeigt, dass das Programm die Messwerte schon für wieder stabil hielt, während sie sich noch langsam änderten. Das Problem kann dadurch gelöst werden, dass die Anzahl der Messwerte, die miteinander verglichen werden sollen, erhöht wird.

Es ist erforderlich, dass die maximale Geschwindigkeit und die maximale Beschleunigung von Schrittmotoren auch vom Benutzer selbst festgelegt werden können. Dadurch ist der Benutzer in der Lage, den Fräsprozess zu beschleunigen.

A. Eigenschaften und Methoden von VMMP3Control.dll

Im Folgenden werden die vom ActiveX-Steuerelement VMMP3Control.dll zur Verfügung gestellten Eigenschaften und Methoden beschrieben, wobei das Schlüsselwort `ByVar` festlegt, dass beim Aufruf dem Parameter der Wert eines Arguments übergeben wird. Eine variable vom Typ `Variant` kann numerische Daten, Zeichenfolgen, Datumsdaten sowie die speziellen Werte Empty und Null annehmen. Der Parameter Channel kann die Integerzahlen 1, 2, 3 und 4 annehmen.

1. Property ADReading(ByVal Channel As Long) As Long
   Schreibgeschützt,
   enthält die aktuellen A/D–Daten des gewählten Kanals

2. Property BalanceMode(ByVal Channel As Long) As Long
   enthält eine der folgenden Integerzahlen:
   0–Brückenabgleichfunktion für den gewählten Kanal ist abgeschaltet
   1–Brückenabgleich des gewählten Kanals wird vom Messgerät ausgeführt
   2–Brückenabgleich des gewählten Kanals wird vom Benutzer manuell ausgeführt

3. Property BalanceValue(ByVal Channel As Long) As Long
   enthält die Anzahl der A/D–Wandlung. Bem Brückenabgleich des gewählten Kanals führt das Messgerät soviel Messungen durch und nimmt davon den Mittelwert

4. Property BridgeType(ByVal Channel As Long) As Long
   enthält den Messbrücketyp(eine Integerzahl zwischen 0–10)des gewählten Kanals
   0 – Viertelbrücke
   1 – Halbbrücke, DMS in benachbarten Brückenzweigen
   2 – Halbbrücke, DMS in gegenüber liegenden Brückenzweigen
   3 – Halbbrücke, DMS in benachbarten Brückenzweigen, shear configuration
   4 – Halbbrücke, 1 DMS in Hauptrichtung und 1 DMS in Richtung Poisson–Dehnung
   5 – Vollbrücke, 4 aktive DMS
   6 – Vollbrücke, 4 aktive DMS, shear configuration
   7 – Vollbrücke, 2 Poisson–DMS in gegenüber liegenden Brückenzweigen
   8 – Vollbrücke, 2 Poisson–DMS in benachbarten Brückenzweigen
   9 – Undefinierte Vollbrücke
   10 – Undefinierte Halb– oder Viertelbrücke

5. Property ChannelActive(ByVal Channel As Long) As Boolean
   enthält den Status des gewählten Kanals,
   kann abgefragt und auf true oder false gesetzt werden.

6. Property ChannelLabel(ByVal Channel As Long) As String
   enthält den Namen des gewählten Kanals

7. Property CurrentReading(ByVal Channel As Long) As Variant
   Schreibgeschützt
   enthält den aktuellen Messwert des gewählten Kanals

8. Property DeviceOpen As Boolean
   enthält den Zustand des angeschlossenen Geräts, muss vor dem Zugriff auf andere Eigenschften auf true gesetzt werden

9. Property EngUnits(ByVal Channel As Long) As String
   enthält die Engineering−Einheiten des gewählten Kanals, Max. 4 Zeichen

10. Property FullScaleInEngUnits(ByVal Channel As Long) As Variant
    enthält den full−scale Wert des gewählten Kanals, in Engineering−Einheiten

11. roperty FullScaleInmVPerV(ByVal Channel As Long) As Variant
    enthält den full−scale Wert in mV/V des gewählten Kanals

12. Property GageFactor(ByVal Channel As Long) As Variant
    enthält den k−Faktor des gewählten Kanals

13. Property MediaCardOpen As Boolean
    enthält den Wert true, dann ist die Multimedia−Karte geöffnet

14. Property PoissonsRatio As Variant
    enthält den Poissons−Ratio Wert

15. Property RecordingActive As Boolean
    enthält den Wert true, dann ist die Aufnahme aktiviert, sonst deaktiviert

16. Property RecordingInterval(ByVal Channel As Long) As Long
    enthält das Aufnahme−Intervall des gewählten Kanals, in Sekunden

17. Property RecordingMode As Long
    enthält eine der folgenden Integerzahlen:
    0 − die Messdaten nicht auf der Multimedia−Karte speichern
    1 − die Messdaten manuell auf der Multimedia−Karte speichern
    2 − die Messdaten automatisch auf der Multimedia−Karte speichern

18. Property ScansRecorded As Long
    Schreibgeschützt,
    enthält die Anzahl der aufgenommenen Scans

19. Property SerialNo As String
    Schreibgeschützt,
    enthält die Seriennummer des Messgeräts P3, im ASCII−Format.

20. Property ShuntCalEnabled As Boolean
    enthält den Wert true, dann ist die Shunt−Kalibrierung aktiviert,
    sonst deaktiviert

21. Property ShuntCalValue(ByVal Channel As Long) As Long
    enthält die Messungsanzahl für die Shunt−Kalibrierung des gewählten Kanals

22. Property VersionNumber As Variant
    Schreibgeschützt,
    enthält die Versionsnummer der Firmware

23. Function CloseMMCDataFile() As Boolean
    schließt eine zuvor geöffnete Datei auf der Multimedia−Karte.

24. Function EraseMMC() As Boolean

löscht die Dateien auf der Multimedia-Karte

25. Sub GetTime(year As Integer, month As Integer, day As Integer,
hour As Integer, minute As Integer, second As Integer)
gibt Datum und Uhrzeit aus der Echtzeituhr vom Messgerät P3 zurück

26. Function OpenMMCDataFile(ByVal Value As String) As Boolean
öffnet die ausgewählte Datei auf der Multimedia-Karte

27. Function RecordOneScan() As Boolean
zeichnet einen Scan der vier Kanäle auf der Multimedia-Karte auf,
gibt true zurück, falls es erfolgreich ist

28. Sub ResetFactoryDefaults(ByVal SaveToFlash As Boolean)
setzt das Messgerät P3 auf die Werkseinstellungen zurück und speichert diese
Einstellungen auf dem internen Flash-Speicher, falls SaveToFlash true ist

29. Sub SaveSetup()
Speichert das aktuelle P3-Setup auf dem internen Flash-Speicher

30. Sub SetTime(ByVal year As Integer, ByVal month As Integer,
ByVal day As Integer, ByVal hour As Integer,
ByVal minute As Integer, ByVal second As Integer)
legt das Datum und die Uhrzeit für das Messgerät P3 fest

31. Function StartFirmwareUpgrade() As Boolean
startet den Firmware-Upgrade-Prozess

B. Funktionen von USBM3X32.DLL und Testprogramm

B.1. USBM3X32.DLL-Funktionen

Die für diese Arbeit relevante Funktionen sind im folgenden aufgelistet. Die Auflistung aller von der USBMotion3XII-Karte zur Verfügung gestellten Funktionen und aller möglichen Rückgabewerte dieser Funktionen befindet sich in [2].

1. `DWORD USBMCCreate(HWND hWnd, unsigned char EventsWait, PCHAR IDString);`

   `hWnd`: das Handle der aufrufenden Anwendung.

   `EventsWait`: 0 oder 1

   `EventsWait=0`: Für Lesen/Schreiben-Funktionen soll die Anwendung auf die Daten aus USB-Karte warten

   `EventsWait=1`: Windows Messages sollen in der Anwendung zum Einsatz kommen

   `IDString`: die vom Benutzer definierte Windows message ID

   Aufrufkonvention: stdcall

   Beschreibung: Der Rückgabewert ist null, falls der Aufruf dieser Funktion fehlschlägt. Sonst ist der Rückgabewert eine Message-ID im Bereich von 0xC000 bis 0xFFFF. Beim Start der Anwendung muss diese Funktion aufgerufen werden, damit die Objekte und Klassen in der DLL erstellt werden.

2. `unsigned char USBMCDestroy();`

   Aufrufkonvention: stdcall

   Beschreibung: Beim Beenden der Anwendung muss diese Funktion aufgerufen werden, damit die Ressourcen wieder freigegeben werden.

3. `unsigned char USBMCinit(unsigned long& PData, unsigned long& PIICData, unsigned long& PIICSCN);`

   `PData`: ein zeiger, der auf die Daten zeigt, die vom Schrittmotor-Controller gelesen wurden.

   `PIICData`: ein zeiger, der auf die Daten zeigt, die von den I2C-Geräten gelesen wurden.

`PIICSCN`: ein zeiger, der auf ein Array zeigt, das die Informationen über alle verfügbaren I2C-Geräte auf dem Bus enthält.

Aufrufkonvention: stdcall

Beschreibung: Nachdem die Funktion `USBMCCreate` erfolgreich ausgeführt wurde, muss diese Funktion aufgerufen werden. Der Rückgabewert ist ungleich Null, falls der Aufruf dieser Funktion fehlschlägt.

4. `unsigned char SetXtarget(unsigned char Motor, long xtarg, unsigned char& Status);`

`Motor`: die Nummer des Schrittmotors (0, 1 oder 2)

`xtarg`: die Zielposition des Schrittmotors

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Beschreibung: Diese Funktion schreibt den Wert von xtarg ins Zielregister. Die Zielposition ist angegeben in Einheiten von Vollschritten bzw. Mikroschritten. Die Einheit ist abhängig von der Einstellung der Mikroschrittauflösung. Diese Funktion kann jede Zeit aufgerufen werden, auch während die Zielposition angefahren wird.

5. `unsigned char SetXYZtarget(unsigned char Motor, long xtarg, long ytarg, long ztarg, unsigned char& Status);`

`Motor`: eine Nummer zwischen 0 bis 7

`xtarg, ytarg, ztarg`: die Zielpositionen der Schrittmotoren 0, 1, 2

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Der Wert von `Motor` wird bestimmt durch die Umrechnung eine 3-Bit-Binärzahl in eine Dezimalzahl:
Bit0 = 1/0 -> Die neue Zielposition des Schrittmotors 0 wird angenommen/nicht angenommen.
Bit1 = 1/0 -> Die neue Zielposition des Schrittmotors 1 wird angenommen/nicht angenommen.
Bit2 = 1/0 -> Die neue Zielposition des Schrittmotors 2 wird angenommen/nicht angenommen.

6. `unsigned char SetVmax(unsigned char Motor, unsigned short Vmax, unsigned char& Status);`

`Motor`: die Nummer des Schrittmotors (0, 1 oder 2)

`Vmax`: die maximale Geschwindigkeit (Schritte/Sekunde)

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Beschreibung: Diese Funktion legt die maximale Motorgeschwindigkeit fest. Der absolute Wert der Geschwindigkeit wird diese Grenze nicht überschreiten, außer wenn der Wert von Vmax während der Bewegung auf einen Wert unterhalb der aktuellen Geschwindigkeit gesetzt wird.

7. `unsigned char SetAmax(unsigned char Motor, unsigned short Vmax, unsigned char& Status);`

`Motor`: die Nummer des Schrittmotors (0, 1 oder 2)

`Amax`: die maximale Beschleunigung (Schritte/Sekunde/Sekunde)

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Beschreibung: Diese Funktion legt die maximale Beschleunigung des Schrittmotors fest. `Amax` während der Bewegung des Schrittmotors auf Null zu setzen, wird dazu führen, dass der Schrittmotor nicht gestoppt wird, da seine Geschwindigkeit nicht mehr verändert werden kann.

8. `unsigned char GetXtarget(unsigned char Motor, unsigned char& Status, long& xtarg);`

Beschreibung: Diese Funktion gibt die Zielposition des Schrittmotors zurück.

9. `unsigned char GetVmax(unsigned char Motor, unsigned char& Status, unsigned short& Vmax);`

Beschreibung: Diese Funktion gibt die maximale Geschwindigkeit des Schrittmotors zurück.

10. `unsigned char GetAmax(unsigned char Motor, unsigned char& Status, unsigned short& Amax);`

Beschreibung: Diese Funktion gibt die maximale Beschleunigung des Schrittmotors zurück.

11. `unsigned char SetMode(unsigned char Motor, unsigned char rMode, unsigned char& Status);`

`Motor`: die Nummer des Schrittmotors (0, 1 oder 2)

`rMode`: die Nummer der Betriebsarten RAMPMODE (0x00), SOFTMODE (0x01), VELOCITYMODE (0x02) und HOLDMODE (0x03)

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Beschreibung: Die Betriebsart RAMPMODE ist für Positionierungsaufgaben vorgesehen. Der Schrittmotorcontroller wird aufgrund der aktuellen Zielposition ein trapezförmiges Geschwindigkeitsprofil berechnen und dann den Schrittmotor dementsprechend steuern.
Die Betriebsart SOFTMODE ist der RAMPMODE ähnlich, außer dass die Zielposition mit exponentiell reduzierter Geschwindigkeit angefahren wird.
Die Betriebsart VELOCITYMODE ist für die Anwendungen geeignet, bei denen Schrittmotoren mit konstanter Geschwindigkeit laufen müssen. Die Positionierung muss dabei nicht speziell betrachtet werden.
In der Betriebsart HOLDMODE wird die Zielgeschwindigkeit direkt vom Benutzer festgelegt. Der Schrittmotorcontroller ignoriert die Geschwindigkeits- und Beschleunigungsbegrenzung, um diese zu erreichen.

12. `unsigned char SetMicroSteps(unsigned char Motor, unsigned char mStep, unsigned char& Status);`

`Motor`: die Nummer des Schrittmotors (0, 1 oder 2)

`mStep`: eine Nummer zwischen 0 und 6

`Status`: gibt den Status aller Schrittmotoren zurück.

Aufrufkonvention: stdcall

Beschreibung: Mit Hilfe dieser Funktion wird die Schrittauflösung des Schrittmotors festgelegt. Hat z.B. ein Schrittmotor n Vollschritte pro Umdrehung und eine Spindelsteigung von 1 mm, so wird die Schrittauflösung durch $\frac{1}{n \cdot 2^{mStep}}$ berechnet. Die Schrittauflösung ist $\frac{1}{n}$ mm/Vollschritt, falls mStep die Nummer 0 ist. Die Schrittauflösung ist $\frac{1}{64 \cdot n}$ mm/Mikroschritt, falls mStep die Nummer 6 ist.

13. `unsigned char GetMode(unsigned char Motor, unsigned char& rMode, unsigned char& Status);`

`rMode`: gibt die eingestellte Betriebsart zurück.

14. `unsigned char GetMicroSteps(unsigned char Motor, unsigned char& mStep, unsigned char& Status);`

`mStep`: gibt die eingestellte Mikroschrittauflösung zurück.

Die Rückgabewerte dieser Funktionen sind u.a. folgende:

`SUCCESS (0x00)`: Eine Lesen/Schreiben-Funktion wurde erfolgreich ausgeführt. Eine Funktion gibt diesen Wert zurück, falls der Parameter `EventsWait` von der Funktion `USBMCCreate` beim Aufruf auf 0 gesetzt wurde.

`TX_SUCCESS (0x01)`: Eine Lesen/Schreiben-Funktion wurde erfolgreich an ein USB-Gerät geschickt. Eine Funktion gibt diesen Wert zurück, falls der `EventsWait` von der Funktion `USBMCCreate` beim Aufruf auf 1 gesetzt wurde.

`DEVICE_BUSY (0x03)`: Das USB-Gerät ist noch nicht mit der vorher aufgerufenen Funktion fertig.

`DEV_NOT_ASSIGNED (0x05)`: Eine Funktion gibt diesen Wert zurück, falls keine USB-Geräte verfügbar sind oder kein USB-Gerät ausgewählt wurde.

B.2. Testprogramm

Das folgende Testprogramm soll zeigen, wie man diese DLL in einer Visual C++ Console-Anwendung einsetzen kann. Bei dieser DLL handelt sich um eine Win32 API-Anwendung.

```
#include "stdafx.h"
#include <windows.h>

using namespace System;

// Definition des Types der DLL-Funktionen, die verwendet werden sollen
typedef DWORD (__stdcall *LPUSBMCCREATE) (HWND, unsigned char, String^);
typedef unsigned char (__stdcall *LPUSBMCDESTROY) ();
typedef unsigned char (__stdcall *LPUSBMCINIT)(unsigned long&, unsigned long&,
                                               unsigned long&);
typedef unsigned char (__stdcall *LPSETXTARGET) (unsigned char,
                                                 long, unsigned char&);
typedef unsigned char (__stdcall *LPGETXTARGET) (unsigned char,
                                                 unsigned char&, long&);
typedef unsigned char (__stdcall *LPSETVMAX) (unsigned char, unsigned short,
                                              unsigned char&);

int main(array<System::String^ > ^args)
{
  HINSTANCE husbm3x32Dll;
  LPUSBMCCREATE           lpUSBMCCreate;
  LPUSBMCDESTROY          lpUSBMCDestroy;
  LPUSBMCINIT             lpUSBMCInit;
  LPSETXTARGET            lpSetXtarget;
```

```
LPGETXTARGET            lpGetXtarget;
LPSETVMAX               lpSetVmax;

String ^szIDString = "MCControl_MyApp_MsgID";
unsigned int    uiPrivateMsg;
unsigned long   PData;
unsigned long   PIICData;
unsigned long   PIICScan;

// DLL Datei wird geladen
husbm3x32Dll = (HINSTANCE)LoadLibrary(L"USBM3x32.dll");
if (husbm3x32Dll) Console::WriteLine(L"DLL wurde erfolgreich geladen");

// Die Einsprungadressen werden abgefragt
lpUSBMCCreate =(LPUSBMCCREATE)GetProcAddress(husbm3x32Dll, "USBMCCreate");
lpUSBMCDestroy =(LPUSBMCDESTROY)GetProcAddress(husbm3x32Dll,"USBMCDestroy");
lpUSBMCInit = (LPUSBMCINIT)GetProcAddress(husbm3x32Dll, "USBMCinit");
lpSetXtarget = (LPSETXTARGET)GetProcAddress(husbm3x32Dll, "SetXtarget");
lpGetXtarget = (LPGETXTARGET)GetProcAddress(husbm3x32Dll, "GetXtarget");
lpSetVmax = (LPSETVMAX)GetProcAddress(husbm3x32Dll, "SetVmax");

// Es soll geprüft werden, ob jeweilige Einsprungadresse nicht Null ist, z.B.
// if(lpUSBMCCreate) Console::WriteLine("lpUSBMCCreate-Einsprungadresse ok");

// Die Objekte und Klassen in der DLL werden erstellt und initialisiert
// Der Rückgabewert ist null, falls der USBMCCreate-Aufruf fehlschlägt
// Der Rückgabewert ist null, falls der USBMCInit-Aufruf erfolgreich ist
uiPrivateMsg = lpUSBMCCreate(0,0,szIDString);
if (uiPrivateMsg > 0)
{
  if (lpUSBMCInit(PData,PIICData,PIICScan) == 0)
  {
    Console::WriteLine(L"DLL wurde erfolgreich initialisiert!");
      } else Console::WriteLine(L"DLL kann nict initialisiert werden!");
  } else Console::WriteLine(L"DLL kann nict initialisiert werden!");

long lXtarget;
unsigned short usVmax;
unsigned char ucstatus;
unsigned char ucfeedback;

Console::Write(L"Die Zielposition von Motor 0 eingeben: ");
lXtarget = Convert::ToInt64(Console::ReadLine());
ucfeedback = lpSetXtarget(0,lXtarget,ucstatus);

// ucfeedback = 0, falls der SetXtarget-Aufruf für Motor 0 erfolgreich ist
switch (ucfeedback)
{
    case 0: Console::WriteLine(L"SUCCESS");
              break;
    case 1: Console::WriteLine(L"TRANSMISSION SUCCESSFULL");
              break;
    case 3: Console::WriteLine("Device is Busy");
              break;
    case 5: Console::WriteLine("ERROR: USB Device not assigned");
              break;
    default: Console::WriteLine(L"??");
}
```

```
    ucfeedback = lpGetXtarget(0,ucstatus,lXtarget);
    if (ucfeedback == 0)
    {
        Console::WriteLine("Die ZielPosition von Motor 0: " + lXtarget);
    }else Console::WriteLine("GetXtarget-feedback: " + ucfeedback);

    Console::Write(L"Die max.Geschwindigkeit von Motor 0 eingeben: ");
    usVmax = Convert::ToInt32(Console::ReadLine());
    ucfeedback = lpSetVmax(0,usVmax,ucstatus);
    Console::WriteLine("SetVmax-feedback: " + ucfeedback);

    // Die Ressourcen werden wieder freigegeben
    lpUSBMCDestroy();
    // Die DLL-Datei wieder entladen
    FreeLibrary((HMODULE)husbm3x32Dll);

    Console::ReadLine();
    return 0;
}
```

C. Quellext vom Anwendungsprogramm Messdateneinlesen und Schrittmotorsteuern

C.1. USBMotion3x.h

```
#pragma once
#include <windows.h>

#pragma unmanaged
  // Definition des Types der DLL-Funktion, die verwendet werden soll
  typedef DWORD (__stdcall *LPUSBMCCREATE) (HWND, unsigned char, char*);
  typedef unsigned char (__stdcall *LPUSBMCDESTROY) ();
  typedef unsigned char (__stdcall *LPUSBMCINIT)(unsigned long&, unsigned long&,
                                                 unsigned long&);
  typedef unsigned char (__stdcall *LPDEVICECOUNT)();
  typedef unsigned char (__stdcall *LPSETXTARGET) (unsigned char, long,
                                                   unsigned char&);
  typedef unsigned char (__stdcall *LPGETXTARGET) (unsigned char,
                                                   unsigned char&, long&);
  typedef unsigned char (__stdcall *LPSETVMAX) (unsigned char, unsigned short,
                                                unsigned char&);
  typedef unsigned char (__stdcall *LPSETXYZTARGET) (unsigned char, long, long,
                                                     long, unsigned char&);

// Die Klasse USBMotion3x fasst die Methoden zur Steuerung
// des Schrittmotors zusammen
class USBMotion3x
{
  HINSTANCE husbm3x32Dll;
  LPUSBMCCREATE           lpUSBMCCreate;
  LPUSBMCINIT             lpUSBMCInit;
  LPUSBMCDESTROY          lpUSBMCDestroy;
  LPSETXTARGET            lpSetXtarget;
  LPSETVMAX               lpSetVmax;
  LPSETXYZTARGET          lpSetXYZtarget;
  LPGETXTARGET            lpGetXtarget;

  unsigned char ucstatus;
  unsigned char ucfeedback;
  unsigned int  uiPrivateMsg;
  unsigned long PData;
  unsigned long PIICData;
  unsigned long PIICScan;

public:
  USBMotion3x(void);
  ~USBMotion3x(void);
```

```
  int SchrittmotorDLLInit();
  int SchrittmotorDLLEntladen();
  int SetXtarget(unsigned char motorIndex, long lXtarget);
  int SetXYZtarget(unsigned char motorIndex, long lXtarget, long lYtarget,
                   long lZtarget);
  long GetXtarget(unsigned char motorIndex);
  int SetVmax(unsigned char motorIndex, unsigned short usVmax);
};

#pragma managed
// Die Klasse ManUSBMotion3x ist die Wrapper-Klasse der nicht
// verwalteten Klasse USBMotion3x
public ref class ManUSBMotion3x
{
  USBMotion3x* UnUSBMotion3x;

public:
  ManUSBMotion3x(void);
protected:
  ~ManUSBMotion3x(void);
public:
  int managed_SchrittmotorDLLInit();
  int managed_SchrittmotorDLLEntladen();
  int managed_SetXtarget(unsigned char motorIndex, long lXtarget);
  int managed_SetXYZtarget(unsigned char motorIndex, long lXtarget,
                           long lYtarget, long lZtarget);
  long managed_GetXtarget(unsigned char motorIndex);
  int managed_SetVmax(unsigned char motorIndex, unsigned short usVmax);
};
```

C.2. USBMotion3x.cpp

```
#include "StdAfx.h"
#include "USBMotion3x.h"

USBMotion3x::USBMotion3x(void)
{
}

USBMotion3x::~USBMotion3x(void)
{
}

int USBMotion3x::SchrittmotorDLLInit()
{
  // DLL Datei laden
  husbm3x32Dll = (HINSTANCE)LoadLibrary(L"USBM3x32.dll");
  // Die Einsprungadresse abfragen
  lpUSBMCCreate =(LPUSBMCCREATE)GetProcAddress(husbm3x32Dll, "USBMCCreate");
  lpUSBMCInit =(LPUSBMCINIT)GetProcAddress(husbm3x32Dll, "USBMCinit");
  lpUSBMCDestroy =(LPUSBMCDESTROY)GetProcAddress(husbm3x32Dll, "USBMCDestroy");
  lpSetXtarget =(LPSETXTARGET)GetProcAddress(husbm3x32Dll, "SetXtarget");
  lpSetVmax =(LPSETVMAX)GetProcAddress(husbm3x32Dll, "SetVmax");
  lpSetXYZtarget =(LPSETXYZTARGET)GetProcAddress(husbm3x32Dll, "SetXYZtarget");
  lpGetXtarget =(LPGETXTARGET)GetProcAddress(husbm3x32Dll, "GetXtarget");
```

```
  uiPrivateMsg = lpUSBMCCreate(0,0,"MCControl_MyApp_MsgID");
  if (uiPrivateMsg > 0)
  {
    if (lpUSBMCInit(PData,PIICData,PIICScan) == 0)
    {
      return 300;                          // "DLL-Init erfolgreich!" ;
    } else return 200;                     // "can not initialize DLL-Init?" ;
  } else return 100;                       // "can not initialize DLL-Curn reate";
}

int USBMotion3x::SchrittmotorDLLEntladen()
{
  lpUSBMCDestroy();
  // Die DLL-Datei wieder entladen
  FreeLibrary((HMODULE)husbm3x32Dll);
  return 400;
}

int USBMotion3x::SetXtarget(unsigned char motorIndex, long lXtarget)
{
  //(ucfeedback) 0: SUCCESS; 1: TRANSMISSION SUCCESSFULL;
  //             3: Device is Busy; 5: USB Device not assigned!
  return ucfeedback = lpSetXtarget(motorIndex, lXtarget, ucstatus);
}

int USBMotion3x::SetXYZtarget(unsigned char motorIndex, long lXtarget,
                              long lYtarget, long lZtarget)
{
  return ucfeedback = lpSetXYZtarget(motorIndex, lXtarget, lYtarget,
                                     lZtarget, ucstatus);
}

long USBMotion3x::GetXtarget(unsigned char motorIndex)
{
  long lXtarget;

  ucfeedback = lpGetXtarget(motorIndex,ucstatus,lXtarget);
  if (ucfeedback == 0)
  {
    return lXtarget;
  } else return LONG_MAX;
}

int USBMotion3x::SetVmax(unsigned char motorIndex, unsigned short usVmax)
{
  return ucfeedback = lpSetVmax(motorIndex, usVmax, ucstatus);
}


ManUSBMotion3x::ManUSBMotion3x(void)
{
  UnUSBMotion3x = new USBMotion3x();
}

ManUSBMotion3x::~ManUSBMotion3x(void)
{
  delete UnUSBMotion3x;
```

```
  UnUSBMotion3x = NULL;
}

int ManUSBMotion3x::managed_SchrittmotorDLLInit()
{
  return UnUSBMotion3x->SchrittmotorDLLInit();
}

int ManUSBMotion3x::managed_SchrittmotorDLLEntladen()
{
  return UnUSBMotion3x->SchrittmotorDLLEntladen();
}

int ManUSBMotion3x::managed_SetXtarget(unsigned char motorIndex, long lXtarget)
{
  return UnUSBMotion3x->SetXtarget(motorIndex, lXtarget);
}

int ManUSBMotion3x::managed_SetXYZtarget(unsigned char motorIndex, long lXtarget,
                                         long lYtarget, long lZtarget)
{
  return UnUSBMotion3x->SetXYZtarget(motorIndex, lXtarget, lYtarget, lZtarget);
}

long ManUSBMotion3x::managed_GetXtarget(unsigned char motorIndex)
{
  return UnUSBMotion3x->GetXtarget(motorIndex);
}

int ManUSBMotion3x::managed_SetVmax(unsigned char motorIndex,
                                    unsigned short usVmax)
{
  return UnUSBMotion3x->SetVmax(motorIndex, usVmax);
}
```

C.3. Form1.h

```
#pragma once
#include "USBMotion3x.h"

namespace VP3undUSBMotion3x {

  using namespace System;
  using namespace System::ComponentModel;
  using namespace System::Collections;
  using namespace System::Windows::Forms;
  using namespace System::Data;
  using namespace System::Drawing;
  using namespace System::IO;
  using namespace System::Collections::Generic;
  using namespace System::Threading;

// Zusammenfassung für Form1
public ref class Form1 : public System::Windows::Forms::Form
{
  VMMP3Control::VMMP3Controller ^NewVP3;
```

```
  int aktAnzahl;
  StreamWriter ^datei1, ^datei2;
  array<List<double>^> ^aVierKanal;
  bool bWiederMalUnverändern;

  ManUSBMotion3x ^NewMotion3x;
  array<int> ^xyzLengthAlt;
  array<List<int>^> ^aKreisbahnDaten;
  ThreadStart ^tsKreisbahn;
  Thread ^threadKreisbahn;

public:
  Form1(void)
  {
    InitializeComponent();
    dateienAnlegen();

    this->NewVP3 = gcnew VMMP3Control::VMMP3Controller();
    this->aktAnzahl = 0;
    this->aVierKanal = gcnew array<List<double>^>(4);
    bWiederMalUnverändern = false;

    NewMotion3x = gcnew ManUSBMotion3x();
    this->xyzLengthAlt = gcnew array<int>(4){0,0,0,0};
    this->aKreisbahnDaten = gcnew array<List<int>^>(2){gcnew List<int>(),
                                                        gcnew List<int>()};
    this->tsKreisbahn = gcnew ThreadStart(this, &Form1::kreisbahnFräsen);
    SchrittmotorSteuerkarteInit();
  }

protected:
  // Verwendete Ressourcen bereinigen.
  ~Form1()
  {
    if (components)
    {
      delete components;
    }
    NewMotion3x->managed_SchrittmotorDLLEntladen();
  }

protected:
private: System::Windows::Forms::Button^  button1;
private: System::Windows::Forms::Button^  button2;
private: System::Windows::Forms::Button^  button3;

private: System::Windows::Forms::Label^  label1;
private: System::Windows::Forms::Label^  label2;
private: System::Windows::Forms::Label^  label3;
private: System::Windows::Forms::Label^  label4;
private: System::Windows::Forms::Label^  label5;
private: System::Windows::Forms::Label^  label7;

private: System::Windows::Forms::TextBox^  textBox1;
private: System::Windows::Forms::TextBox^  textBox2;
private: System::Windows::Forms::TextBox^  textBox3;
private: System::Windows::Forms::TextBox^  textBox4;
private: System::Windows::Forms::TextBox^  textBox6;
```

```
private: System::ComponentModel::IContainer^  components;
private: System::Windows::Forms::Timer^  timer1;

#pragma region Windows Form Designer generated code

    // Erforderliche Methode für die Designerunterstützung.
    // Der Inhalt der Methode darf nicht mit dem Code-Editor geändert werden.

  void InitializeComponent(void)
  {
    this->components = (gcnew System::ComponentModel::Container());
    this->button1 = (gcnew System::Windows::Forms::Button());
    this->button2 = (gcnew System::Windows::Forms::Button());
    this->button3 = (gcnew System::Windows::Forms::Button());
    this->textBox1 = (gcnew System::Windows::Forms::TextBox());
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->textBox2 = (gcnew System::Windows::Forms::TextBox());
    this->label2 = (gcnew System::Windows::Forms::Label());
    this->label3 = (gcnew System::Windows::Forms::Label());
    this->textBox3 = (gcnew System::Windows::Forms::TextBox());
    this->label4 = (gcnew System::Windows::Forms::Label());
    this->label5 = (gcnew System::Windows::Forms::Label());
    this->textBox4 = (gcnew System::Windows::Forms::TextBox());
    this->textBox6 = (gcnew System::Windows::Forms::TextBox());
    this->label7 = (gcnew System::Windows::Forms::Label());
    this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));
    this->SuspendLayout();

    // button1

    this->button1->Location = System::Drawing::Point(44, 57);
    this->button1->Name = L"button1";
    this->button1->Size = System::Drawing::Size(75, 23);
    this->button1->TabIndex = 0;
    this->button1->Text = L"Starten";
    this->button1->UseVisualStyleBackColor = true;
    this->button1->Click += gcnew System::EventHandler(this,
                                                       &Form1::button1_Click);

    // button2

    this->button2->Location = System::Drawing::Point(44, 96);
    this->button2->Name = L"button2";
    this->button2->Size = System::Drawing::Size(75, 23);
    this->button2->TabIndex = 1;
    this->button2->Text = L"Beenden";
    this->button2->UseVisualStyleBackColor = true;
    this->button2->Click += gcnew System::EventHandler(this,
                                                       &Form1::button2_Click);

    // button3

    this->button3->Location = System::Drawing::Point(44, 193);
    this->button3->Name = L"button3";
    this->button3->Size = System::Drawing::Size(75, 23);
    this->button3->TabIndex = 2;
    this->button3->Text = L"Homing";
```

```
this->button3->UseVisualStyleBackColor = true;
this->button3->Click += gcnew System::EventHandler(this,
                                                   &Form1::button3_Click);

// textBox1

this->textBox1->Anchor = System::Windows::Forms::AnchorStyles::None;
this->textBox1->BackColor = System::Drawing::SystemColors::ButtonHighlight;
this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft YaHei", 12,
                        System::Drawing::FontStyle::Regular,
                        System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
this->textBox1->Location = System::Drawing::Point(201, 0);
this->textBox1->Multiline = true;
this->textBox1->Name = L"textBox1";
this->textBox1->ReadOnly = true;
this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
this->textBox1->Size = System::Drawing::Size(685, 417);
this->textBox1->TabIndex = 3;

// label1

this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(24, 233);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(42, 13);
this->label1->TabIndex = 7;
this->label1->Text = L"X (mm):";

// textBox2

this->textBox2->Location = System::Drawing::Point(27, 249);
this->textBox2->Multiline = true;
this->textBox2->Name = L"textBox2";
this->textBox2->Size = System::Drawing::Size(100, 21);
this->textBox2->TabIndex = 8;
this->textBox2->KeyPress +=
                gcnew System::Windows::Forms::KeyPressEventHandler
                (this, &Form1::textBox2_KeyPress);

// label2

this->label2->AutoSize = true;
this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif",
                      9, System::Drawing::FontStyle::Regular,
                      System::Drawing::GraphicsUnit::Point,
                      static_cast<System::Byte>(0)));
this->label2->Location = System::Drawing::Point(23, 29);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(119, 15);
this->label2->TabIndex = 9;
this->label2->Text = L"Messdaten Einlesen";

// label3

this->label3->AutoSize = true;
this->label3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif",
                      9, System::Drawing::FontStyle::Regular,
```

```
                        System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(0)));
this->label3->Location = System::Drawing::Point(24, 161);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(128, 15);
this->label3->TabIndex = 10;
this->label3->Text = L"Schrittmotorsteuerung";

// textBox3

this->textBox3->Location = System::Drawing::Point(27, 289);
this->textBox3->Multiline = true;
this->textBox3->Name = L"textBox3";
this->textBox3->Size = System::Drawing::Size(100, 21);
this->textBox3->TabIndex = 11;
this->textBox3->KeyPress +=
                gcnew System::Windows::Forms::KeyPressEventHandler
                (this, &Form1::textBox3_KeyPress);

// label4

this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(23, 273);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(42, 13);
this->label4->TabIndex = 14;
this->label4->Text = L"Y (mm):";

// label5

this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(24, 313);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(42, 13);
this->label5->TabIndex = 15;
this->label5->Text = L"Z (mm):";

// textBox4

this->textBox4->Location = System::Drawing::Point(27, 329);
this->textBox4->Multiline = true;
this->textBox4->Name = L"textBox4";
this->textBox4->Size = System::Drawing::Size(100, 21);
this->textBox4->TabIndex = 17;
this->textBox4->KeyPress +=
                gcnew System::Windows::Forms::KeyPressEventHandler
                (this, &Form1::textBox4_KeyPress);

// textBox6

this->textBox6->Location = System::Drawing::Point(27, 369);
this->textBox6->Multiline = true;
this->textBox6->Name = L"textBox6";
this->textBox6->Size = System::Drawing::Size(100, 21);
this->textBox6->TabIndex = 18;
this->textBox6->KeyPress +=
                gcnew System::Windows::Forms::KeyPressEventHandler
                (this, &Form1::textBox6_KeyPress);
```

```
    // label7

    this->label7->AutoSize = true;
    this->label7->Location = System::Drawing::Point(24, 353);
    this->label7->Name = L"label7";
    this->label7->Size = System::Drawing::Size(136, 13);
    this->label7->TabIndex = 19;
    this->label7->Text = L"Radius der Kreisbahn (mm):";

    // timer1

    this->timer1->Enabled = true;
    this->timer1->Interval = 1000;
    this->timer1->Tick += gcnew System::EventHandler(this,
                                                     &Form1::timer1_Tick);

    // Form1

    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
    this->ClientSize = System::Drawing::Size(884, 417);
    this->Controls->Add(this->label7);
    this->Controls->Add(this->textBox6);
    this->Controls->Add(this->textBox4);
    this->Controls->Add(this->label5);
    this->Controls->Add(this->label4);
    this->Controls->Add(this->textBox3);
    this->Controls->Add(this->label3);
    this->Controls->Add(this->label2);
    this->Controls->Add(this->textBox2);
    this->Controls->Add(this->label1);
    this->Controls->Add(this->textBox1);
    this->Controls->Add(this->button3);
    this->Controls->Add(this->button2);
    this->Controls->Add(this->button1);
    this->Name = L"Form1";
    this->Text = L"Messdateneinlesen und Schrittmotorsteuern";
    this->ResumeLayout(false);
    this->PerformLayout();

  }
#pragma endregion
  // zwei Dateien zur Speicherung der Messwerte und der Koordinaten der
  // Punkte auf der Kreisbahn anlegen
  private: void dateienAnlegen()
  {
    this->datei1 = gcnew StreamWriter("MessDaten.csv");
    this->datei1->WriteLine(" Nr.;Kanal1;Kanal2;Kanal3;Kanal4;");
    this->datei1->Close();
    this->datei2 = gcnew StreamWriter("KreisbahnDaten.txt");
    this->datei2->WriteLine("Die (X,Y) der Kreisbahn:");
    this->datei2->Close();
  }

  // Messwerte von allen Kanälen einlesen und mit existierten vergleichen.
  // jeweils einen Messwert von jedem Kanal in der Datei MessDaten.csv
  // speichern, wenn sich alle vorhandenen Messwerte nicht mehr ändern
```

```
private: void messvorgang()
{
  double dDifferenz = 2.0;
  bool bStatusUnverändern = true;
  for (int j=0; j<3; j++)
  {
    aVierKanal[j]->RemoveAt(0);
    aVierKanal[j]->Add(Convert::ToDouble(this->NewVP3->CurrentReading[j+1]));
    bStatusUnverändern = bStatusUnverändern
      && (Math::Abs(aVierKanal[j][0]- aVierKanal[j][1]) <= dDifferenz)
      && (Math::Abs(aVierKanal[j][0]- aVierKanal[j][2]) <= dDifferenz)
      && (Math::Abs(aVierKanal[j][1]- aVierKanal[j][2]) <= dDifferenz);
  }
  if (bStatusUnverändern)
  {
    if (this->bWiederMalUnverändern == false)
    {
      this->aktAnzahl ++;
      this->bWiederMalUnverändern = true;
      this->textBox1->AppendText("  " + aktAnzahl + ".  " +
                                 " Kanal1: " + aVierKanal[0][0] +
                                 ";    Kanal2: " + aVierKanal[1][0] +
                                 ";    Kanal3: " + aVierKanal[2][0] +
                                 ";    Kanal4: " + aVierKanal[3][0] + "\n");

      // true: Daten an die Datei anfügen
      this->datei1 = gcnew StreamWriter("MessDaten.csv", true);
      this->datei1->WriteLine(aktAnzahl + ";" +
                              aVierKanal[0][0] + ";" + aVierKanal[1][0] + ";"
                              + aVierKanal[2][0] + ";" + aVierKanal[3][0]);
      this->datei1->Close();
    }
  } else this->bWiederMalUnverändern = false;
}

// die Verbindung zum Messgerät Vishay P3 herstellen
private: System::Void button1_Click(System::Object^ sender,
                                    System::EventArgs^ e)
{
  this->NewVP3->DeviceOpen = true;
  for (int i=0; i<4; i++)
  {
    aVierKanal[i] = gcnew List<double>(3);
    aVierKanal[i]->Add(0.0); aVierKanal[i]->Add(0.0); aVierKanal[i]->Add(0.0);
  }
}

// die Verbindung zum Messgerät Vishay P3 löschen
private: System::Void button2_Click(System::Object^ sender,
                                    System::EventArgs^ e)
{
  this->NewVP3->DeviceOpen = false;
  for (int i=0; i<4; i++)
  {
    aVierKanal[i]->Clear();
  }
  this->textBox1->AppendText(L" Alle MessDaten wurden in der Datei
                             MessDaten.csv gespeichert! \n");
```

```
  this->button1->Focus();
}

// den Zustand des Hintergrundthreads threadKreisbahn abfragen und
// die Methode messvorgang einmal aufrufen
private: System::Void timer1_Tick(System::Object^ sender,
                                  System::EventArgs^ e)
{
  if ((aVierKanal[3] != nullptr) && (aVierKanal[3]->Count == 3))
    messvorgang();

  if (threadKreisbahn != nullptr)
  {
    if (threadKreisbahn->ThreadState.ToString() == "Stopped")
    {
      threadKreisbahn = nullptr;
      this->textBox1->AppendText(" Die Kreisbahndaten wurden erfolgreich
                                   an die Steuerkarte geschickt! \n");
    }
  }
}

// Schrittmotorsteuerkarte initialisieren
private: void SchrittmotorSteuerkarteInit()
{
  if (NewMotion3x->managed_SchrittmotorDLLInit() == 300)
  {
    this->textBox1->AppendText(" SchrittmotorSteuerkarte wurde
                                 erfolgreich initialisiert! \n");
  } else  this->textBox1->AppendText(" SchrittmotorSteuerkarte kann nicht
                                       initialisiert werden! \n");

  NewMotion3x->managed_SetVmax(0, 100);
  NewMotion3x->managed_SetVmax(1, 100);
  NewMotion3x->managed_SetVmax(2, 100);
}

// den Fräser an den Punkt (0, 0, 0) positionieren
private: System::Void button3_Click(System::Object^ sender,
                                    System::EventArgs^ e)    // Homing
{
  if (NewMotion3x->managed_SetXYZtarget(7, 0, 0, 0) != 0)
  {
    this->textBox1->AppendText(" USB Device not assigned!\n");
  } else this->textBox1->AppendText(" Motor X, Y und Z wurden
                                      zurückgesetzt! \n");
}

// motorIndex = 0: Motor X, 1: Motor Y, 2: Motor Z,
//              3: Durchmesser der Kreisbahn
// Eingaben von vier Eingabefeldern entgegennehmen und in
// Double-Werte konvertieren
private: long xyzDatenEinlesen(String^ eingabe, int motorIndex)
{
  System::Char motor = motorIndex + 88;
  int length = xyzLengthAlt[motorIndex];
  String^ ein = " " + eingabe;
  xyzLengthAlt[motorIndex] = ein->Length;
```

```
  long lSchritt = (Convert::ToDouble(ein->Substring(length + 1)))/0.005;
  if (motorIndex <= 2)
  {
    this->textBox1->AppendText(" Eingabe für Motor " + motor.ToString() + ": "
                               + ein->Substring(length + 1) +
                               " mm,  " + lSchritt + " Schritte \n");
  }
  else
  {
    this->textBox1->AppendText( " Eingabe für Radius: " +
                                ein->Substring(length + 1) + " mm,  "
                                + lSchritt + " Schritte \n");
  }
  return lSchritt;
}

// X-Koordinate an die Steuerkarte schicken
private: System::Void textBox2_KeyPress(System::Object^ sender,
                            System::Windows::Forms::KeyPressEventArgs^ e)
{
  if ( e->KeyChar == (char)13 )                  //the ENTER key is pressed
  {
    if (NewMotion3x->managed_SetXtarget(0,
                          xyzDatenEinlesen(this->textBox2->Text, 0)) != 0)
      this->textBox1->AppendText("USB Device not assigned! \n");
    else
    {
      long ziel = NewMotion3x->managed_GetXtarget(0);
      if (ziel == LONG_MAX) this->textBox1->AppendText("?! \n \n");
      else
      {
        this->textBox1->AppendText(" Die ZielPosition von Motor X: " +
                                   ziel + "\n");
      }
    }
  }
}

// Y-Koordinate an die Steuerkarte schicken
private: System::Void textBox3_KeyPress(System::Object^ sender,
                            System::Windows::Forms::KeyPressEventArgs^ e)
{
  if ( e->KeyChar == (char)13 )                  //the ENTER key is pressed
  {
    if (NewMotion3x->managed_SetXtarget(1,
                          xyzDatenEinlesen(this->textBox3->Text, 1)) != 0)
      this->textBox1->AppendText("USB Device not assigned! \n");
    else
    {
      long ziel = NewMotion3x->managed_GetXtarget(1);
      if (ziel == LONG_MAX) this->textBox1->AppendText("?! \n \n");
      else
      {
        this->textBox1->AppendText(" Die ZielPosition von Motor Y: " +
                                   ziel + " \n");
      }
    }
  }
```

```
}

// Z-Koordinate an die Steuerkarte schicken
private: System::Void textBox4_KeyPress(System::Object^  sender,
                                        System::Windows::Forms::KeyPressEventArgs^  e)
{
  if ( e->KeyChar == (char)13 )                    //the ENTER key is pressed
  {
    if (NewMotion3x->managed_SetXtarget(2,
                                xyzDatenEinlesen(this->textBox4->Text, 2)) != 0)
      this->textBox1->AppendText("USB Device not assigned! \n");
    else
    {
      long ziel = NewMotion3x->managed_GetXtarget(2);
      if (ziel == LONG_MAX) this->textBox1->AppendText("?! \n \n");
      else
      {
        this->textBox1->AppendText(" Die ZielPosition von Motor Z: " +
                                   ziel + " \n");
      }
    }
  }
}

// Eingabefeld für den Radius der Kreisbahn
// das Hintergrundthread threadKreisbahn starten
private: System::Void textBox6_KeyPress(System::Object^ sender,
                                        System::Windows::Forms::KeyPressEventArgs^  e)
{
  if ( e->KeyChar == (char)13 )                    //the ENTER key is pressed
  {
    if (textBox4->CanFocus) textBox4->Focus();
    achtelKreisbahn(xyzDatenEinlesen(this->textBox6->Text, 3));

    if (threadKreisbahn != nullptr)
    {
      threadKreisbahn->Abort();
      threadKreisbahn = nullptr;
    }

    threadKreisbahn = gcnew Thread(tsKreisbahn);
    threadKreisbahn->IsBackground = true;
    threadKreisbahn->Start();
  }
}

// wird vom Hintergrundthreads threadKreisbahn aufgerufen.
// Diese ruft zuerst die Methode achtelKreisbahn auf,
// leitet die anderen Punkte auf der Kreisbahn durch die
// Spiegelungen ab, ruft dann die Methode datenSchickenSpeichern
// immer wieder auf, bis die Koordinaten aller Punkte geschickt werden.
private: void kreisbahnFräsen()
{
  try
  {
    int anz = aKreisbahnDaten[0]->Count;
    array<long> ^xDaten = gcnew array<long>(8*anz);
    array<long> ^yDaten = gcnew array<long>(8*anz);
```

```
    for (int i=0; i<anz; i++)
    {
            xDaten[i] = aKreisbahnDaten[0][i];
            yDaten[i] = aKreisbahnDaten[1][i];
            xDaten[i+anz]   = aKreisbahnDaten[1][anz-1-i];
            xDaten[i+2*anz] = aKreisbahnDaten[1][i];
            xDaten[i+3*anz] = aKreisbahnDaten[0][anz-1-i];
            xDaten[i+4*anz] = -aKreisbahnDaten[0][i];
            xDaten[i+5*anz] = -aKreisbahnDaten[1][anz-1-i];
            xDaten[i+6*anz] = -aKreisbahnDaten[1][i];
            xDaten[i+7*anz] = -aKreisbahnDaten[0][anz-1-i];
            yDaten[i+anz]   = aKreisbahnDaten[0][anz-1-i];
            yDaten[i+2*anz] = -aKreisbahnDaten[0][i];
            yDaten[i+3*anz] = -aKreisbahnDaten[1][anz-1-i];
            yDaten[i+4*anz] = -aKreisbahnDaten[1][i];
            yDaten[i+5*anz] = -aKreisbahnDaten[0][anz-1-i];
            yDaten[i+6*anz] = aKreisbahnDaten[0][i];
            yDaten[i+7*anz] = aKreisbahnDaten[1][anz-1-i];
    }

    this->datei2 = gcnew StreamWriter("KreisbahnDaten.txt", true);

    this->datenSchickenSpeichern(0, xDaten[0], 0, yDaten[0]);
    Sleep(2000);              // warte 2 Sekunden

    for (int i=1; i<8*anz; i++)
    {
      if ((i != 2*anz)&&(i != 4*anz)&&(i != 6*anz))
      {
        datenSchickenSpeichern(xDaten[i-1], xDaten[i],
                               yDaten[i-1], yDaten[i]);
      }
    }
  }
  catch (ThreadAbortException^)
  {
  }
  finally
  {
    this->datei2->Close();
    aKreisbahnDaten[0]->Clear();
    aKreisbahnDaten[1]->Clear();
  }
}

// entscheidet nach der aktuellen und der nächsten Position des Fräsers,
// welche Schrittmotoren in Bewegung gesetzt werden sollen
private: void datenSchickenSpeichern(long x1, long x2, long y1, long y2)
{
  if (y2 == y1)
  {
    this->datenSchickenSpeichern1(1, x2, y2);
  }
  else if (x2 == x1)
  {
    this->datenSchickenSpeichern1(2, x2, y2);
  } else this->datenSchickenSpeichern1(3, x2, y2);
```

```
  }

  // Zielposition für X und/oder Y an die Steuerkarte schicken
  private: void datenSchickenSpeichern1(unsigned char motorIndex,
                                        long x, long y)
  {
    Sleep(1000);              // warte 1 Sekunde
    if (NewMotion3x->managed_SetXYZtarget(motorIndex, x, y, 0) == 0)
    {
      this->datei2->WriteLine(x +"," + y + " ok!");
    } else this->datei2->WriteLine(x +"," + y + " nicht erfolgreich!");
  }

  // die Koordinaten der ausgewählten Punkte für die erste Achtelkriesbahn
  // berechnen (Bresenham Algorithmus)
  private: void achtelKreisbahn(long radius)
  {
    long x = 0, r = radius, y = radius;
    long dx,dxy,d;

    d=1-r; dx=3; dxy=-2*r+5;

    aKreisbahnDaten[0]->Add(x);
    aKreisbahnDaten[1]->Add(y);
    this->textBox1->AppendText(" (X,Y) = ("  + x +"," + y + "), \n");

    while (x < y)
    {
      if ((d >= 0) && ((x+1) < (y-1)))
      {
        if ((x-aKreisbahnDaten[0][(aKreisbahnDaten[0]->Count)-1])>=1)
        {
          aKreisbahnDaten[0]->Add(x);
          aKreisbahnDaten[1]->Add(y);
          this->textBox1->AppendText(" (X,Y) = ("  + x +"," + y + "),
                                     " + " \n");
        }

        aKreisbahnDaten[0]->Add(x+1);
        aKreisbahnDaten[1]->Add(--y);
        this->textBox1->AppendText(" (X,Y) = ("  + (x+1) +"," + y + "), " +
                  (Math::Sqrt(((x+1)*(x+1) + y*y))).ToString("N2") + " \n");
        d=d+dxy; dx=dx+2; dxy=dxy+4; x++;
      }
      else
      {
        d=d+dx;  dx=dx+2; dxy=dxy+2; x++;
      }
    }
  }
};
}
```

C.4. VP3undUSBMotion3x.cpp

```
// VP3undUSBMotion3x.cpp: Hauptprojektdatei.
// compile with: /clr
```

```
#include "stdafx.h"
#include "Form1.h"

using namespace VP3undUSBMotion3x;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
  // Aktivieren visueller Effekte von Windows XP,
  // bevor Steuerelemente erstellt werden
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);

  // Hauptfenster erstellen und ausführen
  Application::Run(gcnew Form1());
  return 0;
}
```

D. Literaturverzeichnis

[1] Frank Budszuhn. *Visual C++ / Windows-Programmierung mit den MFC.* Addison-Wesley verlag, 2004.

[2] Coptonix GmbH. Manual usb motion 3x ii, 2008.

[3] Folker Haase. *Eigenspannungsermittlung an dünnwandigen Bauteilen und Schichtverbunden.* Shaker Verlag, 1998.

[4] Karl Hoffmannl. Eine einführung in die technik des messens mit dehnungsmessstreifen, 1987.

[5] Peter Ingerfeld. Konzeption einer architektur für optimierungssoftware zur integration in betriebliche anwendungssysteme, 2008.

[6] Leif Kobbelt and Dominik Sibbing. 32. algorithmus der woche / kreise zeichnen mit turbo, 2006.

[7] Martin Kornmeier. *Analyse von Abschreck- und Verformungseigenspannungen mittels Bohrloch- und Röntgenverfahren.* Gesamthochschul-Bibliothek Kassel Verlag, 1 edition, November 1999.

[8] S. Krömker. Computergraphik i, ws 2009/10, 2009.

[9] Dirk Louis. *Visual C++ 2010 / Das umfassende Handbuch für Programmierer.* Addison-Wesley verlag, 2010.

[10] Vishay Micro-Measurements. Model p3 strain indicator and recorder / activex control developer's guide, 2004.

[11] Vishay Micro-Measurements. Model p3 strain indicator and recorder / instruction manual, 2007.

[12] Vishay Micro-Measurements. Die 3-leiterschaltung für dms-viertelbrücken, 2010.

[13] Vishay Micro-Measurements. Die messung von eigenspannungen mit dem dms-bohrlochverfahren, 2010.

[14] Jochen Wagner. *Ermittlung mechanischer Festigkeitseigenschaften und thermischer Eigenspannungen an ultraschall-geschweißten Keramik/Metall-Verbunden.* Universität Kaiserslautern Lehrstuhl für Werkstoffkunde, 1997.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Jinxu Wu)