Institut für Parallele und
Verteilte Systeme
Abteilung Parallele Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Studienarbeit Nr. 2343

Eigenspannungsmessung an Hochbelastbaren, Keramischen Beschichtungen

Mingzhu Xiu

Studiengang:	Informatik
Prüfer:	Prof. Dr. Sven Simon
Betreuer:	Lars Rockstroh
begonnen am:	24.Mai, 2011
beendet am:	23.November, 2011
CR-Klassifikation:	D.2.3, D.3.3, H.5.2

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Motivation

Die Eigenspannungen können in Werkstoffen und Bauteilen leicht gefunden werden. Und bei den Herstellungsprozessen ist das eine häufige Ursache, weil eine kleine eingebrachte Eigenspannung des Bauteils einen großen Irrtum versehen kann. Deswegen möchte man die Eigenspannung von den Werkstoffen zerlegen, reduzieren oder zerstören.
Wenn jedes Mal ein neues Auto oder ein Zug im Markt hergestellt wird, macht man sich viele Arbeit um das Gewicht zu reduzieren, dass die Geschwindigkeit erhöht und der Kraftstoff gespart wird. Obwohl der leichte oder dünne Werkstoff höhere Effizienz erreichen kann, gibt es den großen Einfluss auf die Sicherheit, wenn die Stärke des Werkstoffs nicht garantiert werden kann. Wenn man im Gegenteil dazu nur die Stärke betrachtet, nimmt das Gewicht zu und der Effekt (der Sicherheit) wird ausgewirkt. Deswegen sind die Koordinierungen zwischen der Sicherheit und der Effizienz in dem Design vom Auto oder Zug auch sehr wichtige Faktoren. Man muss die Eigenspannung von verschiedenen Bauteilen kennen, um die Stärke und die Koordinierung zu gewährleisten.

Was ich oben erläutert habe, sind nur zwei von den Gründen, warum die Erkennung der Eigenspannungen notwendig sind. Normalerweise möchte man die Eigenspannungen vermeiden oder auftrennen, obwohl die Eigenspannungen auf jeden Fall existieren. Man kann durch das Verständnis der Eigenspannungseigenschaften den Einfluss der Eigenspannung reduzieren.
Die Messung von Eigenspannungen in undurchsichtigen Objekten ist mit direkten Methoden (z.B eine experimentelle Spannungsanalyse) nicht möglich, weil der Sensor total unempfindlich für die Geschichte eines Bauteils oder eines Werkstoffs ist. Dieser Sensor kann nur die Änderung von der Dehnung messen.

1.2 Aufgabenstellung

Das Ziel dieser Studienarbeit ist es die Auswertung von den Eigenspannungen mit Hilfe des Dehnungsmessstreifens der Widerstände zu repräsentieren und die Schrittmotoren in Abhängigkeit der Widerstandswerte anzusteuern.

Der Dehnungsmessstreifen ist eine Komponente, die die mechanische Dehnung nach der Mikrodehnung von den Widerständen wechseln kann. Durch eine Brückenschaltung kann man einen unbekannten Widerstand berechnen. Damit kann man die Mikroänderung im Vergleich zum originalen Dehnungsmessstreifen berechnen. Mit anderen Worten bekommt man die Dehnungsänderung. Es gibt ein Gerät, das Modell P3 heißt. Es entspricht allen oben genannten Bedingungen. Das Modell P3 basiert auf der Wheatstone-Brücke mit den Widerständen von 60Ω bis 2000Ω. Man kann den Dehnungsmessstreifen und den

auf dem Dehnungsmessstreifen basierenden Aufnehmer über die Eingangssteckerblocken von den 4 Kanälen verbinden.
Ich benutze eine Platine, die USB Motion 3xII heißt, um 3 Schrittmotoren anzusteuern. Die 3 Schrittmotoren entsprechen den x, y und z Achsen. Mit den 3 Achsen kann man eine Koordinate bestimmen. Jede Koordinate bedeutet ein Bohrloch auf einer Decke. Man kann durch die Platine mit den kleinen Bohrlöchern ein großes Bohrloch aufbauen. Jedes kleine Bohrloch kann aus den Koordinaten von gleicher Tiefe oder verschiedenen Tiefen bestehen.
Die Widerstände, die durch die Messwerte vom Modell P3 berechnet werden, können als die Zielpositionen der Schrittmotoren verwendet werden, um die Dehnungsänderung besser auszudrücken.

2 Grundlage des Modelles P3

Die Eigenspannungen werden mittels Widerstandsdehnungsmessstreifen repräsentiert. Der Dehnungsmessstreifen von den Widerständen ist eine kleine Widerstandsänderung. Mit anderen Worten, das ist ein Präzisionsverfahren von der Messtechnik.
Es gibt 4 verschiedene Präzisionsmethoden [1] für die Widerstandsmessung:

- Kombination von Amperemeter und Voltmeter
- Wheatstone-Brücke für die Widerstände von 5Ω bis $M\Omega$
- Kelvin-Brücke für die Widerstände von $10\mu\Omega$ bis 1Ω
- Amperemeter mit hoher Spannung (mehr als $100V$) und niedrigem Strom (μA) für die Widerstände von $10M\Omega$ bis $G\Omega$

Ich arbeite mithilfe der Wheatstone-Brücke, damit kann man die Mikrodehnungen von den Widerständen messen.

2.1 Brückenschaltung

Die Brückenschaltung besteht normalerweise aus fünf Widerständen, von denen immer zwei in einer Reihenschaltung sind, anschließend verschaltet man die beide Reihenschaltungen in parallel. Zwischen den Beiden Reihenschaltungen gibt es ein Voltmeter, welches mit dem übrigen Widerstand parallelgeschaltet ist, was in der Abbildung 2.1 gezeigt wird, aber in der Abbildung 2.1 muss noch ein Widerstand mehr drin aufgebaut werden. Weil die Form wie das Großbuchstaben "Häussieht, kann man die Schaltung auch die H-Schaltung, H-Brücke oder Vollbrücke nennen.
Die Abbildung 2.1 repräsentiert eine Wheatstone-Brücke. Die Wheatstone-Brücke (die Abkürzung von Wheatstonesche Messbrücke) kann für die Messung der kleinen ohmschen Widerstandsänderung verwendet werden. Im Vergleich zu der Brückenschaltung hat die Wheatstone-Brücke einen Widerstand weniger.
In einer Reihenschaltung von den zwei Reihenschaltungen ist der Spannungsteiler tauglich. D.h jedem Widerstand in der Reihenschaltung wird eine elektrische Spannung verteilt. Deswegen kann man wissen:

$$U_1 = U_0 \frac{R_1}{R_1 + R_2} \tag{2.1}$$

$$U_3 = U_0 \frac{R_3}{R_3 + R_4} \tag{2.2}$$

$$U = U_1 - U_3 = U_0 \frac{R_1 R_4 - R_2 R_3}{(R_1 + R_2)(R_3 + R_4)} \tag{2.3}$$

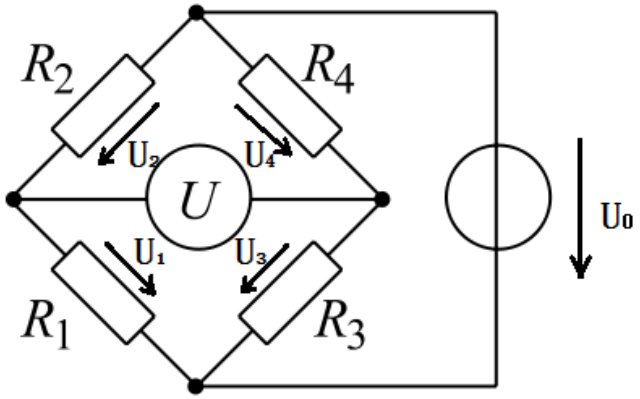


Abbildung 2.1: Ein Beispiel der Wheatstone-Brücke

Man definiert einen abgeglichenen Zustand, wenn die Spannung von dem Voltmeter 0 wäre. D.h man hat jetzt eine Voraussetzung $U = 0$. Dann bekommt man:

$$R_1 R_4 = R_2 R_3 \tag{2.4}$$

oder

$$\frac{R_1}{R_2} = \frac{R_3}{R_4} \tag{2.5}$$

Diese beiden Gleichungen bedeuten, dass man im abgeglichenen Zustand den vierten Widerstand durch die andere schon bekannten Widerstände ausrechnen kann.
Die Wheatstone-Brücke ist in der heutigen Zeit für die Präzisionsmessungen geeignet. Die Brücke ist sehr wichtig in der Messtechnik. Die elektrische Größe ist nicht so wichtig sondern eher die kleine Widerstandsänderung aus dem abgeglichenen Zustand. Es gibt zwei Methoden um die Änderung zu ermöglichen. Eine davon ist mit dem Widerstandsthermometer und die andere ist mit dem Dehnungsmessstreifen. Der Widerstandsthermometer ist abhängig von den Widerständen, die mit der geänderten Temperatur verändern werden. Der Dehnungsmessstreifen wird durch die Verformung der Widerstände verwendet. Die Verformung kann durch die Eigenspannung oder auch externe Kraft vorkommen.
Bei der Situation entsteht normalerweise eine Spannung U als Maß für eine Änderung des Widerstands ΔR. Wenn die aus dem abgeglichenen Zustand heraus R_1 ändert, d.h $R_1 \rightarrow$R$_1$+ΔR_1, dann wird die Gleichung 2.3 gemeinsam geändert:

$$\frac{U}{U_0} = \frac{R_1 + \Delta R_1}{R_1 + \Delta R_1 + R_2} - \frac{R_3}{R_3 + R_4} \tag{2.6}$$

Wenn es $k\epsilon = \frac{\Delta R_1}{R_1}$ (aus der Gleichung 2.10) und eine Voraussetzung $\frac{R_2}{R_1} = \frac{R_4}{R_3} = 1$ gibt, dann

$$\frac{U}{U_0} = \frac{1}{4}\frac{\Delta R_1}{R_1} = \frac{1}{4}k\epsilon \tag{2.7}$$

Dann kann man wissen. Wenn die alle vier Widerständen jeweils eine kleine Änderung entstehen, erhält man bei symmetrischer Brücke [4]:

$$\frac{U}{U_0} = \frac{1}{4}(\frac{\Delta R_1}{R_1} - \frac{\Delta R_2}{R_2} - \frac{\Delta R_3}{R_3} + \frac{\Delta R_4}{R_4}) \tag{2.8}$$

In meine Arbeit verwende ich den Dehnungsmessstreifen mit der Wheatstone-Brücke zusammen, um das Verhältnis der Eigenspannung zu erkennen. Normalerweise, wenn man eine Wheatstone-Brücke verwenden würde, sollte man vor allem die Widerstandsdekaden haben. Durch die Einstellung der Widerstandsdekaden um den abgeglichenen Zustand zu erreichen kann man den unbekannten Widerstand ausrechnen. Bei der Wheatstone-Brücke mit dem Widerstandsdehnungsmessstreifen muss die Methode jedoch ein bisschen geändert werden. Man kann mit der Hilfe eines Messwertaufnehmers, der auf Dehnung basiert, die Mikrodehnung direkt bekommen oder einen relativen Messwert bekommen. Bei der geänderten Methode braucht man keine Widerstandsdekaden sondern die Widerstandsdehnungsmessstreifen statt der Widerstände.

2.2 Dehnungsmessstreifen

Der Dehnungsmessstreifen (kurz: DMS) besteht aus einer mit Metall beschichteten Folie, aus der die Form des Messgitters gelegt wird, was wie in der Abbildung 2.2 gezeigt wird. Der Streifen wird mit speziellen Klebern auf das Werkstück appliziert, mit Anschlussdrähten versehen und meist mit einem Schutzlack abgedeckt. Der verändert seinen Widerstand durch die mechanischen Spannungen. Der wird fest mit der Werkstücks-Oberfläche verklebt. Man benutzt die speziellen Messgeräte, um sehr kleine Widerstandsänderung zu messen. Es gibt einige typische Widerstandswerte, zum Beispiel 120Ω,

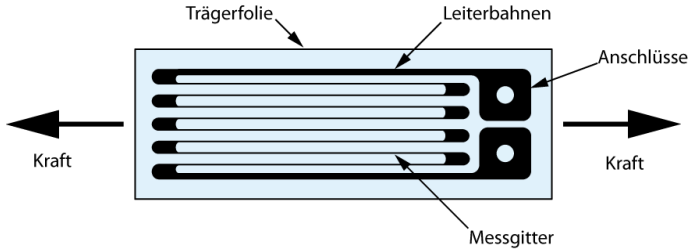


Abbildung 2.2: Aufbau eines typischen Dehnungsmessstreifen

350Ω, 600Ω und 1000Ω.
Die originale Form von der Widerstandsberechnung ist:

$$R = \rho \frac{l}{A} \tag{2.9}$$

Wenn der Drähten unter einer Zugdehnung (wie die Abbildung 2.2) oder einer Druckdehnung sind, werden l, ρ und A geändert. Man ersetzt die Rate des Widerstandsdehnungsmessstreifens mit zwei Koeffizienten k und ϵ.

$$\frac{\Delta R}{R} = k\epsilon \tag{2.10}$$

Die Dehnung oder die Verformung gibt die relative Längenänderung an. Sie ist die sogenannte Mikrodehnung.

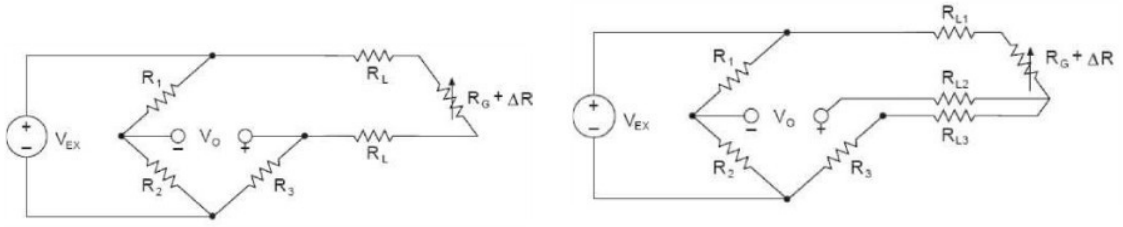
$$\epsilon = \frac{\Delta l}{l} \tag{2.11}$$

Der k-Wert heißt auch Koeffizient der Empfindlichkeit von den Drähten. Die physikalische Bedeutung ist, dass die Größe des k-Werts das Verhältnis zwischen der Widerstandsrate und der Dehnung ist, wenn die Längeneinheit von Drähten geändert wird. Er besteht aus zwei Faktoren. Einer der Faktoren wird wegen der geometrischen Verformung und mit der Form $1+2\mu$ angezeigt. Der andere ist wegen der Änderung von ρ und repräsentiert mit λE. Man bekommt die Form von k-Wert:

$$k = 1 + 2\mu + \lambda E \tag{2.12}$$

Die Koeffizienten sind: λ ist der Piezowiderstandskoeffizient, μ ist die Poissonszahl, E ist die Dehnungsmasse vom Drähten.
Bei hohem k-Wert ergibt sich bei gleicher Dehnung eine große Widerstandsänderung (und damit ein hohes Messsignal). Der k-Wert wird auch durch den Gefügeaufbau und die Vorgänge im Gefüge während der Dehnung bestimmt. Der k-Wert liegt bei den meist verwendeten Metallen bei $k = 2$.[11]

Es gibt die Viertel-, Halb- und Vollbrückenschaltungen. Bei den Viertel- und Halbbrückenschaltungen verwendet man normalerweise Brückenergänzungswiderständen, um die Schwierigkeiten von der Kalibrierung zu reduzieren.
Zum Beispiel, jetzt werden zwei Schaltungen in der Abbildung 2.3 gegeben. Eine ist 2-Leite-Verdrahtung (Abbildung 2.3(a)) und die andere ist die 3-Leite-Verdrahtung (Abbildung 2.3(b)). In einer Viertelbrückenschaltung der 2-Leite-Verdrahtung führt eine lange



(a) 2-Leite-Verdrahtung (b) 3-Leite-Verdrahtung

Abbildung 2.3: Zwei Viertelbrückenschaltungen

Leite einen Widerstand in dieser Schaltung, dadurch wird der abgeglichene Zustand gebrochen. Die $6m$ Lange Leite wird $29000\mu\epsilon$ Änderung in den abgeglichenen Zustand mitgebracht. Gleichzeitig gibt es auch den Einfluss von der Temperatur. Wenn man mit der Hand an die Leite fasst, wird die Änderung mehr als $100\mu\epsilon$ betragen.
In einer Viertelbrückenschaltung der 3-Leite-Verdrahtung wird eine $6m$ Lange Leite nur $400\mu\epsilon$ in den abgeglichenen Zustand mitgebracht. In dieser Situation ist der abgeglichene Zustand einfach zu ändern. Die Temperatur hat also keinen Einfluss darauf.

2.3 Modell P3

Das Modell P3 wird von Vishay hergestellt. Die Grundidee ist eine Wheatstone-Brücke, die mit den Dehnungsmessstreifen aufgebaut wird. Und alle verwendeten Informationen (z.B die Daten von der Dehnungsmessbrücke) können in die Multimediakarte oder durch

USB auf dem PC gespeichert werden. Das ist ein tragbares und mit Batterie gespeistes Präzisionsmessinstrument.
Die Viertel-, Halb- und Vollbrückenschaltung können mit dem Modell P3 angeschlossen werden [8]. Die 120Ω, 350Ω und 1000Ω Widerstände werden als Brückenergänzungen in die Viertelbrückenschaltungen eingesetzt und 1000Ω Brückenergänzungswiderstände werden in den Halbbrückenschaltungen eingesetzt. Die Anschlussverbindungen können ohne Werkzeuge durchgeführt werden. Die Brückenimpedanz ist von 60Ω bis 2000Ω möglich.
Das Modell P3 enthält die Algorithmen zur Skalierung und Linearisierung für die gemessene Datenbearbeitung des Dehnungsmessstreifens.

2.3.1 Frontplatte

Die Frontplatte besteht aus folgendem: USB-Interface, Speicherkarteneinschub, Ein-

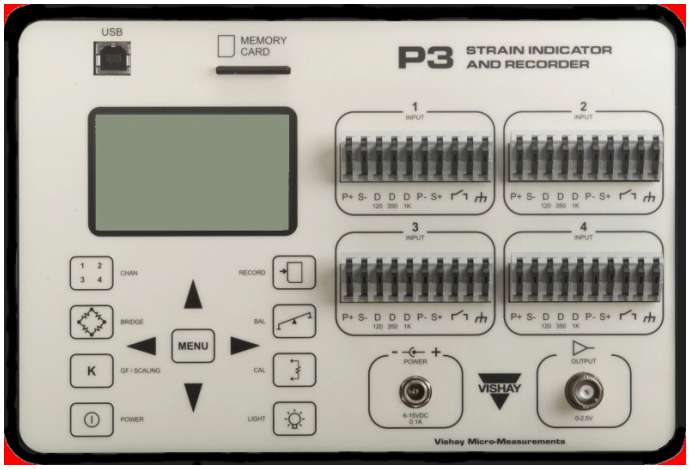


Abbildung 2.4: Modell P3

gangsklemmverbindungsblocks, LCD-Anzeige, Softkey-Tastatur, Netzadapter-Eingang und Analogausgang.
Der Einsteckschlitz nimmt eine Multimediaspeicherkarte nach Industriestandard auf. Alle Messdaten können in der Karte gespeichert werden. Und die Daten können durch einen Softwarebefehl über USB auf einen PC oder mittels eines Multimediakarte-Lesegeräts in einen Computer transferiert werden [8].
Der Bildschirm zeigt alle Einstellung- und Messdaten an. Den oberen Teil des Bildschirms unterteilt man in vier Quadranten, je einer für einen Kanal. Der Status (von Speicherung und Kalibrierung) des Modells P3 wird in der unteren Zeile links und rechts angezeigt.
Man kann die Einstellung und die Anwendung durch die Softkey-Tastatur erzielen und auch durch das USB-Interface mit dem Befehl vom Computer schaffen und ändern. Die Eingangsklemmanschlüsse verbinden die Dehnungsmessstreifen oder der auf dem Dehnungsmessstreifen basierenden Messwertaufnehmer mit dem Instrument. [8]
Es stehen vier Kanäle mit den Nummern eins bis vier zu Verfügung, um den Dehnungsmessstreifen und den Aufnehmer über die Eingangssteckerblocken anzuschließen.
Das Modell P3 bietet alle möglichen Brückenschaltungen z.B Viertel-, Halb- und Voll-

brückenschaltungen. Wie man die Dehnungsmessstreifen mit dem Modell P3 verbindet wird in der Abbildung 2.5 gezeigt. Die Viertel- und Halbbrückenschaltung werden als 3-Leite-Verdrahtung (Abbildung 2.3(b)) verwendet. In der Brückenverdrahtung der Vollbrückenschaltung steht ein Kurzschlussschalter für eine externe Kalibrierschaltung zur Verfügung.

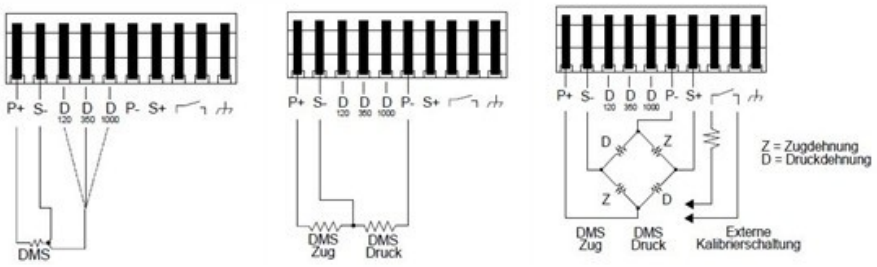


Abbildung 2.5: Viertel-, Halb- und Vollschaltungen

2.3.2 Anwendungen

Alle Anwendungen vom Modell P3 habe ich in meiner Arbeit durch ein GUI-Programm geschaffen. Damit kann man die Einstellung, die Messwertvorlesen und die Speicherverfahren mit dem Computer ansteuern.

Versorgungsmöglichkeiten

Das Modell P3 hat drei Versorgungsmöglichkeiten: mit Batterie, über USB-Schnittstelle oder über einen AC-Adapter. Wenn mehr als eins von diesen Versorgungen vorliegen, wird es in folgender Reihenfolge benutzt: USB, danach AC-Adapter und dann die Batterie.

Kanalauswahl

Die Kanäle sind von eins bis vier definiert. Jeder Kanal hat 10 Pins. Man kann durch die Softkey-Tastatur die Aktivierung oder Deaktivierung der Messeingangskanäle auswählen. Der Messwert vom aktiven Kanal wird auf dem Bildschirm angezeigt. Die Anzeigeerneuerungsrate ist 2 pro Sekunde. Die Abbildung 2.6(a) zeigt das GUI von meiner Arbeit.

Auswahl der Brückenart

Es gibt 11 Auswahlmöglichkeiten von der Brückenart: 1 Viertelbrückenschaltung, 4 Halbbrückenschaltungen, 5 Vollbrückenschaltungen und eine undefinierte Viertel- oder Halbbrückenschaltung. Ich habe für jede Art nur eine kurze Erklärung in meinem GUI wie in der Abbildung 2.7(a) steht. Die genaue Erklärung kann Man im Manual gut erfassen. Und man kann mehr Informationen der Rechnung im Homepage `http://www.vishaypg.com/micro-measurements/` von Vishay bekommen.

k-Faktor und Skalierung

Zuerst muss man wissen, was für ein Messwert auf dem Bildschirm steht. Deswegen ist die Erstellung von der Dehnungseinheit sehr wichtig. Und die Auswahl der Skalierung ändert sich wegen der Dehnungseinheit. Hier gibt es 3 Möglichkeiten, um die Skalierung auszuwählen.

- Wenn die Dehnungseinheit ϵ (Mikrodehnung) ausgewählt wird, wird die Skalierung durch den k-Faktor determiniert. Der default k-Faktor ist 2.000. Zum Beispiel, wenn der Messwert $1\mu\epsilon$ mit $k = 2$ ist, wegen der Gleichung 2.7:

$$\frac{U[mV]}{U_0[V]} = \frac{1}{4} * 2 * 1\frac{[\mu m]}{[m]} = 0.0005\frac{[mV]}{[V]} \tag{2.13}$$

- Wenn die Messwerteinheit $\frac{mV}{V}$ ausgewählt wird, ist keine Skalierung notwendig.

- Wenn die Messwerteinheit außer $\mu\epsilon$ und $\frac{mV}{V}$ sondern eine andere Ingenieureinheit verwendet wird, muss man den Vollausschlag und der Vollausschlag in $\frac{mV}{V}$ geben. Der default Vollausschlag ist 1, und kann zwischen $+99999$ bis -99999 außer 0 geändert werden. Der default Vollausschlag in $\frac{mV}{V}$ ist 2.000, und die Änderungsbereich ist zwischen $+15.625\frac{mV}{V}$ und $-15.625\frac{mV}{V}$ außer 0.

Die Abbildung 2.6(d) zeigt alle Möglichkeiten, welche ich in Python geschrieben habe.

Brückenabgleich und Kalibrierung

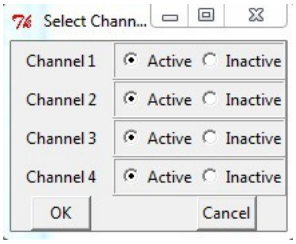
Man kann bei jedem Kanal den Zustand des Brückenabgleichs machen. Es gibt ohne Abgleichen, einen automatischen und einen manuellen Brückenabgleich wie in der Abbildung 2.6(b) angezeigt wird. Wenn es ohne Abgleichen ist, wird kein korrigierter Abgleich gegeben. Und der Benutzer kann die initiale Brücke abschätzen.
Durch die Kalibrierung kann man die Empfindlichkeit des gewählten Kanals einstellen.

Speicherung

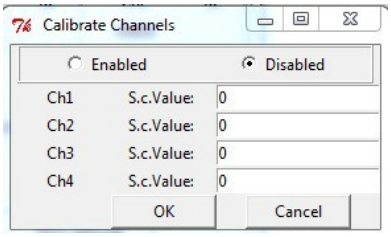
3 Speichermethoden werden im Modell P3 verwendet. Sie sind keine Datenspeicherung, manuelle Speicherung und automatische Speicherung. Die manuelle Speicherung arbeitet durch eine Taste. Jedes Mal, wenn man die Messwerte von den aktiven Kanälen speichern möchte, drückt man diese Taste. Die automatische Speicherung läuft durch die gegebene Speicherzeit. Das Modell speichert die Messwerte je nach der gegebenen Zeit. Die beiden Methoden enden, wenn man die Methode als keine Datenspeicherung auswählt. Und alle Speicherdaten werden in der Multimediaspeicherkarte gelegt.
Ich habe in meinem GUI-Programm definiert, dass die Messwerte auch direkt in den Desktop des Computers als CVS-Datei speichert werden können. Man braucht nicht mithilfe des Multimediakarte-Lesegeräts die Messwerte aus der Multimediakarte auf dem PC zu speichern. Die explizierte Realisierung sieht in der Abbildung 2.7(b).

2.3.3 Optionen

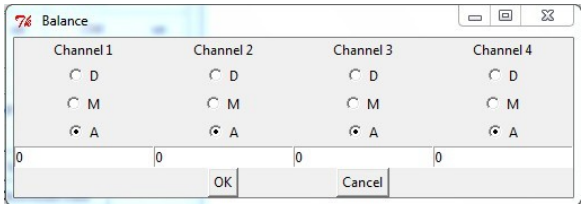
Man kann in diesem Menü die aktuelle Zeit des Modells P3 korrigieren, die Beleuchtung und den Kontrast des Bildschirms einstellen, die Multimediakarte löschen, die Poissonzahl (Dieses Zahl ist wichtig für die Dehnung bei den Brückenschaltungen.2.2) ändern, den Analogausgang ansteuern, die Firmware-Version des Modells P3 anzeigen und neue Version laden, die vorliegenden Einstelldaten im Flash-Memory speichern, die aufgenommenen Messdaten von der Multimedia-Karte auf den PC laden u.s.w.
Die Bilder in der Abbildung 2.8 sind die GUIs, die ich in Python gemacht habe. Die Abbildung 2.8(b) ist das GUI um die aktuelle Zeit und Datum abzulesen und zu korrigieren. Die Abbildung 2.8(a) zeigt ein GUI vom Bildschirm. Sie kann die Beleuchtungsdauer von $5s$, $15s$ und $60s$ umgeschaltet werden. Während der Programmierung bin ich auf ein Problem gestoßen, dass der Umschalter von $5s$ in Python unmöglich ist. Aber für andere Zeitdauern gibt es keine solchen Probleme. Man kann nur durch die Softkey-Tastatur den Umschalter von $5s$ ermöglichen. Es gibt auch ein GUI für die weiteren Optionen. Das steht in der Abbildung 2.8(c). Der wichtige Koeffizient von der Empfindlichkeit k-Wert ist in diesem Menü und heißt die Poissonzahl. Die Poissonzahl ist der Name von der Querdehnungszahl und eine wichtige Zahl in der Mechanik und Festigkeitslehre. Wenn man das Modell P3 benutzt, muss die Zahl mehr beobachtet werden.



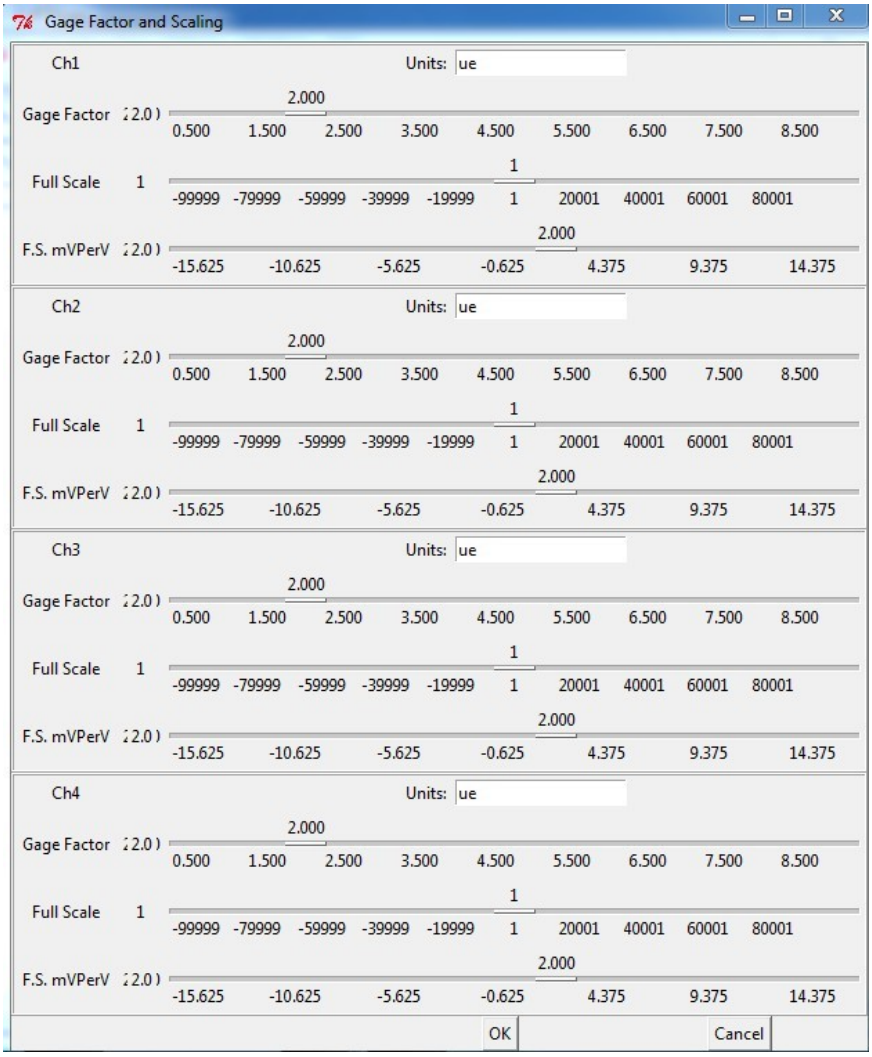
(a) Kanalauswahl



(b) Kalibrierung

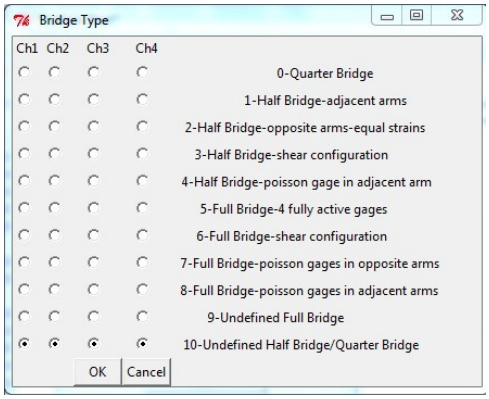


(c) Brückenabgleich

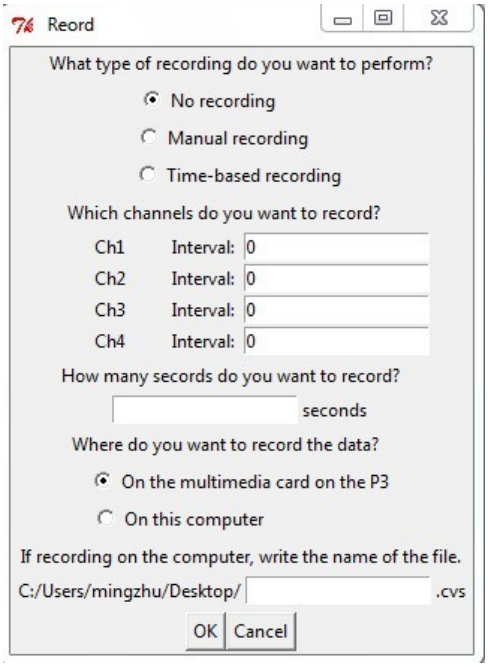


(d) k-Faktor und Skalierung

Abbildung 2.6: Anwendungen(1)



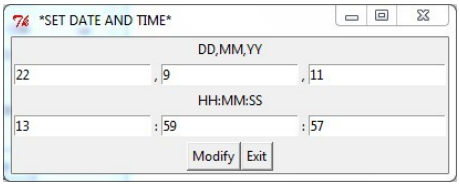
(a) Auswahl der Brückenart



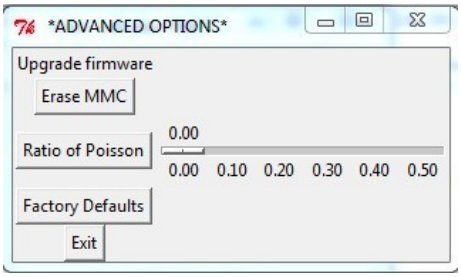
(b) Speicherung

Abbildung 2.7: Anwendungen(2)

(a) Bildschirm



(b) Zeiteinstellung



(c) Weitere Optionen

Abbildung 2.8: Optionen

3 Grundlage von USB Motion 3xII

Die USB Motion 3xII Platine verwendet man, um die Schrittmotoren anzusteuern.

3.1 Grundlage der Schrittmotoren

Der Schrittmotor ist ein Synchronmotor und besteht aus Stator und Rotor. Das Eingangssignal ist der Impulsstrom und das Ausgangssignal ist eine Winkelverschiebung oder eine Positionsveränderung. D.h. Wenn ein Impulsstrom in einem Schrittmotor eingegeben wird, dreht der Motor sich um einen Winkel oder bewegt der sich um einen Schritt nach vorn.
Wenn man die Eigenschaft vom Schrittmotor begreift, kann man dann den Schrittmotor ansteuern. Ohne Ansteuerung kann der Rotor sich frei drehen. D.h. es gibt keinen Kraftangriff. Die Reihenfolge des bestromten Stators determiniert die Bewegungsrichtung der

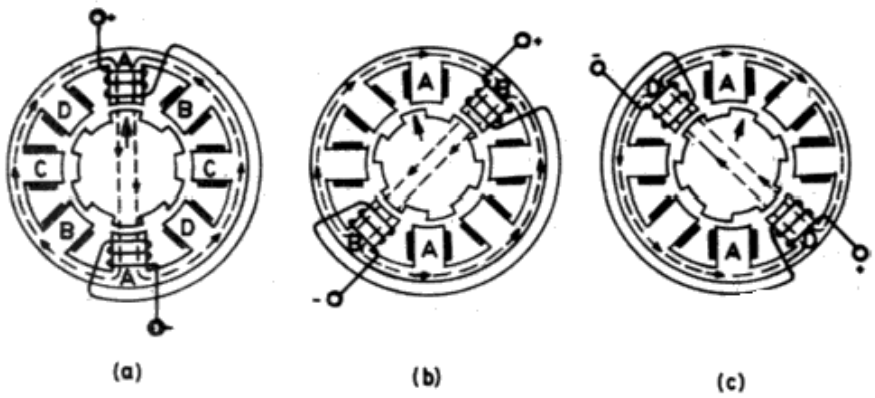


Abbildung 3.1: Betrieb des Schrittmotors

Motorachse. Die Drehgeschwindigkeit der Motorachse ist abhängig von der Frequenz des Stroms.

In der Abbildung 3.1 hat der Schrittmotor ein Polpaarzahl von $P = 3$ und ein Strangzahl von $S = 4$. Um einem kürzesten magnetischen Weg zu erreichen, lässt sich der Rotor im magnetischen Feld drehen. Dann werden der Pol vom Rotor und der Strang vom Stator in gleicher Richtung sein. Der Abstand zwischen diesem Pol und Strang ist am kleinsten. Z.B Der Strang A (in der Abbildung 3.1 (a)) wird bestromt. Dann bildet es am Stator oben ein Nordpol und unten ein Südpol. Danach gibt es zwei Möglichkeiten für die Drehung: gegen den Uhrzeigersinn (zeigt in der Abbildung 3.1 (b)) und im Uhrzeigersinn (zeigt in der Abbildung 3.1 (c)). In der Abbildung (b) wird der Strang B bestromt. Ein magnetisches Feld entsteht. Zuerst gibt es einen Winkel zwischen dem Strang B und dem Pol, der rechts von dem Pfeil in der Abbildung (a) steht. Dann ist

der Winkel gleich 0 wegen des magnetischen Feldes. Das ist wie ein Schrittmotor, der gegen den Uhrzeigersinn laufen kann. Die Abbildung (c) zeigt wie ein Schrittmotor im Uhrzeigersinn laufen kann. Das Prinzip ist analog wie die Abbildung (b). Deswegen gibt es zwei Möglichkeiten für die Sequenz von dem Bestromen: A-B-C-D und A-D-C-B [6].

3.2 USB Motion 3xII

3.2.1 Aufbau der Platine

Die USB Motion 3xII heißende Platine wird vom Coptionix GmbH in Berlin hergestellt. Der USB Motion 3xII ist eine Schrittmotorsteuerkarte mit einem USB-I^2C-Adapter in einer Platine. Und die Platine wird durch einen 32*bit* Controller aufgebaut. Die Platine

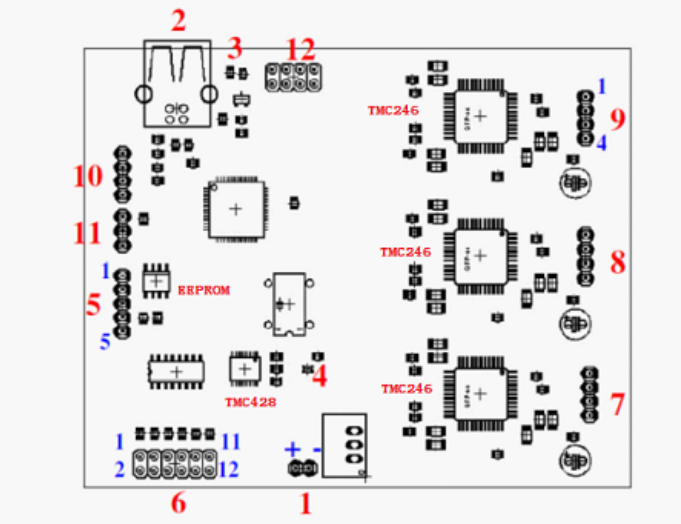


Abbildung 3.2: USB Motion 3xII Platine

arbeitet mit einer Spannungsversorgung von $7V$ bis $34V$. Die Schnittstelle in der Abbildung 3.2 ist der Nummer 1.
Die Platine kann bis zu drei 2-Phasen Schrittmotoren ansteuern. Die 3 Schrittmotoren repräsentieren die 3 Achsen x, y und z. Alle 3 Achsen können unabhängig voneinander durch einen TMC428 Chip angesteuert werden. Der TMC428 Chip ist ein kleiner aber hoch effizienter Ansteuerungschip für maximalen 3 2-Phasen Schrittmotoren. Er kann die Positionen, Geschwindigkeiten und Mikroschritte kontrollieren und kann gleichzeitig 3 Schrittmotoren unabhängig ansteuern. Er arbeitet in 4 verschiedenen Arbeitsmodi. Man kann für jeden Schrittmotor unabhängig programmieren. Es gibt RAMP-, SOFT-, VELOCITY- und HOLD-Modus, um die Positionen zu kontrollieren [2]. Ich habe ein GUI wie in der Abbildung 3.3(a) während meiner Arbeit programmiert. Mit diesem Chip kann man die unterschiedlichen Mikroschritte ermöglichen. In dieser Platine ist der Mikroschrittbetrieb neben dem Vollschritt- und Halbschrittbetrieb mit bis zu 64 Mikroschritten pro Vollschritt möglich. Der Schrittmotorcontroller bietet ein Interrupt-Register mit 8 Flaggen für jeden Schrittmotor. Jedes Interrupt-Bit kann mithilfe des assoziierten Interrupt-Maske-Bits entweder aktiviert (1) oder deaktiviert (0) sein. Wenn das Maske-Bit 0 gesetzt wird, wird das entsprechende Interrupt-Flagge nach 1 gezwun-

Tabelle 3.1: 4 Positionsansteuerungsmodi

Modus	**Wert**	**Beschreibung**
RAMP_Modus	0x00	default Modus für die Positionsanwendung mit trapezförmiger Rampe
SOFT_Modus	0x01	ähnlich wie RAMP-Modus aber zum Zielposition mit weichem Nahen
VELOCITY_Modus	0x02	Modus für die Anwendung der Geschwindigkeitskontrolle, mit der linearen Rampe um die Geschwindigkeit zu ändern
HOLD_Modus	0x03	Geschwindigkeit wird durch den Benutzer kontrolliert. Die Begrenzung des Bewegungsparameters wird ignoriert.

gen. Die Beschreibungen kann man in der Tabelle 3.2 erkennen. Das GUI von meiner Arbeit wird wie 3.3(b) gezeigt.

Die Nummern 8, 9, 10 in der Abbildung 3.2 sind die Anschlüsse von diesen 3 Schrittmotoren. Jeder Schrittmotor wird durch einen TMC246 Chip angetrieben. Der TMC246 Chip ist eine Mikroschrittantriebschip von einem 2-Phasen Schrittmotor. Der Antrieb des Chips kann durch einen SPI-Interface sowie durch das analoge oder digitale Eingangssignal kontrolliert werden. Die Versorgungsspannung des Motors ist zwischen $7V$ und $34V$. Der digitale Teil braucht eine Versorgungsspannung von $3.3V$ oder $5V$.

3.2.2 Anwendungen von USB Motion 3xII

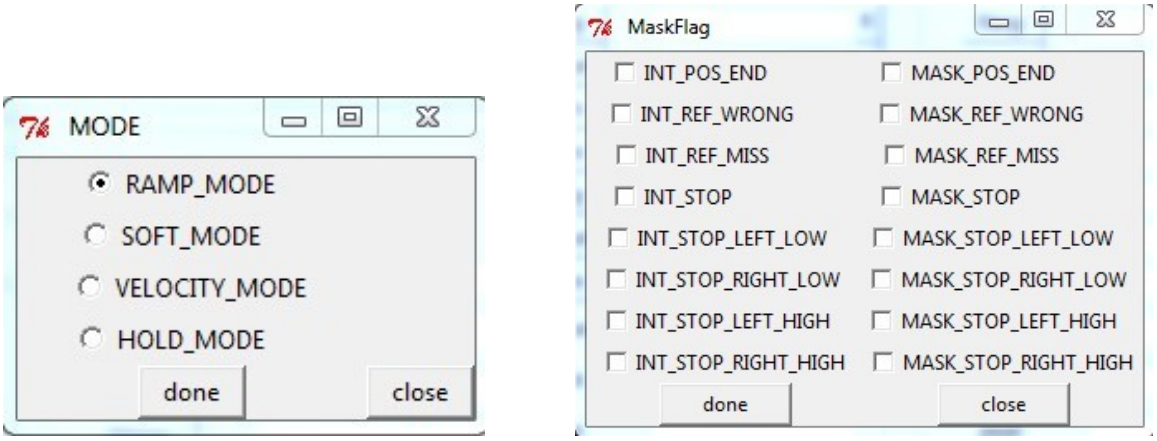
Die Platine steuert die Schrittmotoren durch die gegebenen Funktionen in der DLL an. Man kann via USB (Nummer 2 in der Abbildung 3.2) mit dem Computer verbinden. Die DLL enthält USB Funktionen, I^2C Funktionen, Schrittmotor Funktionen, extended stepper motor Funktionen und Motion calculation Funktionen. Wenn man eine Funktion implementieren möchte, soll man nicht selbst neu programmieren sondern man benutzt direkt eine vorhandene Funktion in DLL. Alle Funktionserklärungen werden in [2] gelegt. Man kann einfach die Funktion aufrufen und die entsprechende Argumente einfügen. Aber die Datentypen müssen mehr beobachtet werden und man muss auch unterscheiden, ob ein Argument *byValue* oder *byReference* aufgerufen wird. *ByReference* ist im Manuel via *var* erkennbar. Wenn man den Rückgabewert von einer Funktion wissen möchte, würden die Resultate von der DLL-Funktion vor dem Aufrufen der Funktion als Modul *ctypes* Datentyp angegeben werden.
Die USB Funktionen sind für die Verwaltung von USB-Geräten verfügbar. Man kann die maximale und minimale Geschwindigkeiten, die Zielposition, die Beschleunigung und alle aktuelle Zustände von den 3 Schrittmotoren ermitteln und auch verändern. D.h alle Anwendungen arbeiten in zwei Richtungen mit zwei verschiedenen Funktionen. Die Ziel-

Tabelle 3.2: Interrupt-Flagge und Maske

Interrupt	**Bit**	**Beschreibung**
POS_END	0x01	Der Schrittmotor erreicht die Zielposition.
REF_WRONG	0x02	Das Signal von dem Referenzschalter ist aktiv außer dem Toleranzbereich des Referenzschalters.
REF_MISS	0x04	Das Signal des Referenzschalters ist verloren auf der Nullposition.
STOP	0x08	Der Motor wird zum Stoppen während der Bewegung gezwungen.
STOP_LEFT_LOW	0x10	hoher bis niedriger Übergang von links Referenzschalter
STOP_RIGHT_LOW	0x20	hoher bis niedriger Übergang von rechts Referenzschalter
STOP_LEFT_HIGH	0x40	niedriger bis hoher Übergang von links Referenzschalter
STOP_RIGHT_HIGH	0x80	niedriger bis hoher Übergang von rechts Referenzschalter

positionen von den 3 Schrittmotoren können durch eine Funktion gleichzeitig eingegeben oder durch eine Funktionen einmal für einen Schrittmotor insgesamt 3 Mal ermöglichen. Für jede Funktion gibt es einen Rückgabewert, der als $ReturnValues$ in [2] explizit erklärt wird, und manchmal einen zusätzlichen Rückgabewert, der in der Eingabe von der Funktion via *var* erkennbar ist. Für die Anwendungen der Schrittmotoren gibt es normalerweise einen *Status* heißenden zusätzlichen Rückgabewert, der durch einen Register zu repräsentieren ist. Die Bit-Bedeutungen des Register werden in der Tabelle 3.3 angezeigt.

Es gibt auch einige Koeffizienten, die man zuerst einstellen muss, um die Schrittmotoren besser ansteuern zu können und zu betreiben. Z.B bei der *SetCoilCurrent* Funktion repräsentiert die Parameter *agtat* (wenn die Beschleunigung größer als die Schwellenbeschleunigung ist), *aleat* (wenn die Beschleunigung kleiner als die Schwellenbeschleunigung ist), $v0$ (wenn der Schrittmotor eine Pause hat, um das Power zu sparen) und *threshold* die aktuelle Skalierung. Die Parameter werden mit dem Motor angewendet und sind abhängig von der Rampe-Phase. Die ersten drei Parameter sind normalerweise 3 Bits lang. D.h Diese Werte müssen von eins bis sieben sein. Ich habe bei meinem GUI eine Überwachung für die Werte gemacht. Das ist das, was in der Abbildung 3.2.2 liegt. Wenn man diese Argumente einführen würde, muss man mehr beobachten. Diese Funktion wird auch durch die Funktion *EnableDriveChain* beeinflusst. Die Funktion *EnableDriveChain* hat einen $RetValue$ heißenden zusätzlichen Rückgabewert. Durch



(a) 4 Modi

(b) Interrupt-Flagge und Maske

Abbildung 3.3: GUI der Schrittmotoransteuerung

das *RetValue* (Die Erklärung ist in der Tabelle 3.4) kann man wissen, welcher Motor aktiviert ist und welcher deaktiviert ist. Wenn der Motor deaktiviert wäre, würden die Koeffizienten, die oben genannt wurden, auf 0 reduziert werden und gäbt es keine Bewegung.
Um die Platine in Python zu programmieren, gibt es einige Funktionen, die man am Anfang des Codes und zum Schluss des Codes aufrufen muss. Die Funktion *USBMCCreate* muss am Anfang installiert werden, dann wird ein Objekt oder eine Klasse erstellt. Wenn diese Funktion erfolgreich aufzurufen ist, muss die Funktion *USBMCInit* aufgerufen werden. Die Funktion *USBMCDestroy* muss zum Schluss des Code aufgerufen werden.

Während meiner Arbeit habe ich ein unsicheres Problem getroffen. Wenn ich das GUI von den Funktionen *SetSwitchSettings* und *GetSwitchSettings* machte, glaubte ich, dass die Bit-Abbildung von dem Argument *Settings* falsch ist. Aber ich bin mir unsicher, ob ich Recht habe. Meine Vorschlagzahlen stehen in der Richtung des Pfeils in der Tabelle 3.5. Die Funktion *SwitchSettings* kann das Verhalten des Referenzschalters bestimmen. Dieser Schalter kann als Referenzschalter sowie automatischen Stopp-Schalter verwendet werden. Die Bit-Bedeutungen können im Manual [2] bekommen werden.

Tabelle 3.3: Bit-Bedeutung des Statusregisters

Status	**Bit**	**Beschreibung**
xEQt1	0x01	Schrittmotor 1 erreicht die Zielposition.
RS1	0x02	Referenzschalter des Schrittmotors 1 ist nach links.
xEQt2	0x04	Schrittmotor 2 erreicht die Zielposition.
RS2	0x08	Referenzschalter des Schrittmotors 2 ist nach links.
xEQt3	0x10	Schrittmotor 3 erreicht die Zielposition.
RS3	0x20	Referenzschalter des Schrittmotors 3 ist nach links.
CDGW	0x40	Wartungsbit des Abdeckungsdatagramm
INT	0x80	Interrupt-Bit

Tabelle 3.4: Bit-Bedeutung von RetValue

Bit-Abbildung	**Bedeutung**
0x01	Falls gesetzt ist Motor 1 aktiviert.
0x02	Falls gesetzt ist Motor 2 aktiviert.
0x04	Falls gesetzt ist Motor 3 aktiviert.

Tabelle 3.5: Bit-Adresse von Settings

Settings	**Bit-Abbildung**
DISABLE_STOP_L	0x00 $\rightarrow$ 0x01
DISABLE_STOP_R	0x01 $\rightarrow$ 0x02
SOFT_STOP	0x02 $\rightarrow$ 0x04
REF_RnL	0x04 $\rightarrow$ 0x08

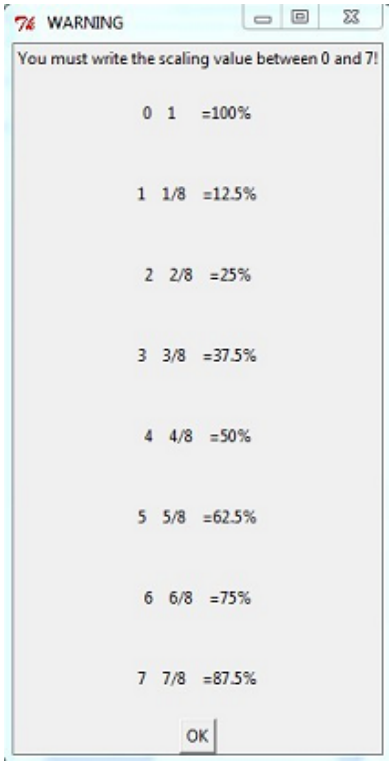


Abbildung 3.4: Überwachung für einige Koeffizienten

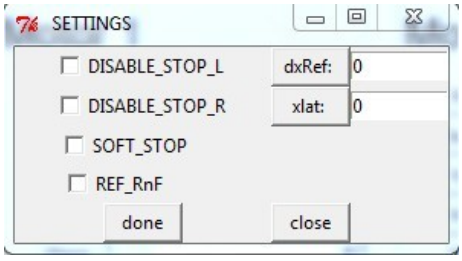


Abbildung 3.5: GUI von der Funktion settings

4 Python

4.1 Grundlage von Python

Python ist eine dynamische und objektorientierte Skriptsprache. Der Unterschied zwischen einer Programmier- und Skriptsprache liegt im sogenannten Compiler. Ähnlich wie Java verfügt Python über einen Compiler, der aus dem Quelltext eine Kompilation, den sogenannten Byte-Code erzeugt. Dieser Byte-Code wird dann in einer virtuellen Maschine, dem Python-Interpreter, ausgeführt.
Für viele Betriebssystemen (Unix/Linux, Windows, MacOS u.s.w) ist Python verfügbar und kostenlos herunterzuladen. Die größten Vorteile von Python sind die Flexibilität und Erweiterbarkeit. Deswegen kann Python beispielsweise als Programmiersprache für die kleinen und großen Applikationen und auch als serverseitige Programmiersprache im Internet verwendet werden. Viele Erweiterungen aus den verschiedensten Bereichen (GUI, Netzwerke, Datenbanken, Graphik, Web, u.s.w) haben Python zum Mittel der Wahl bei vielen Open-Source-Projekten und in den namenhaften Unternehmen und Organisationen gemacht. Python kann schneller erfasst werden und muss vergleichsweise weniger Aufwand als die anderen Sprachen für das gleiche Ergebnis aufbringen. Mit Python lässt sich eine Idee in sehr kurzer Zeit zu einem ersten lauffähig Programm fortentwickeln.
Die Python-Programme kann man leicht lernen und lesen, weil es eine einfache Syntax besitzt. Der Programmtext ist einfach zu schreiben. Die Pythons Syntax hat keine Klammern für die Befehlsblöcke, keine Anweisungsendzeichen sondern immer die Einrückungen.
Man kann die Klasse ähnlich wie in Java zuerst erzeugen, wenn man eine Funktion realisieren möchte. Man kann auch einfach eine Methode ohne Klasse erstellen, aber man muss die globalen Variablen beobachten. Man kann sogar die Methode in der Main-Funktion schreiben, wenn die Methode keine rekursive oder iterative Funktion realisiert.

4.2 Einige Module von Python

Wenn man in Python die vorhandenen Funktionen aufrufen möchte, sollen die Module, welche die Funktionen enthalten, am Anfang des Codes importiert werden. Man kann das Importieren mithilfe der import-Anweisung realisieren. Es gibt zwei Weisen von der Verwendung der import-Anweisung:

```
1 import ctypes
```

```
1 from ctypes import *
```

Es gibt einen kleinen Unterschied zwischen den beiden Weisen. *Import* lädt ein Modul im eigenen Namensraum, so dass man den Namen, auf den man sich aus den importierte Modulen bezogen hat, durch einen Punkt vor der Referenz verwendet. *From* lädt ein

Python-Modul in dem aktuellen Namensraum, damit es nicht den Name nochmal benutzen muss, wenn man sich auf den Namen beziehen möchte.
In meiner Arbeit muss man auf der USB Motion 3xII Platine mit DLL und auf dem Modell P3 mit ActiveX 4.3.1 programmieren.

4.2.1 Dynamisch ladbare Bibliotheken - *ctypes*

Mit dem Modul *ctypes* ist es möglich, Funktionen einer sogenannten dynamischen ladbaren Bibliothek, aufzurufen. Zu solchen dynamischen Bibliotheken zählen beispielsweise DLL-Dateien 4.3.1 unter Windows und Linux.
Das Modul *ctypes* exportiert *cdll* und unter Windows-Betriebssystem auch *windll* und *oledll*. Diese drei können für das Importieren der DLLs benutzt werden. Die durch *cdll* geladene Exportfunktionen ermöglichen nach der *cdecl* Aufruf-Konvention. Die durch *windll* geladene Exportfunktionen folgen nach der *stdcall* Aufruf-Konvention. Die durch *oledll* geladene Exportfunktionen benutzen die gleiche *stdcall* Aufruf-Konvention wie *windll* aber ein Windows' HRESULT-Fehlercode ist für die Rückgabe der Funktion angenommen. Wenn die Funktion nicht erfolgreich aufgerufen wird, wird der Fehlercode automatisch als ein Windows' Fehlerausnahme benutzt.

Listing 4.1: Ein Beispiel von USB Motion 3xII

```
1  from ctypes import *
2  filename="USBM3x32.dll"
3  f=windll.LoadLibrary(filename)
4
5  f.USBMCCreate(0,0,c_char_p('USB_MyApp_ID'))    # um die
       Applikation aufzurufen
6
7  PData=c_void_p()
8  PIICData=c_void_p()
9  PIICScan=c_void_p()
10 f.USBMCinit(byref(PData),byref(PIICData),byref(PIICScan))  #
       Diese Funktion muss nach der Funktion USBMCCreate aufgerufen
       werden.
11
12 number=f.DeviceCount()
13 if number==0:
14     print('No Device is connected!')
15 else:
16     print('Connected!')
17     f.GetDevName.restype=c_char_p
18     print(f.GetDevName(0)) # Ergebnis:USB Motion 3x
           16/2010/05/02
19     f.M3xDLLVersion.restype=c_char_p
20     print(f.M3xDLLVersion()) # Ergebnis:104/05/2007
21     Switch=c_byte()
22     Status=c_byte()
23     f.GetSwitch(byref(Status),byref(Switch))
```

```
24
25 f.USBMCDestroy()  # um die Applikation auszuschalten
```

Wenn man das Modul *ctypes* importiert, soll man während der Benutzung die Datentypen beachten. Ist das Datentyp Referenzargument auf C Sprache, wird *byref* vor dem Argument geschrieben und das Argument muss unbedingt das entsprechende Modul *ctypes* Typ sein. Die Datentypen der Sprache C stehen in der Tabelle 4.1 [3] im Vergleich zu den Datentypen des Moduls *ctypes*.

Tabelle 4.1: Datentypen zwischen Modul ctypes und C

C Typen	**Modul ctypes Typen**	**Python**
char	c_char	str
wchar_t	c_wchar	unicode
char	c_byte	int, long
unsigned char	c_ubyte	int, long
short	c_short	int, long
unsigned short	c_ushort	int, long
int	c_int	int, long
unsigned int	c_uint	int, long
long	c_long	int, long
unsigned long	c_ulong	int, long
float	c_float	int, long
double	c_double	int, long
char *	c_char_p	str, None
wchar_t *	c_wchar_p	unicode, None
void *	c_void_p	int, long, None

4.2.2 GUI-Programmierung - *Tkinter*

Das Modul *Tkinter* ermöglicht GUI, deswegen ist es das am meisten verwendete Modul. Es instanziiert die Tcl/Tk Schnittstelle, die eine plattformübergreifende GUI-Schnittstelle ist. Die Widgets (*Frame*, *Label*, *Button*, *Entry* u.s.w) sind allgemein Teile von GUI. Mit den Widgets macht man das Aussehen von GUI besser und deutlich. Um das Widget anzuzeigen, muss der geometrische Manager (*pack*, *place* und *grid*) aufgerufen werden. Man kann durch einen Befehl die Schriftgröße, Schriftart und sogar die Schriftfarbe wechseln. Der Text ist mithilfe des *Label* Widgets dargestellt. Das *Button* Widget wird benutzt, um auf die Benutzerinteraktion zu reagieren. D.h ein Button reagiert, wenn

man darauf drückt. Das *Entry* Widget stellt ein Eingabefeld dar. Man kann mit dem Befehl *get*() den Eingabeinhalt abfragen und mit *set*() den Inhalt einfügen. Alle Widgets können durch das *Frame* Widget unterteilen werden, damit der GUI Benutzer den Zusammenhang der Funktionen besser erkennen kann.
Ein besonderes Widget, das ein Fenster darstellt und in dem die Zeichnungen erstellt werden können, nennt man *Canvas*. Ich benutze das um die Korrektheit und Genauigkeit der Schrittmotoransteuerung zu testen. Nach jedem Schritt des Motors wird ein Punkt und eine Linie zwischen zwei Punkten gezeichnet, dann kann man intuitiv erkennen, ob die Ansteuerung richtig läuft.
Bei jedem Widget muss ein geometrischer Manager aufgerufen werden.
Jedes übergeordneten Widget soll immer denselben Manager aufrufen, sonst wird das Programm nicht kompiliert. Mit dem geometrischen Manager *pack*() wird ein Widget nach Seite angeordnet. Mit *place*() kann man ein Widget an der genauen x/y-Position ausrichten. Mit dem geometrischen Manager *grid*() wird in Reihe und Spalte geordnet, wie in einer Tabelle. Während der Programmierung muss man auch beobachten, dass zwei Widgets nicht an der gleichen Stelle liegen dürfen, sonst wird das erste Widget überschrieben.
Das *Tkinter* Modul enthält noch eine Funktion, welche *update*() Funktion heißt. Diese Funktion ist für die zeitliche Aktualisierung zu realisieren. Wenn man die Funktion benutzen würde, müsste man vor dem *update*() das aktualisierte Widget aufzeigen, dann wird das aufgezeigte Widget in dem GUI aktualisiert. Was man nicht vergessen darf ist, dass der geometrische Manager nochmals aufzurufen werden muss.

Listing 4.2: Countdown-Beispiel von GUI mit einigen Widgets und update()

```
1  from Tkinter import *
2  import time
3  def funk(a):
4      a=int(a)
5      while a>=0:
6          Label(root,text=str(a)).grid(row=0,column=0)
7          root.update()
8          time.sleep(1) #Die While-Schleife muss jedes mal eine
               Sekunde Pause haben.
9          a=a-1
10     Label(root,text='Stop').grid(row=0,column=0)
11     root.update()
12
13 root=Tk(className=' Time Countdown')
14 t=StringVar('')
15 Label(root,text=t.get()).grid(row=0,column=0)
16 f=Frame(root,relief=RIDGE,borderwidth=2)
17 f.grid(row=1,column=0,columnspan=1)
18 Label(f,text='time').grid(row=0,column=0)
19 Entry(f,text=t).grid(row=0,column=1)
20 f1=Frame(root,relief=RAISED,borderwidth=2) #Es gibt 4
       verschiedene Frame-Typen.
```

```
21 f1.grid(row=2,column=0,columnspan=1)
22 Button(f1,text='Start',command=lambda:funk(t.get())).grid(row
       =0,column=0)
23 Button(f1,text='Close',command=root.destroy).grid(row=0,column
       =1) #Man schaltet die GUI aus.
24 root.mainloop()
```

4.2.3 Mathematische Modul - *math*

Das Modul ist eine Standardbibliothek und bietet alle mathematische Funktionen und Konstanten. Aber das Modul kann die Berechnung des komplexen Zahlenraumes nicht ermöglichen, muss man das Modul *cmath* verwenden. Das heißt, eine in *math* enthaltene Funktion akzeptiert niemals einen komplexen Parameter und gibt niemals ein komplexes Ergebnis zurück [3]. Einige Funktionen vom Modul *math* habe ich in der Tabelle 4.2 mit den Beispielen geschrieben.

Tabelle 4.2: Funktionen des Moduls math

Funktion	**Erklärung**	**Beispiel**
fabs(x)	Absolutwert von x	$fabs(5.1) = 5.1$
acos(x)	Arkuskosinus von x, das Ergebnis ist im Bogenmaß	$acos(1.0) = 0.0$
cos(x)	Kosinus von x, das Ergebnis ist im Bogenmaß	$cos(0.0) = 1.0$
ceil(x)	Runden von x, das Ergebnis ist die minimale ganze Zahl, die nicht kleiner als x ist.	$ceil(9.2) = 10.0$
floor(x)	Runden von x, das Ergebnis ist die maximale ganze Zahl, die nicht großer als x ist.	$floor(9.2) = 9.0$
exp(x)	Ergebnis von e^x	$exp(1.0) = 2.71828$
pow(x,y)	Ergebnis von x^y	$pow(5.0, 2.0) = 25.0$
log10(x)	Logarithmus von x, Basis ist 10	$log10(100.0) = 2.0$
fmod(x,y)	Rest von x/y	$fmod(9.8, 4.0) = 1.8$
sqrt(x)	Quadratwurzel von x	$sqrt(900.0) = 30.0$

4.2.4 Elementare Zeitfunktionen - *time*

Durch das Modul *time* kann man den aktuellen Zeitpunkt in verschiedene Neuntupel umwandeln (z.B von Sekunde bis Jahr). Das Modul enthält eine Zeitfunktion, die $sleep(n)$ heißt. Die Funktion unterbricht den aktuellen Prozess für n Sekunden. Ein explizites

Beispiel wird gegeben. Das ist ein Code für ein Zeit-Countdown. Man kann am Anfang die Countdown-Zeit einfügen und dann drückt man die Taste *Start*. Im Oberteil des Fensters wird die Zahl pro Sekunde um eins reduziert. Zum Schluss wird *Stop* angezeigt.

4.2.5 Komfortable Datumsfunktionen - *datetime*

Das Modul *datetime* ist im Vergleich zum *time*-Modul wesentlich abstrakter und durch seine eigenen Zeit- und Datumstypen auch angenehmer zu benutzen [3]. Dieses Modul bietet die Methode um die Zeitdauer zwischen zwei Zeitpunkte zu berechnen. In meiner Arbeit brauche ich das Modul während des Speichermodels. Weil man den Messwert von jedem Kanal in der unterschiedlichen Speicherzeit auf dem Computer speichert, soll man die genaue Zeitdauer berechnen, damit jeder Speicherpunkt eines Kanals gegen die Zeitdauer unterschieden werden kann.

4.3 *Pywin*32

*Pywin*32 ist eine Python-Unterstützung für Windows aber nicht für allgemeine Python verfügbar. Wenn man *Pywin*32 benutzen will, muss man manchmal *Pywin*32 zuerst von der Homepage `http://sourceforge.net/projects/pywin32/` herunterladen und in Python installieren, dann wird das Modul *win*32*com* vor die Anwendung importiert.

4.3.1 Unterschied zwischen DLL und ActiveX

DLL ist die Abkürzung von Dynamic Link Library auf Englisch. DLL ist eine Funktionen enthaltende Bibliothek. Der Programmierer kann die DLL-Datei mit dem Code in den gleichen Ordner stecken, damit die DLL und der Code verbunden werden können, wenn der Code läuft. Die dynamische Verbindung bedeutet, dass der Code die Funktionen von DLL aufrufen kann, obwohl der Code die DLL-Datei nicht enthält.
ActiveX bietet eine Standard-Kontrolle für den Programmierer unter dem Windows-Betriebssystem in VB, VC oder VF zu programmieren. Jede Standard-Kontrolle hat ihre eigenen Eigenschaften und Methoden. Der Programmierer soll nur für jede Methode einen Code schreiben. Aber die Programmierung, die nur mit der Standard-Kontrolle ist, kann der Programmierer manchmal nicht erfüllen, dann entsteht die ActiveX-Kontrolle. Sie ist eine Datei mit der Dateiendung *.dll* oder *.ocx*. Während der Programmierung muss man die ActiveX-Kontrolle zuerst registrieren, dann wird sie gleich wie die Standard-Kontrolle verwendet. Bei meiner Arbeit befindet sich die ActiveX Bibliothek in DLL. Man kann diese Bibliothek mit dem Namen $VMMP3Control.dll$ in der default installierenden Bibliothek finden.
$VMMP3Control.dll$ setzt die Schnittstelle von VMMP3Controller aus.[7]
Der größte Unterschied zwischen DLL und ActiveX-Kontrolle ist, dass sich die ActiveX-Kontrolle am Anfang anmelden muss, sonst ist die Kontrolle unerkennbar und unverwendbar.

4.3.2 *win*32*com*

*Pywin*32 ist eine Erweiterung von dem *win*32 in Python, um das COM (Component Object Model)-Objekt in Betrieb zu nehmen. Die COM-Komponenten sind eine Menge von den Schnittstellen aber nicht eine Menge von den Funktionen. In *Pywin*32 darf man

die Anwendungen von *win*32 in Python gleich wie in VB entwickeln. Mit dem Model *win*32*com* kann man viel APIs z.B die office API aufrufen. Man muss zuerst das Modul *win*32*com* importieren und dann ein COM-Objekt erstellen. Es gibt zwei Funktionen um das Objekt zu erstellen. Diese sind *Dispatch*() und *DispatchEx*(). Die *Dispatch*() Funktion versucht vor allem die *GetObject*() Funktion zu verwenden, bevor das Objekt erstellt wird. Wenn es ein laufendes Beispiel gäbe, würde es das entsprechende Objekt erhalten. Die *DispatchEx*() Funktion arbeitet um ein Objekt direkt zu erstellen.
Ich habe ein Beispiel des Modells P3 gegeben, um die Verwendung von *Pywin*32 besser zu erklären. Das Ergebnis dieser Speicherung steht in der Abbildung 4.1. Es handelt sich dabei um eine TXT-Datei. Man kann den Namen des Geräts, die Seriennummer des Geräts, die kreierte Datum der Datei und die Messwerteinheit ablesen. Die Speicherzeiten sind 7 Sekunden für den Kanal 1, 3 Sekunden für den Kanal 2 und 9 Sekunden für den Kanal 3. Der Kanal 4 wird deaktiviert. Die Speicherung dauert im automatischen Modell 18 Sekunden. Die Messwerteinheit ist $\mu\epsilon$ und die Messwerte werden in der Multimediakarte gespeichert.

Listing 4.3: Ein Beispiel vom Modell P3 um die Messwerte zu speichern

```
1  import win32com
2  from win32com import client
3  import time
4  P3=win32com.client.Dispatch('VMMP3Control.VMMP3Controller')
5  P3.DeviceOpen=True #Der Device wird eingeschaltet.
6  P3.RecordingMode=2 #Das Speichermodell ist automatische Modell.
7  P3.SetRecordingInterval(1,7) #Die Speicherzeit vom Kanal 1 ist
       7 Sekunde.
8  P3.SetRecordingInterval(2,3) #Die Speicherzeit vom Kanal 2 ist
       3 Sekunde.
9  P3.SetRecordingInterval(3,9) #Die Speicherzeit vom Kanal 3 ist
       9 Sekunde.
10 P3.MediaCardOpen=True #Die Multimediakarte wird eingeschaltet.
11 P3.RecordingActive=True #Das Modell P3 arbeitet im
       automatischen Modell um die Werte zu speichern.
12 P3.ScansRecorded #Um die Scannensanzahl auf dem Bildschirm
       anzuzeigen.
13 time.sleep(18) #Die Speicherung dauert 18 Sekunde.
14 P3.RecordingActive=False #Die Spericherung des Modells P3
       stoppt.
15 P3.MediaCardOpen=False #Die Multimediakarte wird ausgeschaltet.
16 P3.DeviceOpen=False #Der Device wird ausgeschaltet.
```

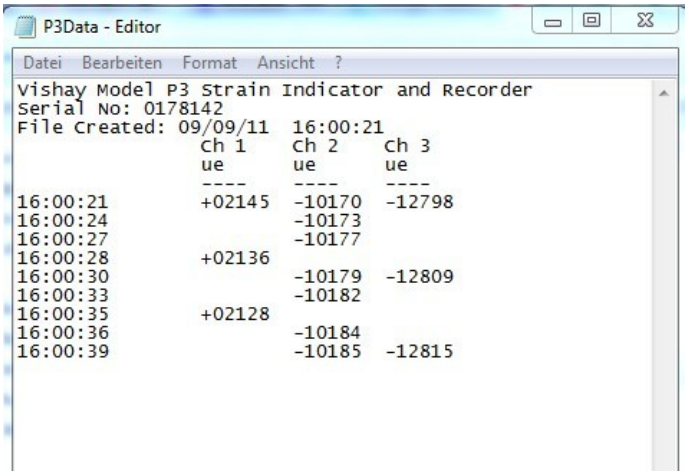


Abbildung 4.1: Ergebnis des Beispiels

5 Implementierung

Die Abbildung 5.1 zeigt die Grund-Idee von meiner Arbeit. Über ein USB-Messgerät Modell P3 sollen die Widerstandswerte der Dehnungsmessstreifen auf dem PC eingelesen und als Tabelle (CVS-Datei) dargestellt und abgespeichert werden. Weiterhin sollen 3

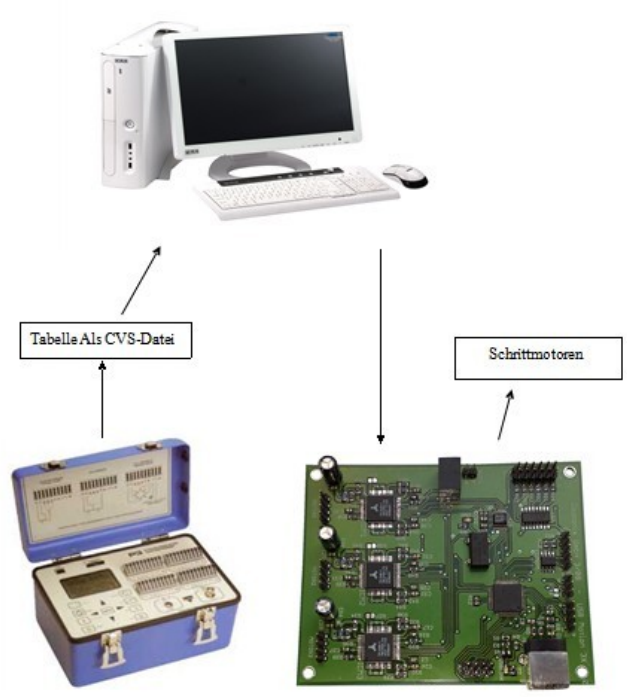


Abbildung 5.1: Design-Idee

Schrittmotoren mit dem Programm von der Platine USB Motion 3xII angesteuert werden. Die Schrittmotoren steuern einen Roboterarm. Hierbei sollen die Schrittmotoren mit den Widerstandswerten gesteuert werden, z.B einer Kreisbahn fahren.
Was ich gemacht habe ist die Ansteuerung von Modell P3 und USB Motion 3xII mit Python. Aber das Kontrollieren der Schrittmotoren, die abhängig von den Widerstandswerten sind, habe ich nicht gemacht.

5.1 USB Motion 3xII

5.1.1 GUI von USB Motion 3xII

Ich habe ein GUI mit Python für fast alle Funktionen von der Platine gemacht und in den 6 Dateien aufgetrennt. Einer davon ist eine Hauptdatei und ich nenne sie $MAIN$. Sie funktioniert um die Einrichtung des GUI zu ermöglichen. Ich hoffe, dass es intuitiv

und deutlich ist, wenn man das GUI von der Schrittmotoransteuerung benutzen möchte. Es gibt auch andere Subdateien: *controlfunktion*, *gp*, *numbercheck*, *rfunktion* und *SMfunktionen*.
Man kann durch die Subdatei *SMfunktionen* die maximale und minimale Geschwindigkeiten, die Zielposition, die Beschleunigung, und alle aktuellen Zustände von den 3 Schrittmotoren erkenne und wieder einstellen. Die Mikroschritte und die Arbeitsmodi sind auch veränderbar. Man kann auch die Interrupt-Flaggen und Interrupt-Masken definieren. Die Platine arbeitet mit einer Spannungsversorgung von $16V$. In der Abbildung

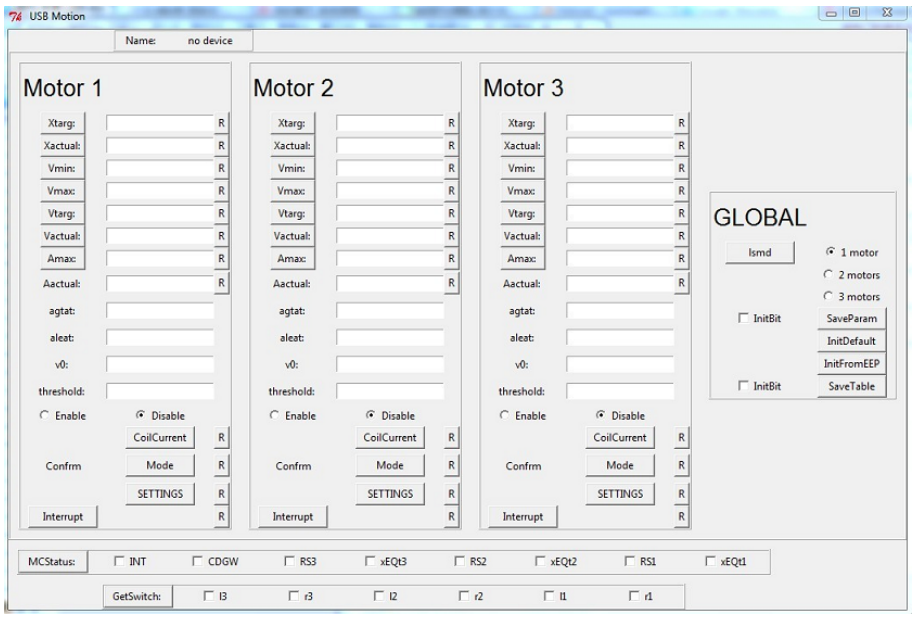


Abbildung 5.2: GUI von USB Motion 3xII

5.2 wird der aktuelle Zustand durch *MCStatus* angezeigt. Die Bedeutungen von jedem Registerbit kann man von der Tabelle 3.3 ablesen. Durch die Anzeige von *GetSwitch* kann man wissen, welchen Status der Schalter besitzt. Die Beschreibungen jeden Bit stehen in der Tabelle 5.1. In der Abbildung 3.2 kann man die Nummer 6 verwendet, um den Status von dem Schalter zu ändern. Mit der Hilfe von einem Jumper kann man durch die Anzeige von *GetSwitch* erkennen, wenn man den Jumper zwischen Pin 11 und Pin 12 einstellt, dreht der Schrittmotor 1 nach links. Das Bit $l1$ soll auf 1 gesetzt werden. Das Codebeispiel kann man in dem Kapitel 4.2.1 anschauen. Dieser Teil von Dateien heißt *controlfunktion*.
Der Buchstabe R in der Abbildung 5.2 zeigt, ob die Funktion jetzt eine Read-Funktion ist. Wenn man die R Taste drückt, wird diese R rot. Das bedeutet, dass die Funktion jetzt als eine Read-Funktion arbeitet, ansonsten ist die Funktion eine Write-Funktion. Die Einrichtung der Taste in der Abbildung 5.2 habe ich durch die *rfunktion* Datei realisiert. Die R Taste ist ein Kontrollschalter, deswegen muss sie durch einen globalen Parameter ermöglicht werden. Ich habe eine PY-Datei erstellt und lege alle globalen Parameter hinein. Sie heißt *gp*. In allen Dateien soll die *qp*-Datei vor allem importiert werden, weil die Wahrscheinlichkeit in jeder Datei, die globale Parameter zu ändern besteht.

Tabelle 5.1: Status des Schalters

Status	**Bit-Abbildung**	**Beschreibung**
r1	0x01	Motor 1, rechter Schalter
l1	0x02	Motor 1, linker Schalter
r2	0x04	Motor 2, rechter Schalter
l2	0x08	Motor 2, linker Schalter
r3	0x10	Motor 3, rechter Schalter
l3	0x20	Motor 3, linker Schalter

In dem Kapitel 3.2 habe ich schon eingeführt, dass es noch einige Koeffizienten gibt, die man vor allem einstellen soll. Solche Koeffizienten sind am längsten 3 Bits, dann habe ich ein Programm gemacht, um die Länge zu sichern, damit es nicht länger als 3 Bits ist. Die Subdatei ist *numbercheck* und muss nur vor dem Code der $SMfunktionen$ importiert werden.
In der Abbildung 5.2 kann man die Anwendungen von 3 Motoren deutlich ablesen und entstellen. Ich lege alle Buttons, die die Anwendungen enthalten, in der Tabelle 5.2. Je-

Tabelle 5.2: Details von GUI-Programm

Button	**Abbildung**
CoilCurrent	3.2.2
Mode	3.3(a)
SETTINGS	3.2.2
Interrupt	3.3(b)

der Button kann ein entsprechendes GUI erstellen.

5.1.2 Ein Beispiel von der Platine

Ich habe noch einen zusätzlichen Code geschrieben, um die 3 Schrittmotoren sicher anzusteuern. Ein Bohrloch wird gemacht. Den Radius des Bohrlochs kann man ändern, danach kann man ein großes Bohrloch von vielen kleinen Bohrlöchern mit gleichem Radius aufbauen. Die beiden Radien des Bohrlochs kann man im Code verändern. In dem Code habe ich eine while-Schleife definiert, sodass man warten muss, wenn der $xtarg$ von $SetXtarget$ gleich wie der $Xtarg$ von $GetXtarget$ ist. D.h der Motor erreicht die Zielposition. Wenn man eine weite Zielposition einfügt, dauert der Abgleich dieser

Gleichung etwas länger wegen der Bewegung des Motors. Ich erstelle ein Bild mit einer Größe von $800 * 600$ und die Motoren beginnen ab dem Punkt $(400, 300)$ an zu arbeiten. Das heißt, dass der Ursprung des großen Bohrlochs $(400, 300)$ ist. Für den Hintergrund habe ich schwarze Farbe gewählt und jedes Bohrloch habe ich mit weißer Farbe realisiert, dann kann man deutlich erkennen, mit welcher Spur die Schrittmotoren laufen sollten. Die 3 Schrittmotoren repräsentieren 3 Achsen (x, y, z). Die x und y Achsen legen einen Punkt auf dem Hintergrund fest und die z Achse beschreibt die Tiefe des Bohrlochs.

Listing 5.1: Ansteuerung der Schrittmotoren

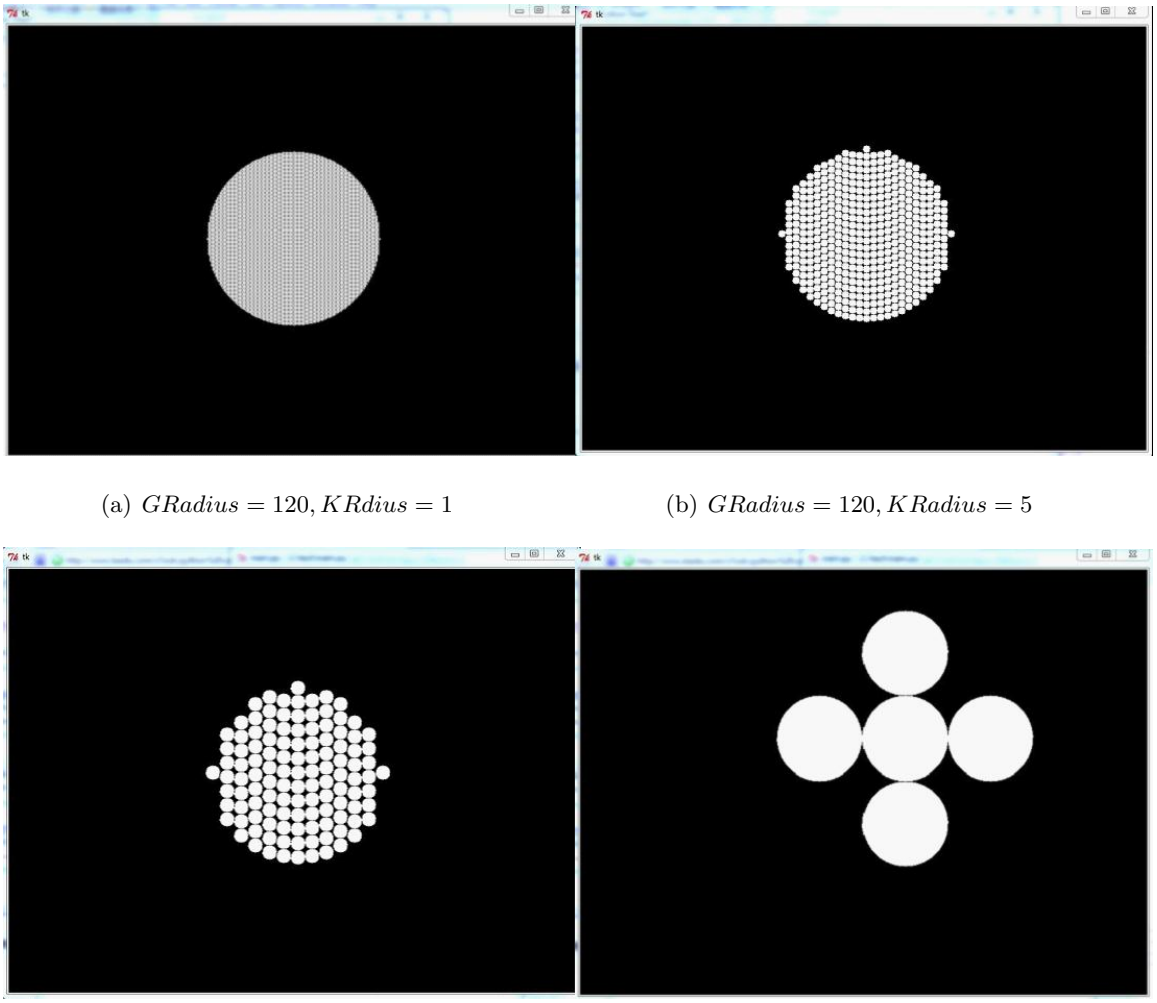
```
from ctypes import *
from math import *
from Tkinter import *
filename="USBM3x32.dll"
f=windll.LoadLibrary(filename)

Motor1=c_byte(0)
Motor2=c_byte(1)
Motor3=c_byte(2)
GRadius=c_int(120) # die Lange ist wechselbar
Radius=c_int(5) # die Lange ist wechselbar
Tiefe=c_int(13) # die Lange ist wechselbar
Status=c_byte()

def kreis(rx,t,a,b):
    X2=a
    Y2=b
    ra=rx.value
    rl=0-ra
    rr=ra+1
    for x in range (rl,rr):
        yrl=0-sqrt(ra*ra-x*x)
        yrr=sqrt(ra*ra-x*x)+1
        for y in range (int(yrl),int(yrr)):
            f.SetXtarget(Motor1,c_int(x),byref(Status))
            f.SetXtarget(Motor2,c_int(y),byref(Status))
            f.SetXtarget(Motor3,t,byref(Status))
            Xact1=c_int()
            Xact2=c_int()
            Xact3=c_int()
            while Xact1.value!=x and Xact2.value!=y and Xact3.
                value!=Tiefe.value:
                f.GetXactual(Motor1,byref(Status),byref(Xact1))
                f.GetXactual(Motor2,byref(Status),byref(Xact2))
                f.GetXactual(Motor3,byref(Status),byref(Xact3))
            X1=int(x+a)
            Y1=int(b-y)
            cv.create_line(((X2,Y2),(X1,Y1)),fill="white")
```

```
38                  X2=X1
39                  Y2=Y1
40                  f.SetXtarget(Motor3,c_int(0),byref(Status))
41                  Xact4=c_int(10)
42                  while Xact4.value!=0:
43                      f.GetXactual(Motor3,byref(Status),byref(Xact4))
44
45  MessageID=f.USBMCCreate(0,0,c_char_p('USB_MyApp_ID'))
46  if MessageID<=0:
47      print('can_not_be_create')
48
49  PData=c_void_p()
50  PIICData=c_void_p()
51  PIICScan=c_void_p()
52  f.USBMCinit(byref(PData),byref(PIICData),byref(PIICScan))
53
54  root=Tk()
55  cv=Canvas(root,bg="black",width=800, height=600)
56  GR=GRadius.value
57  X=400
58  Y=300
59  xl=0-GR
60  xr=GR+1
61  for m in range (xl,xr,2*Radius.value):
62      yl=0-sqrt(GR*GR-m*m)
63      yr=sqrt(GR*GR-m*m)+1
64      for n in range (int(yl),int(yr),2*Radius.value):
65          kreis(Radius,Tiefe,m+Radius.value+X,Y-(n+Radius.value))
66  cv.pack()
67  root.mainloop()
68
69  f.USBMCDestroy()
```

5.2 Modells P3

5.2.1 GUI vom Modell P3

Ich habe ein Haupt-GUI mit anderen Sub-GUIs erstellt und mit dem werden die Werte, die das Gerät gemessen hat, auf dem Label von jeden Kanal angezeigt. Die Ausrichtung vom Haupt-GUI habe ich in der Datei *t* definiert und die Sub-GUIs sind in *funktion* zu realisieren. Die Datei *subfunktion* zeigt alle Anwendungen von dem Modell P3, um die Einstellung abzufragen und zu korrigieren.
Das GUI enthält all die Informationen, die im Manual [8] liegen. Man kann mit dem GUI die Kanäle auswählen. Die Brückenart für jeden gewählten Kanal wird angezeigt und geändert. Den Koeffizient von dem K-Faktor und der Skalierung, die Einheit der Messwerte kann man auch mit dem GUI einlesen, zeigen und natürlich auch einstellen. Die Brückenabgleich und die Nebenschluss-Kalibrierung können auch durch das GUI



(a) $GRadius = 120, KRdius = 1$ (b) $GRadius = 120, KRadius = 5$

(c) $GRadius = 120, KRdius = 10$ (d) $GRadius = 120, KRadius = 60$

Abbildung 5.3: Ergebnisse des Beispiels

angesteuert werden. Die Details jeder Taste stehen in der Tabelle5.3.
Das GUI vom Hauptmenü wird in der Abbildung 5.2.1 gezeigt. Man kann die Taste *Record* vom GUI drücken, um die angezeigte Werte auf dem PC als CVS-Datei oder einfach in der Multimediakarte als TXT-Datei abzuspeichern. Es gibt zwei Methoden für das Speichern. Die sind das manuelle Speichern und das automatische Speichern. Beim manuellen Speichern soll man für jedes Stück vom Speichern die Taste *Record* von der Softkey-Tastatur drücken. Beim automatische Speichern muss man zuerst die Daten der gewählten Kanäle in den spezifizierten Intervallen abspeichern, um die Speicherzeit einzugeben. Jeder Kanal kann von einmal pro Sekunde bis einmal pro Stunde (3600 Sekunden) abgefragt und unterschiedliche Zahlen ausgegeben werden. Man kann auch die Zeit am Anfang des Speicherns bis am Ende des Speicherns ausgeben. Mit anderen Worten wird die Speicherdauer ausgegeben.

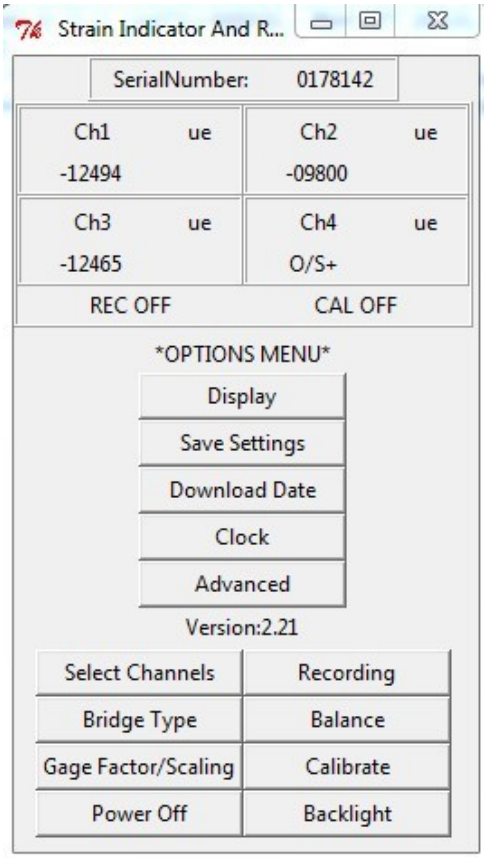


Abbildung 5.4: GUI vom Modell P3

5.2.2 Speicherung auf den PC

Um die Daten mit automatischer Methode auf dem PC zu speichern habe ich zwei Methoden gefunden. Eine davon ist mithilfe der Multimediakarte, die andere ist mit dem Befehl in Python. Ich möchte für das GUI eine Funktion erstellen, sodass bei jeder Speicherung die Labels vom Haupt-GUI aktualisiert werden müssen. Die Speicheranzahl und die Speichermethode müssen gemeinsam im Unterteil des Labels angezeigt werden. Zum Schluss der Speicherung wird die Speichermethode auf den Zustand - No Recording umgeschaltet.
Im Manual [7] stehen alle Methoden und Eigenschaften von der ActiveX-Kontrolle. [7] bietet zwei Methoden: $ReadMMCDirectoryEntry()$ und $ReadMMCDataFilePacket()$. Die Methode $ReadMMCDirectoryEntry()$ ist eine Read-Funktion, um die nächste Bibliothek aus dem Modell P3 einzulesen. Die Funktion läuft solange bis der Rückgabewert gleich Null ist. Der Rückgabewert wird mit den Informationen gegeben, damit er den Dateiname, das erstellte Datum und Zeit sogar die Größe enthält. Bevor man die Methode $ReadMMCDataFilePacket()$ aufruft, muss eine Datei geöffnet sein. Es gibt eine Methode $OpenMMCDataFile$, um eine gewählte Datei aufzuschalten. Die Eingabe von $OpenMMCDataFile$ ist der Dateiname. Wenn die gewählte Datei geöffnet ist, ruft man die Methode $ReadMMCDataFilePacket()$ mit der Dateigröße als eine Eingabe auf. Die Rückgabewerte sollen die Inhalte der Datei, jede Zeile von 1 bis 26 Bytes sein. Ich habe

Tabelle 5.3: Details vom Modell P3 des GUI-Programms

Taste	**Abbildung**
Display	2.8(a)
Clock	2.8(b)
Advanced	2.8(c)
Select Channels	2.6(a)
Bridge Type	2.7(a)
Gage Factor/Scaling	2.6(d)
Recording	2.7(b)
Balance	2.6(c)
Calibrate	2.6(b)

die oben genannte Folge vielmal probiert, aber ich habe das Ergebnis immer noch nicht rausbekommen, welches ich vorher erwartet habe. Ich kann nicht folgern warum, die Rückgabewerte von der Methode $ReadMMCDataFilePacket()$ unerwarteterweise als String ausgegeben werden. Ich weiß nicht, wie die Probleme zustande kommen. Ich probierte viel aus um das Problem zu lösen. Zuerst habe ich gedacht, dass die Problem vielleicht wegen der Multimediakarte war, weil die Methode $ReadMMCDirectoryEntry()$ nicht immer funktionierte, d.h manchmal konnte sie nicht alle Bibliotheken zurückgeben, manchmal war die Karte unlesbar, manchmal gab es den Fehler bei dem Aufschalten der Karte. Ich glaubte, dass der Kontakt zwischen der Multimediakarte und dem Einsteckschlitz vom Modell P3 nicht voll funktionsfähig war. Um diese Probleme zu lösen habe ich auf zwei verschiedenen Weisen implementiert. Eine Weise ist mithilfe der Methode $EraseMMC$, damit wird die Dateianzahl so weniger wie möglich und damit wird die Wahrscheinlichkeit des Fehlers während des Aufrufens der Methode $ReadMMCDirectoryEntry()$ reduziert. Die andere Weise ist es mithilfe der anderen Multimediakarte zu verwenden. Sie funktionierte auch nicht, wie ich mir erhoffte. Die Methode ist, dass man die Daten von der Multimediakarte auf dem PC speichert. Ich habe aber letztendlich diese Methode verzichtet.
Die zweite Methode habe ich in Python geschaffen. Ich habe zuerst durch die Methode *open*() eine leere CVS-Datei auf dem Desktop erstellt, dann habe ich einige Informationen (z.B den Namen des Modells P3, die Seriennummer, das erstellte Datum, die Messwerteinheit), die gleich wie die in der Multimediakarte gespeicherte Dateien sind, am Anfang der Datei gespeichert. Die Messwerte und der Zeitpunkt wurden nach der ausgerechneten Zeitfolge gespeichert, dann wurde ein kleiner Algorithmus für die Zeitfolge gemacht. Der Algorithmus basiert auf die Speicherzeiten vom jedem aktivierten Kanal und die Speicherdauer. Ich habe ein Beispiel gemacht, um den Arbeitsprozess des Algorithmus besser zu erklären. Die Speicherung dauerte 10 Sekunden und die Speicherzeiten für Kanal 1, 2, 3 sind $1, 2, 4$ Sekunden. Am Anfang der Speicherung sollten

die aktuellen durch die Methode *CurrentReading* erkannten Messwerte von allen aktivierten Kanälen gemeinsam in der Datei erstellten Zeit gespeichert werden, obwohl die Speicherzeiten unterschiedlich waren. Bei der zweiten Speicherung wurden die Messwerte nach der Speicherzeitfolge aus dem Algorithmus gespeichert. Man muss diese Situation beobachten, wenn in dem gleichen Zeitpunkt mehr als einen Messwert gespeichert wird. Im Beispiel habe ich die Aktualisierung jeder Speicherung nicht beachtet. Ich möchte ein nicht so kompliziertes Beispiel angeben, damit der Algorithmus deutlich erklärt wird.

Listing 5.2: Ein Beispiel: um die Messwerte als eine CVS-Datei auf den PC zu speichern

```
1  import win32com
2  from win32com import client
3  import datetime
4  a1=1 #Die Speicherzeit vom Kanal 1 ist 1 Sekunde.
5  a2=2 #Die Speicherzeit vom Kanal 2 ist 2 Sekunden.
6  a3=4 #Die Speicherzeit vom Kanal 3 ist 4 Sekunden.
7  a4=0 #Der Kanal 4 ist deaktiviert.
8  P3=win32com.client.Dispatch('VMMP3Control.VMMP3Controller')
9  x=open(r'C:/Users/mingzhu/Desktop/automatische Speicherung.cvs'
       ,'a') #a bedeutet, dass die Daten in einem vorhandene Datei
       weiter gespeichert werden nicht in einer neuen Datei
       gespeichert werden.
10 x.write('Vishay Model P3 Strain Indicator and Recorder\n')
11 P3.DeviceOpen=True
12 x.write('Serial No: '+P3.SerialNo+'\n')
13 s=P3.GetTime()
14 start_time=datetime.datetime(s[0],s[1],s[2],s[3],s[4],s[5])
15 end_time=start_time+datetime.timedelta(seconds=10) # Die
       Speicherung dauert 10 Sekunden.
16 s=[s[0],s[1],s[2],s[3],s[4],s[5]]
17 if s[0]<10:
18     s[0]='0'+str(s[0])
19 elif s[0]>=10:
20     s[0]=str(s[0])
21 if s[1]<10:
22     s[1]='0'+str(s[1])
23 elif s[1]>=10:
24     s[1]=str(s[1])
25 if s[2]<10:
26     s[2]='0'+str(s[2])
27 elif s[2]>=10:
28     s[2]=str(s[2])
29 if s[3]<10:
30     s[3]='0'+str(s[3])
31 elif s[3]>=10:
32     s[3]=str(s[3])
33 if s[4]<10:
34     s[4]='0'+str(s[4])
```

```
35 elif s[4]>=10:
36     s[4]=str(s[4])
37 if s[5]<10:
38     s[5]='0'+str(s[5])
39 elif s[5]>=10:
40     s[5]=str(s[5])
41 date=s[1]+'/'+s[2]+'/'+s[0]+'   '+s[3]+':'+s[4]+':'+s[5]
42 x.write('File Created: '+date+'\n')
43 if P3.ChannelActive(1)==True:
44     x.write('                Ch 1')
45 elif P3.ChannelActive(1)==False:
46     x.write('                    ')
47 if P3.ChannelActive(2)==True:
48     x.write('      Ch 2')
49 elif P3.ChannelActive(2)==False:
50     x.write('          ')
51 if P3.ChannelActive(3)==True:
52     x.write('      Ch 3')
53 elif P3.ChannelActive(3)==False:
54     x.write('          ')
55 if P3.ChannelActive(4)==True:
56     x.write('      Ch 4\n')
57 elif P3.ChannelActive(4)==False:
58     x.write('          \n')
59 if P3.ChannelActive(1)==True:
60     x.write('                '+P3.EngUnits(1))
61 elif P3.ChannelActive(1)==False:
62     x.write('                    ')
63 if P3.ChannelActive(2)==True:
64     x.write('        '+P3.EngUnits(2))
65 elif P3.ChannelActive(2)==False:
66     x.write('          ')
67 if P3.ChannelActive(3)==True:
68     x.write('         '+P3.EngUnits(3))
69 elif P3.ChannelActive(3)==False:
70     x.write('          ')
71 if P3.ChannelActive(4)==True:
72     x.write('         '+P3.EngUnits(4)+'\n')
73 elif P3.ChannelActive(4)==False:
74     x.write('          \n')
75 if P3.ChannelActive(1)==True:
76     x.write('                ----')
77 elif P3.ChannelActive(1)==False:
78     x.write('                    ')
79 if P3.ChannelActive(2)==True:
80     x.write('      ----')
81 elif P3.ChannelActive(2)==False:
```

```
82      x.write('␣␣␣␣␣␣␣␣␣␣␣')
83  if P3.ChannelActive(3)==True:
84      x.write('␣␣␣␣␣␣-----')
85  elif P3.ChannelActive(3)==False:
86      x.write('␣␣␣␣␣␣␣␣␣␣␣')
87  if P3.ChannelActive(4)==True:
88      x.write('␣␣␣␣␣␣-----\n')
89  elif P3.ChannelActive(4)==False:
90      x.write('␣␣␣␣␣␣␣␣␣␣␣\n')
91  d=s[3]+':'+s[4]+':'+s[5]
92  x.write(d)
93  count=0
94  if P3.ChannelActive(1)==True:
95      x.write('␣␣␣␣␣␣␣'+P3.CurrentReading(1))
96      count=count+1
97  else:
98      x.write('␣␣␣␣␣␣␣␣␣␣␣')
99  if P3.ChannelActive(2)==True:
100     x.write('␣␣␣'+P3.CurrentReading(2))
101     count=count+1
102 else:
103     x.write('␣␣␣␣␣␣␣')
104 if P3.ChannelActive(3)==True:
105     x.write('␣␣␣'+P3.CurrentReading(3))
106     count=count+1
107 else:
108     x.write('␣␣␣␣␣␣␣')
109 if P3.ChannelActive(4)==True:
110     x.write('␣␣␣'+P3.CurrentReading(4)+'\n')
111     count=count+1
112 else:
113     x.write('␣␣␣␣␣␣␣\n')
114 b1=a1
115 b2=a2
116 b3=a3
117 b4=a4
118 seq=[a1,a2,a3,a4]
119 p=10
120 before_time=start_time
121 before_wait_time=0
122 while before_time<end_time:
123     seq.sort()
124     i=0
125     while seq[i]==0 and i<len(seq):
126         i=i+1
127     after_wait_time=seq[i]
128     wait_time=after_wait_time-before_wait_time
```

```
129     recording_time=before_time+datetime.timedelta(seconds=
            wait_time)
130     while before_time!=recording_time:
131         s1=P3.GetTime()
132         before_time=datetime.datetime(s1[0],s1[1],s1[2],s1[3],
                s1[4],s1[5])
133     if before_time<=end_time:
134         s1=[s1[0],s1[1],s1[2],s1[3],s1[4],s1[5]]
135         if s1[3]<10:
136             s1[3]='0'+str(s1[3])
137         elif s1[3]>=10:
138             s1[3]=str(s1[3])
139         if s1[4]<10:
140             s1[4]='0'+str(s1[4])
141         elif s1[4]>=10:
142             s1[4]=str(s1[4])
143         if s1[5]<10:
144             s1[5]='0'+str(s1[5])
145         elif s1[5]>=10:
146             s1[5]=str(s1[5])
147         d1=s1[3]+':'+s1[4]+':'+s1[5]
148         x.write(d1+'␣␣␣␣␣␣␣␣')
149         const=seq[i]
150         if const==b1:
151             before_wait_time=const
152             x.write(P3.CurrentReading(1))
153             count=count+1
154             seq[i]=const+a1
155             b1=b1+a1
156             if const!=b2:
157                 x.write('␣␣␣␣␣␣␣␣␣␣')
158                 if const!=b3:
159                     x.write('␣␣␣␣␣␣␣␣␣␣')
160                     if const!=b4:
161                         x.write('␣␣␣␣␣␣␣␣␣␣\n')
162                     else:
163                         i=i+1
164                         x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                )
165                         count=count+1
166                         seq[i]=seq[i]+a4
167                         b4=b4+a4
168                 else:
169                     i=i+1
170                     x.write('␣␣␣'+P3.CurrentReading(3))
171                     count=count+1
172                     seq[i]=seq[i]+a3
```

```
173                     b3=b3+a3
174                     if const!=b4:
175                         x.write('␣␣␣␣␣␣␣␣␣␣\n')
176                     else:
177                         i=i+1
178                         x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                )
179                         count=count+1
180                         seq[i]=seq[i]+a4
181                         b4=b4+a4
182             else:
183                 i=i+1
184                 x.write('␣␣␣'+P3.CurrentReading(2))
185                 count=count+1
186                 seq[i]=seq[i]+a2
187                 b2=b2+a2
188                 if const!=b3:
189                     x.write('␣␣␣␣␣␣␣␣␣␣')
190                     if const!=b4:
191                         x.write('␣␣␣␣␣␣␣␣␣␣\n')
192                     else:
193                         i=i+1
194                         x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                )
195                         count=count+1
196                         seq[i]=seq[i]+a4
197                         b4=b4+a4
198                 else:
199                     i=i+1
200                     x.write('␣␣␣'+P3.CurrentReading(3))
201                     count=count+1
202                     seq[i]=seq[i]+a3
203                     b3=b3+a3
204                     if const!=b4:
205                         x.write('␣␣␣␣␣␣␣␣␣␣\n')
206                     else:
207                         i=i+1
208                         x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                )
209                         count=count+1
210                         seq[i]=seq[i]+a4
211                         b4=b4+a4
212         else:
213             x.write('␣␣␣␣␣␣␣␣')
214             if const==b2:
215                 before_wait_time=const
216             x.write('␣␣␣'+P3.CurrentReading(2))
```

```
217                     count=count+1
218                     seq[i]=const+a2
219                     b2=b2+a2
220                     if const!=b3:
221                         x.write('␣␣␣␣␣␣␣␣␣␣')
222                         if const!=b4:
223                             x.write('␣␣␣␣␣␣␣␣␣␣\n')
224                         else:
225                             i=i+1
226                             x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                    )
227                             count=count+1
228                             seq[i]=seq[i]+a4
229                             b4=b4+a4
230                     else:
231                         i=i+1
232                         x.write('␣␣␣'+P3.CurrentReading(3))
233                         count=count+1
234                         seq[i]=seq[i]+a3
235                         b3=b3+a3
236                         if const!=b4:
237                             x.write('␣␣␣␣␣␣␣␣␣␣\n')
238                         else:
239                             i=i+1
240                             x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                    )
241                             count=count+1
242                             seq[i]=seq[i]+a4
243                             b4=b4+a4
244                 else:
245                     x.write('␣␣␣␣␣␣␣')
246                     if const==b3:
247                         before_wait_time=const
248                         x.write('␣␣␣'+P3.CurrentReading(3))
249                         count=count+1
250                         seq[i]=seq[i]+a3
251                         b3=b3+a3
252                         if const!=b4:
253                             x.write('␣␣␣␣␣␣␣␣␣␣\n')
254                         else:
255                             i=i+1
256                             x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                    )
257                             count=count+1
258                             seq[i]=seq[i]+a4
259                             b4=b4+a4
260                     else:
```

```
261                         x.write('␣␣␣␣␣␣␣␣')
262                         if const==b4:
263                             before_wait_time=const
264                             x.write('␣␣␣'+P3.CurrentReading(4)+'\n'
                                    )
265                             count=count+1
266                             seq[i]=seq[i]+a4
267                             b4=b4+a4
268                         else:
269                             x.write('␣␣␣␣␣␣␣␣␣␣\n')
270 x.close()
271 P3.DeviceOpen=False
```

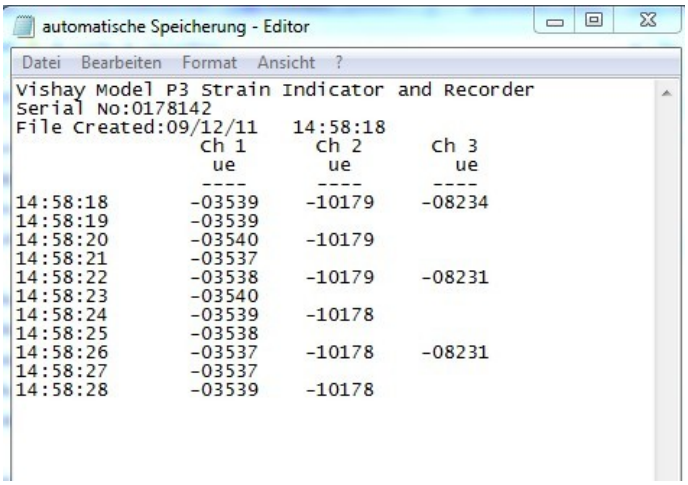


Abbildung 5.5: Ergebnis von Speicherung der CVS-Datei

6 Zusammenfassung und Fazit

Die Eigenspannung spielt eine wichtige Rolle in dem Herstellungsprozess. Bevor man die Eigenschaft der Eigenspannung untersucht, soll man die Methoden der Spannungsmessung vorstellen. Weil es unmöglich ist, um die Eigenspannungen direkt zu messen, ersetzt die Dehnungsänderung die Eigenspannung. Man benutzt die Mikrodehnung eines Widerstands statt die Dehnungsänderung von einem Bauteil, weil es viele Methoden gibt, um die unbekannten Widerstände zu berechnen. In meiner Arbeit verwende ich die Wheatstone-Brücke im Modell P3. Das Modell P3 ist die Produktion von Vishay in den USA. Die Brücke steht schon drin. Man braucht kein Werkzeug, um die Brücke aufzubauen. Die Dehnungsmessstreifen und der auf dem Dehnungsmessstreifen basierte Aufnehmer werden durch die Eingangssteckerblocken von den Kanälen verbunden. Die Messwerte können durch 2 Weisen (auf der Multimediakarte oder auf dem PC via USB) unter 2 Model (automatische oder manuelle Speicherung) gespeichert werden. Man kann mithilfe der Messwerte 3 Schrittmotor kontrollieren. Es gibt eine Platine, die eine Produktion von Coptonix GmbH ist, um maximal 3 2-Phasen Schrittmotoren unabhängig anzusteuern.

Die Probleme, auf welche ich getroffen bin, habe ich teilweise gelöst.
Als ich in Python die GUIs programmierte, hatte ich zuerst sehr viele Probleme mit der Ausrichtung eines Widgets. Der geometrische Manager darf nicht durcheinander benutzt werden. Wenn man einen Manager benutzt hätte, würden alle Widgets mit diesem Manager verwendet. Ansonsten würde das Kompilieren nicht funktionieren. Wenn man auf der gleichen Position zwei Widgets geschrieben hätte, würde nur das letzte Widget da bleiben. D.h wenn man so viele Widgets auf eine Position gestellt hätte, würde nur das letzte Widget angezeigt werden. Wenn es die Möglichkeit gäbe um in die Prozesse zu schauen, würde man die Funktion $update()$ vom Modul $Tkinter$ verwenden.
Wenn ich das Modul *ctypes* für die DLL benutzte, habe ich zuerst $byref$ vor dem Modul *ctypes* den Datentyp eines Arguments nicht gut unterschieden. Ich habe die Version von der Platine vor allem auch nicht erkannt, deswegen funktionierte die Platine anfangs in meiner Arbeit überhaupt nicht.
Als ich die Platine in Python programmierte, sollte ich mir die Anwendung von jeder Funktion und die Beziehung zwischen verschiedenen Funktionen überlegen. Einige Argumente existierten nicht nur in einer Funktion. Einige Funktionen mussten hinter einer anderen Funktion aufgerufen werden.
Bei der Programmierung für das Modell P3 hatte ich ein großes Problem, weil nur die Read-Methode als Beispiel im Manual gegeben wurde. Die Write-Funktionen wurden nicht gegeben, dann habe ich sehr viele Methode ausprobiert. Zum Schluss habe ich rausgefunden, dass man mit ein *set* vor den Methoden zusammenschreiben konnte, um eine Write-Funktion zu erzielen.
Bei der Speichermethode brauchte ich mehr Zeit als andere Teile von meiner Arbeit. Ich konnte vor allem die Speicherfolge nicht so genau berechnen, wenn zwei Kanäle in einem

Zeitpunkt gleichzeitig gespeichert wurden. Ich habe zuerst auch nicht gewusst, was für eine CVS-Datei es ist und wie man eine CVS-Datei erstellen kann.
Aber die Speichermethode, mit der man die Daten von der Multimediakarte via USB auf dem PC speichert, habe ich nicht geschaffen.

Die GUI-Programme können für das Kontrollieren und Anwenden intuitiv und deutlich verwendbar sein. Man kann auch die Schrittmotoren in Abhängigkeit von den Messwerten des Modells P3 ansteuern, um die Dehnungsänderung lebendiger anzuzeigen.

Literaturverzeichnis

[1] Hohe, Niedrige Präzisionswiderstandsmessung, 2008.

[2] Coptonix GmbH, Luxemburger Str. 31 D-13353 Berlin. *Manual USB Motion 3X II.*

[3] Pter Kaiser and Johannes Ernesti. *Python - Das umfassende Handbuch.* Galileo Computing, 2008.

[4] KYOWA. *What're STRAIN GAGES*, November 2005.

[5] Alex Martelli. *Python in a Nutshell.* O'Reilly, March 2003.

[6] Felix Schörlin. *Mit Schrittmotoren steuern, regeln und antreiben.* Franzis, 1995.

[7] VISHAY, Vishay Micro-Measurements P.O. Box 27777 Raleigh, NC 27611 USA. *Model P3 Strain Indicator And Recorder, ActiveX Control Developer's Guide*, February 2004.

[8] VISHAY, Vishay Micro-Measurements P.O. Box 27777 Raleigh, NC 27611 USA. *Model P3 Strain Indicator And Recorder, Instruction Manual*, July 2007.

[9] Michael Weigend. *Python GE-PACKT.* mitp, 2008.

[10] Michael Weigend. *Objektorientierte Programmierung mit Python 3.* mitp, 4 edition, 2010.

[11] W.Müller. Dehnungsmessstreifen (dms). Technical report, Carl-Engler-Schule Karlsruhe, 2009.

Erklärung

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

(Mingzhu Xiu)