Inhaltsverzeichnis

1. Hintergrund

1.1. Grundidee

Die Grundidee ist, ein Positioniersystem für Testobjekte in einem Thorax-Phantom zu entwerfen und zu programmieren. Dies ermöglicht ein Testen und Vergleichen verschiedener Elektroimpedanztomographie-Hardware und Software. Bei der Elektroimpedanztomographie (EIT) werden Wechselströme durch Elektroden in den Körper eingespeist. Durch die Änderung des erzeugten elektrischen Feldes und somit der elektrischen Potentiale, die durch Sensoren gemessen werden, können in Echtzeit Rückschlüsse auf die Änderungen der lokalen Leitfähigkeiten im Inneren des Körpers gezogen werden. So ist es möglich, durch einen Elektrodengurt, der sich am Thorax eines Patienten befindet, in Echtzeit die Funktion der Lunge zu visualisieren, da die isolierende Luft und deren Menge das elektrische Feld in der Lunge beeinflussen. Das ermöglicht zum Beispiel die Früherkennung eines Lungenkollapses. Die Einordnung solcher Systeme bezüglich Genauigkeit und Zuverlässigkeit erfordert das Testen folgender Eigenschaften:

1. Homogeneity
2. Detectability
3. Linearity
4. Distinguishability
5. Precision

Um solche Tests in einem Thoraxphantom vorzunehmen, wird ein „EIT-Phantom-Positioning-System“ (EIT-P-PS) benötigt, welches in der Lage ist, zwei Testobjekte verschiedener Form, Volumen und Impedanz unabhängig voneinander und vollautomatisch an vorbestimmten Orten zu positionieren.

1.2. Anforderungen

Die Anforderungen an das Positioniersystem sind im Dokument „Positioning means (working title „robot“: System Requirement Specification“ V 0.2 festgelegt.

2. Umsetzung

2.1 Hardware

2.1.1 Prinzip

Es gilt, ein Prinzip zu entwerfen, dass das völlig unabhängige Positionieren von zwei Objekten in einem zylinderförmigen Wassertank erlaubt. Nach eingehender Prüfung von verschiedenen Möglichkeiten wie Schienensysteme oder Roboterarme entscheide ich mich für folgendes Prinzip:

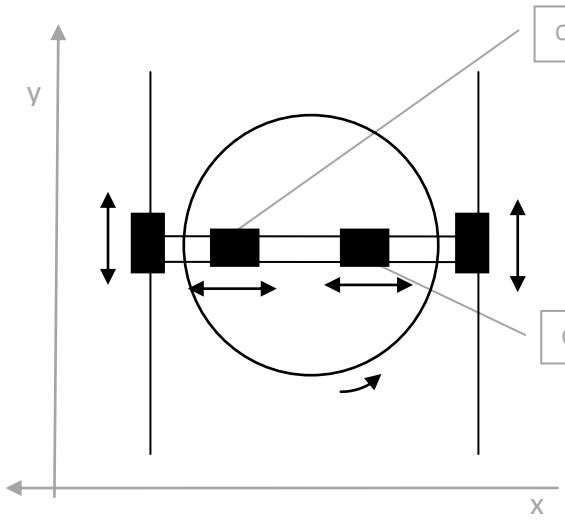


Abbildung 1

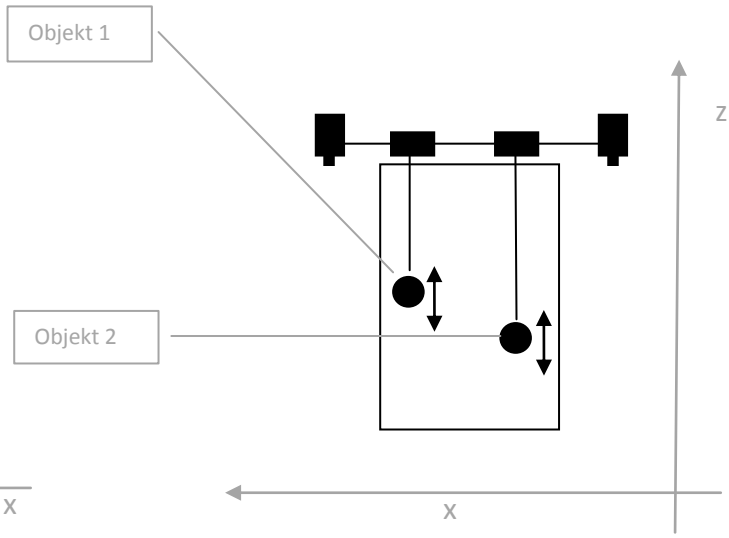


Abbildung 2

In Abbildung 1 und 2 ist das Prinzip schematisch illustriert. Jedes Objekt ist nun unabhängig vom anderen sowohl in x- als auch in z-Richtung platzierbar. Die zwei Schlitten auf der x-Achse befinden sich jedoch auf einer Schiene, was bedeutet, dass sie in y-Richtung nur gemeinsam bewegt werden können. Um nun mit beiden Objekten jede Position erreichen zu können, muss aus geometrischen Überlegungen das ganze System auf dem Zylinder zusätzlich drehbar sein. Ausgenommen sind Positionen mit identischen xy-Koordinaten oder Positionen die je nach Objektgrösse zu deren Kollision führen würden. Diese beiden Bedingungen sind jedoch nicht gefordert. Der Glaszylinder wird so in ein xyz-Koordinatensystem eingeteilt, dass dessen Ursprung sich auf der Ebene des Elektrodengürtels und in der Mitte des Zylinders befindet.

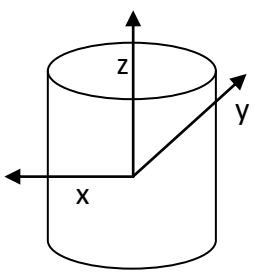


Abbildung 3

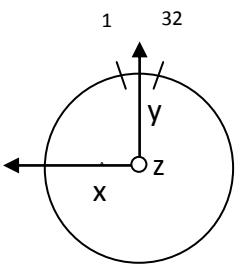


Abbildung 4

In den Abbildungen 3 und 4 ist die Orientierung des Koordinatensystems ersichtlich. Da in der Medizin Tomographie-Bilder von den Füssen in Richtung Kopf betrachtet werden, ist eine Umkehrung der x-Achse und der Elektrodenbeschriftung erforderlich, um beim resultierenden Bild ein kartesisches Koordinatensystem mit einer Elektrodenbeschriftung im Uhrzeigersinn zu erhalten.

2.1.2 System

In diesem Kapitel geht es um die Umsetzung des gewählten Prinzips. Nach Rücksprache mit einem Maschineningenieur entscheide ich mich, das gesamte System aus Kostengründen und aufgrund der dadurch einsetzbaren hochpräzisen Motoren vorerst mit LEGO und deren Mikrokontroller „Mindstorms NXT“ umzusetzen. Ich benötige zwei Servomotoren mit integrierten optischen Rotationssensoren für die x-Bewegungen, zwei für die z-Bewegungen, einen für die y-Bewegung und einen für die Rotation. Diese sechs Motoren müssen aus Mangel an geeigneten Motor-Multiplexer für den NXT, der nur drei Outputs steuern kann, mit zwei Mindstorms NXT kontrolliert werden. Es gibt verschiedene Möglichkeiten, die Rotation der Motoren in eine Translation umzuwandeln. Ich verwende für alle Motoren einen Antrieb auf Basis von Zahnrad auf Zahnstange, da diese Umsetzung eine hohe Genauigkeit garantiert. Ein 71.5 cm x 48.0 cm LEGO-Rahmen dient als Grundplatte.

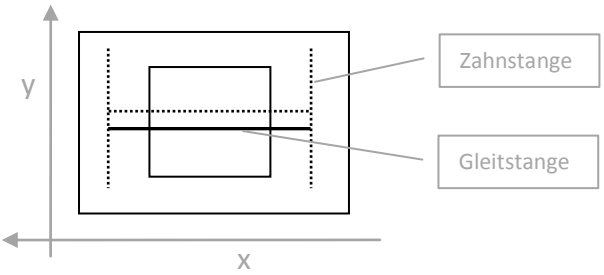


Abbildung 5

Nun befestige ich auf der linken und der rechten Seite in y-Richtung zwei Zahnstangenschienen. Auf diesen Schienen bewegen sich zwei y-Schlitten durch den y-Motor, der ein Zahnrad antreibt, und die Rotation durch eine Achse auf den zweiten y-Schlitten transferiert. Zwischen diesen Schlitten wiederum wird eine Zahnstange, in x-Richtung, befestigt. Parallel zu dieser Zahnstange hilft eine Gleitstange, das Gewicht der Objekte zu tragen. Auf diese beiden Stangen werden nochmal zwei x-Schlitten platziert, die durch die Motoren x_1 und x_2 und ein Zahnrad auf der Zahnstange bewegt werden. Auf dem x-Schlitten befinden sich die Motoren z_1 und z_2, die, wieder durch ein Zahnrad, eine senkrechte Bewegung zweier Zahnstangen ermöglichen, an deren Ende die Objekte in ein

M5 Gewinde eingeschraubt werden können. Am Rahmen befestige ich nun auf jeder Seite ein Rad, so dass die Anlage drehbar wird. Der Abstand der Räder wird so festgelegt, dass sie genau auf einen ringförmigen Spritzschutz passen, der wiederum auf das Thoraxphantom aufsetzbar ist. Der Rotations-Motor treibt nun ein Reibungsrad und ein Zahnrad auf einem Zahnkranz an. Die Rotation erfordert aufgrund der hohen Untersetzung, die durch das Gewicht der gesamten Anlage nötig ist, den Einsatz eines zusätzlichen optischen Sensors. Dieser Sensor ermöglicht die regelmässige Kalibrierung der Rotation alle 15 Grad und somit eine akzeptable Präzision. An den Mikrokontrollern werden ausserdem noch zwei Berührungssensoren angeschlossen, die für die Initialisierung an festgelegten Markierungen platziert werden und die Motoren stoppen, sobald die Ausgangspositionen (50, 0, -0.5 Objektdurchmesser) und (-50, 0, -0.5 Objektdurchmesser) erreicht sind. Diese z-Koordinaten lassen sich durch die verschiedenen Objektgrössen begründen und werden in der Software entsprechend ausgeglichen. Eine Ausnahme zu dieser Vorgehensweise bildet die Positionierung von langen Stangen. Für dies wird ein zusätzlicher Plexiglaszylinder auf das Phantom aufgesetzt. Nun kommt das Positioning-System auf diesen Zylinder, so dass sich die z-Achse um den zusätzlichen Zylinder nach oben verlängert. Dies ermöglicht das erforderliche vollständige Herausziehen der Stangen. Die Ausgangspositionen bei der Wahl von Stangen sind (50, 0, 0) und (-50, 0, 0).

2.2 Software

Es gilt, ein Programm zu schreiben, das es ermöglicht, das gesamte Positioning-System von einem beliebigen Computer aus zu steuern. Dabei müssen Protokolle erstellbar sein, die dann vollautomatisch und hochpräzise vom System abgefahren und in Echtzeit wieder gespeichert werden. Eine praktische Programmierumgebung für Windows Formulare ist .Net. Da die Kontroll-Oberfläche später in eine in C# programmierte Software integriert werden soll, entscheide ich mich für die objektorientierte Programmiersprache C# von Microsoft. Die Kommunikation via Bluetooth mit den Mikrokontrollern gestaltet sich jedoch auch mit der Bibliothek „Mindsqualls“ V1.2 Beta schwierig. Die Kontrolle der Motoren in C# unterstützt ausserdem keine Winkel-Grad-Befehle, was bedeutet, das Präzision verloren geht. Ich entscheide mich deshalb für zwei Programme, eines für die beiden Mikrokontroller und ein zweites für den Computer.

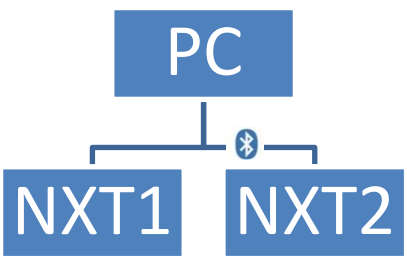


Abbildung 6

Auf dem Computer sind nützliche Koordinatenpaar-Protokolle erstellbar, deren Inhalt seriell via Bluetooth an die Mikrokontroller übergeben werden, wo sie in Motorenbefehle umgerechnet und abgefahren werden. Ist eine Position erreicht, erhält der Computer via Bluetooth eine Antwort und kann darauf reagieren, indem er die aktuellen Koordinaten in ein Protokoll einträgt, eine benötigte Wartezeit einschaltet und danach das nächste Koordinatenpaar sendet. Die Kommunikation zwischen Computer und Mikrokontroller erfolgt via ASCII-Steuerzeichen.

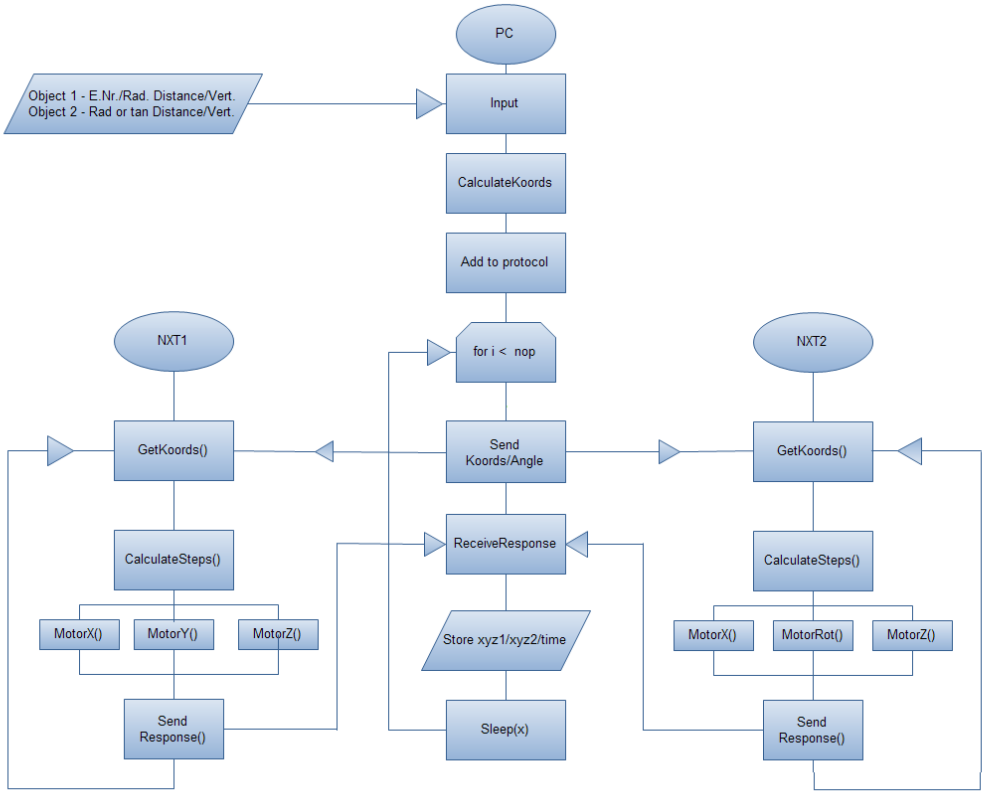


Abbildung 7

2.2.1 NXT EIT-P-PS Programm

Beide Mikrokontroller benötigen je ein Programm, wobei diese Programme sich nur in den Motorenbefehlen unterscheiden.

```
init();
while(true)
    {
    GetKoords();
    CalculateSteps();
    sem = 0;
    start Motorx1;
    start Motory;
    start Motorx2;
    NewtoAct();
    until (sem ==3);
    SendResponse();
```

Dies ist ein Codeauszug aus dem Programm für den NXT1, der die Motoren x_1, x_2 und y steuert. Das Programm wurde in der Programmiersprache NoteXactlyC (NXC) in der Programmierumgebung Bricx Command Center 3.3 erstellt. Das Programm beginnt mit der Funktion init(), in der alle Variablen und die Ausgangsposition initialisiert werden. Ausserdem wird der Schwellenwert des Lichtsensors für die Markierungen festgelegt. Der Lichtsensor, der sich zu Beginn auf einer schwarzen Markierung befindet, nimmt dazu den aktuellen Wert und erhöht diesen, um ein wenig Toleranz zu haben. Dann beginnt eine Endlosschlaufe.

Zuerst werden in GetKoords() die Koordinaten empfangen:

```
while(k < 8)
{
        ReceiveMessage(0, true, ins);
        if(StrLen(ins) > 0) {         //String-Länge in Mailbox grösser 0
        in = StrToNum(ins);             //String zu Nummer
         switch (k)                     //Switch
           {
             case 1:                    //wenn k=1
             x[2] = in;                 //1. in ist 1. x-Koordinate
             msg = NumToStr(x[2]);      //Ausgeben ans Display
             TextOut(0, LCD_LINE1, msg, false);
             break;
```

Obiger Codeausschnitt illustriert vereinfacht den Empfang der ersten Nachricht. Der Computer sendet die Zielkoordinaten der zwei Objekte in x_1, y_1, z_1 und x_2, y_2, z_2 in Millimetern. Ausserdem erhalten die Mikrokontroller noch den Steigungswinkel der Geraden durch die beiden Punkte (siehe CalculateSteps), weil NXC keine Arcus-Tangens Funktion besitzt. Während k < 8 rufen die NXT ihre Mailbox ab, das sogenannte „Automatic polling“. Sobald die Länge des Strings in der Mailbox grösser als null ist, gelangt das Programm in die Switch-Bedingung. Diese teilt die empfangene Nachricht je nach Wert von k den Arrays x, y, z und der Variable phi2 zu. Dann wird k um eins erhöht und die While-Schlaufe wiederholt sich bis k = 8 gilt und somit die Koordinaten und der Winkel empfangen sind. In der Funktion CalculateSteps werden nun die Schritte berechnet:

Um die von den Motoren auszuführenden Schritte zu berechnen, benötige ich zuerst die Schritte (in x_1, x_2, y, z_1, z_2 Richtung und der Rotation) die vom Ursprung zur Zielposition gemacht werden müssten. Danach kann ich die Schritte, die vom Ursprung zur neuen Position nötig sind, von den Schritten, die vom Ursprung zur aktuellen Position gemacht wurden, abziehen, und erhalte die Distanzen, die von den Motoren zurückgelegt werden

```
phis = - phi1 + phi2;         //Berechnung der auszuführenden Schritte
xsO1 = - xs[0] + xs[2];
xsO2 = - xs[1] + xs[3];
ysO = - ys[0] + ys[1];
zsO1 = - z[0] + z[2];
zsO2 = - z[1] + z[3];
```

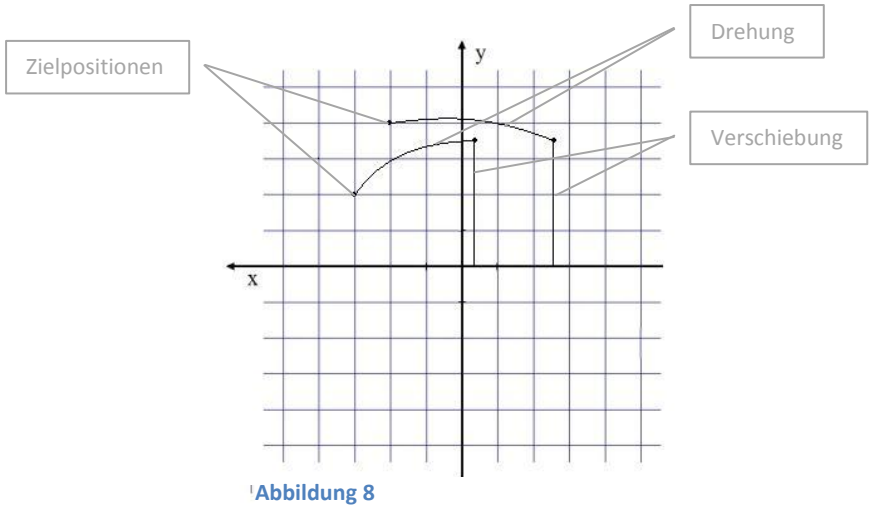
Die Schritte auf der z-Achse sind einfach zu berechnen, da sie nicht von der Drehung oder von der Position des jeweiligen anderen Schlittens abhängig sind. Das bedeutet, dass man mit der Differenz zwischen den z-Koordinaten der aktuellen Position und der neuen Position bereits die benötigte zurückzulegende Distanz in z-Richtung erhält. Durch die Verzahnung von Zahnrad auf Zahnstange kommt man auf den Faktor 7, der mit den z-Schritten in mm multipliziert werden muss, um die Gradanzahl zu erhalten, um die die z-Motoren rotieren sollen. Da jedoch sowohl bei einer Drehung als auch bei einer Bewegung in y-Richtung beide Schlitten abhängig voneinander bewegt werden, gestaltet sich die Berechnung der x- und y-Schritte und der Rotation schwieriger.

```
ys[1] = (y[3]*Cos(-phi2) + x[3] * Sin (-phi2))/100;
xs[2] = (x[2]*Cos(-phi2) - y[2] * Sin (-phi2))/100;
xs[3] = (x[3]*Cos(-phi2) - y[3] * Sin (-phi2))/100;
```

Ich benutze hierfür ein imaginäres Koordinatensystem anstelle der xy-Ebene. Um diese Schritte zu berechnen, kehre ich das Problem um. Das heisst, ich versuche die beiden Zielpunkte mit Hilfe affiner Abbildungen auf den Ursprung zu projizieren und drehe danach einfach die Vorzeichen. Da es sich jedoch aufgrund der Irregularität einer solchen Abbildung um kein inverses Problem handelt, muss ich die Abbildung aufteilen. Deshalb drehe ich zuerst beide Zielpunkte um denselben Winkel φ, und zwar so, dass ihre Verbindungslinie parallel zur x-Achse zu stehen kommt. Dieser Winkel ist gerade der Steigungswinkel der Verbindungsgeraden und wird mit dem Arcus-Tangens der Differenz der y- bzw. der y-Werte der beiden Positionen berechnet. Folglich ist φ auch der Drehwinkel für die Rotation der Anlage. Die neuen Koordinaten werden nun mit einer affinen Abbildung (einer Drehung um den Ursprung) in der Polarform berechnet. Hier ein Beispiel mit der ersten Position:

$$x_{Schritt} + y_{Schritt}i = (x_{Ziel} + y_{Ziel}i) * e^{-\varphi i}$$
$$x_{Schritt} + y_{Schritt}i = (x_{Ziel} + y_{Ziel}i) * (Cos(-\varphi) + Sin(-\varphi)i)$$
$$x_{Schritt} = x_{Ziel} * Cos(-\varphi) - y_{Ziel} * Sin(-\varphi)$$
$$y_{Schritt} = y_{Ziel} * Cos(-\varphi) + x_{Ziel} * Sin(-\varphi)$$

Nun entsprechen die neuen Koordinaten der Punkte exakt den benötigten Schritten zum Ursprung, wobei die y-Koordinaten gleich sind, was somit das Problem der gemeinsamen y-Schiene löst.



[1]Abbildung 8

Eine Schwierigkeit taucht jedoch auf, sobald sich die beiden x-Schlitten kreuzen müssen. Das wird von einer if-Bedingung kontrolliert, die ausrechnet, ob die Differenz der zu gehenden Schritte der x-Motoren grösser als der Abstand zwischen den Objekten ist. Ist das der Fall, wird der Winkel φ um 180 Grad je nach Vorzeichen erhöht oder verkleinert. Nun werden die x- und y- Schritte mit dem neuen Winkel berechnet. Wurde vom Benutzer nur ein zu platzierendes Objekt gewählt, erhalten die Kontroller dies als Nachricht und berechnen die Schritte einfach in x- und y- Richtung, da eine Drehung nun unnötig wäre. Jetzt werden die Motoren gestartet:

Da NXC Multitasking unterstützt, werden die Motoren in verschiedenen Tasks gesteuert, so dass sie gleichzeitig rotieren. Die Distanzen in mm werden aufgrund der Verzahnung von Zahnrad in Zahnstange mit dem Faktor 7 multipliziert. Die Motoren drehen um diese Zahl in Grad. Diese Rechnung gilt für die Motoren x_1, x_2 und y. Bei der Rotation gestaltet sich eine Bewegung mit hoher Präzision jedoch schwieriger. Wie in Kapitel 2 erwähnt, schafft der Einsatz eines optischen Sensors Abhilfe. Alle 15 Grad befindet sich eine schwarze Markierung. Das heisst der Winkel φ wird durch 15 Grad geteilt und das Resultat entspricht der Anzahl Markierungen, die der Lichtsensor überfahren muss. φ modulo 15 ergibt nun den Rest. Dieser Rest wird nun, wie bei den anderen Motoren, nur noch auf der Zahnstange gefahren, mit dem Unterschied, dass der Multiplikationsfaktor höher ist, da hier eine Untersetzung stattfindet. Wird die nächste Position angefahren, dreht der Motor den Sensor zurück auf die Markierung, sodass das System wieder kalibriert wird. Dieser zurückgefahrene Winkel muss nun, je nach Drehrichtung, dazu- oder abgezählt werden. Nun kann der neue Winkel wie gewöhnlich abgefahren werden. Dazu wurde eine Kontrolle des Lichtsensors programmiert. Dieser meldet jede überfahrene Markierung. Ist der gedrehte Winkel des Motors zwischen zwei gemeldeten Markierungen zu gross oder zu klein, reagiert das Programm, indem bis zur letzten gesehenen Markierung zurückgefahren wird. Dann kann der noch nicht gedrehte Winkel nochmals abgefahren werden. So kontrollieren sich der optische Sensor und der integrierte Rotationssensor des Motors gegenseitig.

Ich benutze eine Semaphore sem um das Programm wissen zu lassen, wann die Objekte ihre Zielposition erreicht haben. Diese Semaphore wird von jedem Task nach der Abarbeitung des Motorenbefehls um eins erhöht. Sobald sem = 3 gilt, fährt da Programm mit der Funktion SendResponse fort:

In dieser Funktion senden die Kontroller eine Bestätigung an den Computer, dass die gewünschte Zielposition erreicht ist. Dies erfolgt jedoch nicht durch ein eigentliches Senden einer Nachricht, sondern die Kontroller speichern einen spezifischen Text in ihrer eigenen Mailbox ab. Der Inhalt dieser Mailbox wird dann durch den Computer via Automatic-Polling eingelesen. Nun beginnt die while-Schlaufe erneut, bis der Computer die Programme abbricht.

2.2.1.2 Initialization-Programm

Zusätzlich zu diesen beiden Programmen kommen noch zwei Programme zur Initialisierung des Systems. Diese beinhalten eine LCD gestützte Steuerung der Motoren. Die einzelnen Motoren können hintereinander vor- und zurückbewegt werden. Zeitgleich kontrolliert ein anderer Task die Touch-Sensoren. Werden diese betätigt, stoppt der laufende Motor. Das bedeutet, dass die Sensoren an die markierten Positionen gebracht werden können und bei einem Anstossen des entsprechenden Schlittens an den Sensor die Ausgangsposition erreicht ist. Damit ist der jeweilige Kalibrier-Vorgang beendet.

2.2.2 Computer EIT-P-PS Programm

Der Computer ist der Master in der NXT-PC-Bluetooth-Verbindung. Das heisst, er steuert die Kontroller. Ich benötige hier also ein benutzerfreundliches Programm, indem vom medizinischen Aspekt her nützliche Koordinaten-Protokolle erstellt und dann gesendet werden können. Ich erstelle in C# ein UserControl, so dass es sowohl über ein eigenes Windows Formular als auch über die EIT-Surfer-Software von Pascal Gaggero aufgerufen werden kann. Das gesamte UserControl wurde im Microsoft Visual C# 2008 Express Studio programmiert und setzt sich aus den dort verfügbaren Tools zusammen. Das gesamte Programm beruht, wie oft bei Windows Formularen, hauptsächlich auf Eventbehandlung. Sprich: jedem Button wird Code zugeordnet, der beim Event Mouse-Click abgearbeitet wird. Zusätzlich verwende ich die Bibliothek „MindSqualls“ Version 1.2, die es ermöglicht, mit einem Mindstorms NXT via Bluetooth in Verbindung zu treten. Wie bereits erwähnt, ist es möglich, via Direct Commands die Motoren zu steuern. Dies unterstützt jedoch keinen Einsatz der in den Motoren integrierten optischen Rotationssensoren. So speichere ich mit einem MessageWrite Befehl die Koordinaten in den Mailboxes der Mikrokontroller, die diese dann weiterverarbeiten.

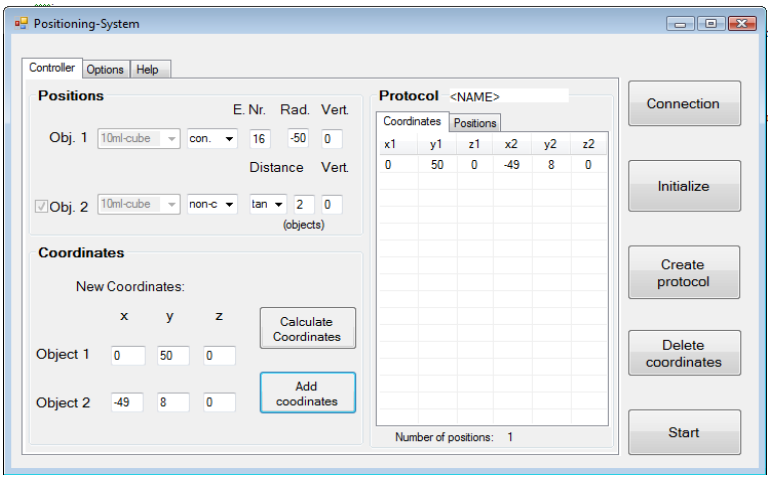


Abbildung 9

Ich unterscheide zwei mögliche Darstellungsmöglichkeiten einer Zielposition.

2.2.2.1 Positions

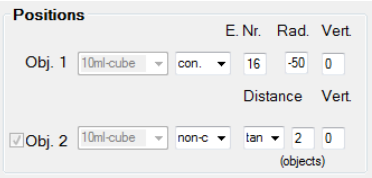


Abbildung 10

Die Positions-Darstellung ist eine Hilfe zur Erstellung medizinisch sinnvoller Koordinaten, denn zum Testen der in Kapitel 1.1 Grundidee genannten Punkte sind vor allem Eigenschaften wie die Nummer der Elektrode oder der Abstand zwischen den Objekten relevant. Zuerst werden in der ComboBox die Objekte ausgewählt. Bei diesen Objekten handelt es sich um 10ml-Würfel, 10ml-Kugeln, 50ml-Würfel, 100ml-Würfel, 100ml-Kugeln, 1ml-Stangen und 10ml-Stangen, die entweder leitend oder nichtleitend sind. Die Wahl anderer Objekte ist vorerst nicht möglich. Bei der Positionierung von anderen Objekten empfiehlt sich die Auswahl von Stangen in der ComboBox, da dann keine Autokorrektur in z-Richtung stattfindet. Diese Objekte sollten sich dann in der Ausgangsposition mit ihrer Mitte auf der Höhe der Elektrodenebene befinden. Das Positionieren in Nähe des Randes ist jedoch dann mit einem gewissen Kollisionsrisiko verbunden, da die Reachability-Guards nicht mehr richtig arbeiten. Die zwei Zielpositionen werden durch die Werte E.Nr, Rad, Vert1, Distance, Vert2 und der Auswahl zwischen rad oder tan definiert. E.Nr steht für den Raum zwischen zwei Elektrodennummern. Das bedeutet, dass eine E.Nr von „16“ das erste Objekt zwischen den Elektroden 16 und 17 platziert. Nun kann unter Rad der radiale Abstand vom Ursprung und unter Vert1 der vertikale Abstand von der Elektrodenebene in Millimetern des ersten Objekts gewählt werden. Das zweite Objekt wird nun in Abhängigkeit des ersten Objekts platziert. Es wird zwischen rad oder tan unterschieden, was bedeutet, dass das zweite Objekt in tangentialem oder radialen Abstand zum ersten Objekt platziert werden kann. Mit anderen Worten kann das zweite Objekt entweder im gleichen Winkel oder aber mit gleichem radialem Abstand vom Ursprung platziert werden. Das ist insofern interessant, als dass das elektrische Feld Richtung Zentrum an Dichte verliert. Um nun den Einfluss dieses Dichteverlustes auf die Visualisierung zu testen, kann derselbe Drehwinkel und somit rad gewählt werden. Will man jedoch zum Beispiel einen Test bezüglich unterschiedlicher Funktionsweisen einzelner Elektronen durchführen, darf das Bild nicht von einer veränderten Dichte abhängig sein und ein identischer radialer Abstand, also tan, ist gefragt. Der Abstand wird unter Distance eingegeben und hat die Einheit des Durchmessers des zweiten Objekts. Mit Vert2 wird dann der vertikale Abstand von der Elektrodenebene in Millimetern des zweiten Objekts festgelegt.

2.2.2.2 Coordinates

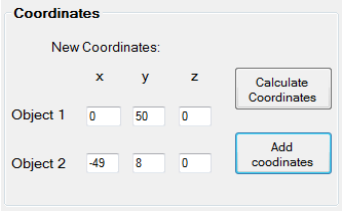


Abbildung 11

Die Coordinates-Darstellung dient in erster Linie dem Programm. Sowohl der Computer als auch die Kontroller können die Zielposition nur einordnen, wenn sie in der Coordinates-Darstellung stehen. Der Computer überprüft damit die Erreichbarkeit der gewählten Positionen und die Kontroller rechnen damit die Schritte zur neuen Position aus. Dieses Feld dient also als Zwischenspeicher für die gewählten Zielpositionen. Es ist jedoch auch möglich, direkt die Zielposition in der Coordinates-Darstellung einzugeben.

2.2.2.3 Umrechnung

Um die Umrechnung von der Positions-Darstellung zur Coordinates-Darstellung zu vollziehen, benötige ich ein imaginäres Koordinatensystem wie zur Berechnung der Schritte auf den Kontrollern. Die erste Zielposition wird in der Polarform $k * e^{\varphi i}$ dargestellt, wobei k = Rad und $\varphi = \frac{\pi}{2} - ENr * \frac{\pi}{16}$. In der zweitgenannten Formel wird der Winkel von der y-Achse ausgerechnet, indem die gewählte Elektrodenraumnummer mit dem Bogenmassabstand zwischen zwei Elektrodenräumen multipliziert wird. Um den Winkel von der x-Achse aus zu erhalten, wird das Resultat von 90° (im Bogenmass) abgezählt. Nun kann das ganze in die Form der komplexen Zahlen x + yi umgerechnet werden und es lassen sich die x- und y- Koordinaten ausrechnen.

$$x_1 = Rad * Cos(\varphi)$$
$$y_1 = Rad * Sin(\varphi)$$

Vert1 kann logischerweise direkt in z_1 umgeschrieben werden. Bei der Berechnung der zweiten Koordinaten unterscheidet man zwischen der Wahl von rad oder tan. Ist rad gewählt, so rechnet man x_2 und y_2 ähnlich wie in den oben dargestellten Formeln. Einzig statt Rad wird die Differenz zwischen Rad und Distance, in Millimetern umgerechnet, eingesetzt. Ist tan gewählt worden, muss der Drehwinkel der zweiten Position errechnet werden, dafür ist der radiale Abstand vom Ursprung gegeben (da er identisch mit dem radialen Abstand des ersten Objekts ist).

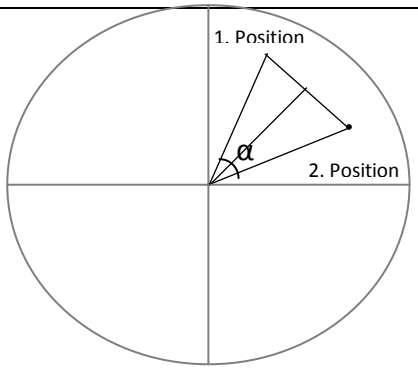


Abbildung 12

Da der Winkel der ersten Position bekannt ist, genügt es, den Zwischenwinkel α des eingezeichneten gleichschenkligen Dreiecks zu berechnen, um durch eine einfache Rechnung den Winkel der zweiten Position zu erhalten.

$$\alpha = 2 * Arcsin(\frac{Dist/2}{Rad})$$

Dieser Zwischenwinkel α erhält man durch die trigonometrische Formel oben. Das neue φ ist nun nichts anderes als die Differenz zwischen α und dem alten φ und die x- und y-Koordinaten können wie bei der ersten Position berechnet werden.

2.2.2.4 Protokoll

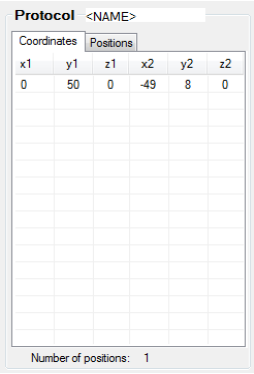
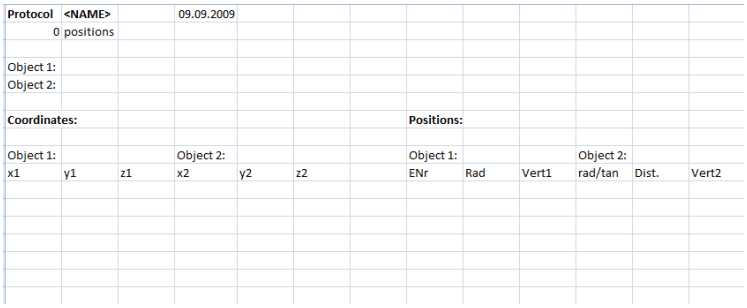


Abbildung 13

Das Protokoll ist die Liste der Koordinaten, die abgefahren werden sollen. Hier werden die Zielpositionen in der Coordinates-Darstellung und auch, falls ausgefüllt, in der Positions-Darstellung in zwei List-views niedergeschrieben. Es ist ausserdem möglich, mit einer Vorlage in Microsoft Office Excel 12.0 Protokolle zu erstellen und zu laden. In diesem Fall öffnet sich ein neues Excel-Sheet beim Event Mouse-Click auf Button „Create Protocol“. Um ein einwandfreies Einlesen eines erstellten Protokolls zu gewährleisten, wird immer dieselbe Vorlage verwendet.



Protocol	<NAME>		09.09.2009								
0	positions										
Object 1:											
Object 2:											
Coordinates:						**Positions:**					
Object 1:			Object 2:			Object 1:			Object 2:		
x1	y1	z1	x2	y2	z2	ENr	Rad	Vert1	rad/tan	Dist.	Vert2

Abbildung 14

Wird in dieser Vorlage ein Excel-Protokoll erstellt, werden in den spezifischen Feldern die Informationen eingetragen. Da für dies eine Referenz zu Excel hinzugefügt werden musste, gelingt das nur mit einer englischen Version des Excel 12.0. Diese ist in einer Trial-Version online verfügbar. Da das Programm überprüft, ob die Stringlänge unter Coordinates oder unter Positions null ist, ist es möglich, ein Protokoll mit allen Tools von Excel in beiden Darstellungen zu erstellen. Sollte das Protokoll in der Positions-Darstellung erstellt worden sein, erfolgt automatisch die Umrechnung (siehe Kapitel 2.2.2.3. Umrechnung). Sollte nur ein Objekt gewählt worden sein, ist ein Auffüllen der leeren Zellen mit 0 unter Objekt 2 erforderlich, da ansonsten beim Einlesen eine ArgumentException ausgelöst wird. Ob die Zielpositionen über Excel oder über das Formular selbst in die List-Views geladen werden, sie müssen dabei auf ihre Reachability überprüft werden. Dazu müssen die Objekte eingelesen und mit der CheckObj Methode die Durchmesser zugeordnet werden. Nun überprüft die CheckKoords Methode die Koordinaten auf drei Fehler. Der radiale Abstand der ersten Zielposition plus der Radius des Objekts darf den Radius des Zylinders nicht überschreiten. Dasselbe gilt für die zweite Zielposition. Zusätzlich darf der Abstand der Zielpositionen nicht kleiner als die Summe der Radien der Objekte sein. So wird jede Zielposition überprüft, bevor sie in die List-View eingetragen wird. Das garantiert die Erreichbarkeit jeder Zielposition, die sich in der List-View befindet.

2.2.2.5 Start

Beim Starten der Positionierung wird das gesamte Protokoll in der List-View abgefahren. Dazu wird zuerst mit den NXT's eine Verbindung hergestellt, um die Programme zu starten. Da es nicht möglich ist, mit beiden NXT's gleichzeitig verbunden zu sein, muss jeder Kontakt seriell erfolgen. So werden zuerst in der StopProgram Methode die laufenden Programme auf den Mikrokontroller beendet, um dann mit der StartProgram Methode die in 2.2.1 NXT EIT-P-PS Programm beschriebenen Programme gestartet. In einer For-Schlaufe werden nun die Items der List-View eingelesen. Zu den z-Koordinaten wird, sofern keine Stangen eingesetzt werden, jeweils der halbe Durchmesser der gewählten Objekte mit Hilfe der CheckObj-Methode addiert, da sich in der Ausgangsposition die Oberkante der Objekte auf der Elektrodenebene befindet. Dies erleichtert die Initialisierung. Die Koordinaten werden dann in der SendKoords-Methode zusammen mit dem Steigungswinkel der

Verbindungsgeraden mit MessageWrite als String seriell an die Kontroller gesendet. Dann, je nach Wahl, in Echtzeit in einem Excel-Sheet oder in einem Textfile mit dem Zeitraum des Haltens der Position zusammen abgespeichert. Sollte nur ein Objekt gewählt worden sein, wird statt des Winkels eine 4-stellige Zahl gesendet. Diese veranlasst die Mikrokontroller zu einer anderen Umrechnung der Koordinaten in die Motorenbefehle. Nun beginnt das Automatic-Polling in der ReceiveResponse-Methode. Dies geschieht solange, bis die Nachricht in den Mailboxes der Mikrokontroller eine bestimmte Form annimmt. Danach schaltet das Programm die gewählte Holding time ein. Dieser Ablauf wiederholt sich für jede Zielposition. Sind die Koordinaten in der List-View abgearbeitet, werden die Programme auf den Mikrokontrollern mit der StopProgram-Methode beendet.

2.2.2.6 Zusatzfunktionen

Zusatzfunktionen sind solche, die nicht unbedingt notwendig wären, aber die Benutzerfreundlichkeit erhöhen. Dazu gehören bestimmte Einstellungen in Options.

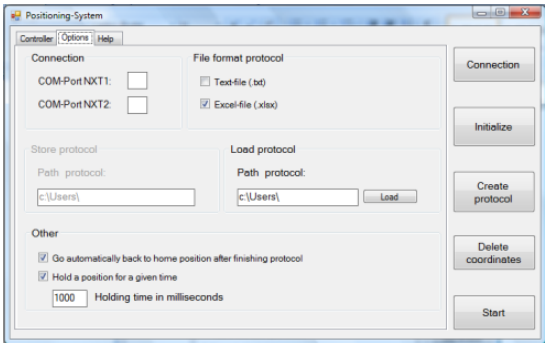


Abbildung 15

Es ist dort über eine Checkbox das automatische Anfahren der Ausgangsposition wählbar, sowie die Holding-time in Millisekunden einstellbar. Der Button „Initialize“ lässt den Computer mit dem StartProgram-Befehl die Initialization-Programme auf den Kontrollern starten und erleichtert das Anfahren der Ausgangsposition. Die Eventbehandlung des Buttons „Connection“ bewirkt das Aufrufen des Bluetooth-Device-Wizards, sofern eine Verknüpfung auf dem Desktop existiert. Ausserdem ist es möglich, durch den Button „Delete coordinates“, das gewählte Item in der List-View zu löschen. Auch ist in das Windows Formular eine Tree-View unter Help integriert. Beim MouseClick Event auf ein Element dieser Tree-View erscheinen Textboxes mit Erklärungen und Hilfen.

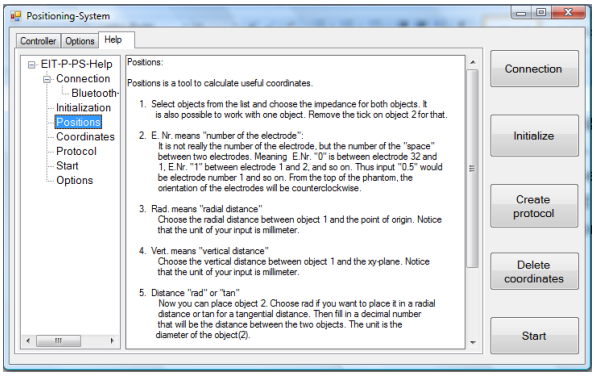


Abbildung 16

Die drei Komponenten Controller, Options und Help sind in einem „Tab“ zusammengefasst. Ausserdem ist eine Schnittstelle programmiert, die die nahtlose Integrierung in die EIT-Surfer-Software von Pascal Gaggero erlaubt. Diese setzt zwei Events PositionReached und PositionLeaved beim Erreichen, beziehungsweise Verlassen der Zielpositionen. Als Absicherung dient der bool-Parameter isbusy. Das ermöglicht das Abspeichern von sinnvollen EIT-Rohdaten, während die Objekte sich stationär in der gewünschten Position befinden, ohne die „nutzlosen“ Daten, die während der Bewegung der Objekte entstehen, zu berücksichtigen. Mit den Events werden ausserdem die Koordinaten mitgeliefert, sodass auch in der EIT-Surfer-Software die Zielpositionen in einem Protokoll abspeicherbar sind und direkt mit dem entstandenen EIT-Bild verglichen werden können.

3. Resultate

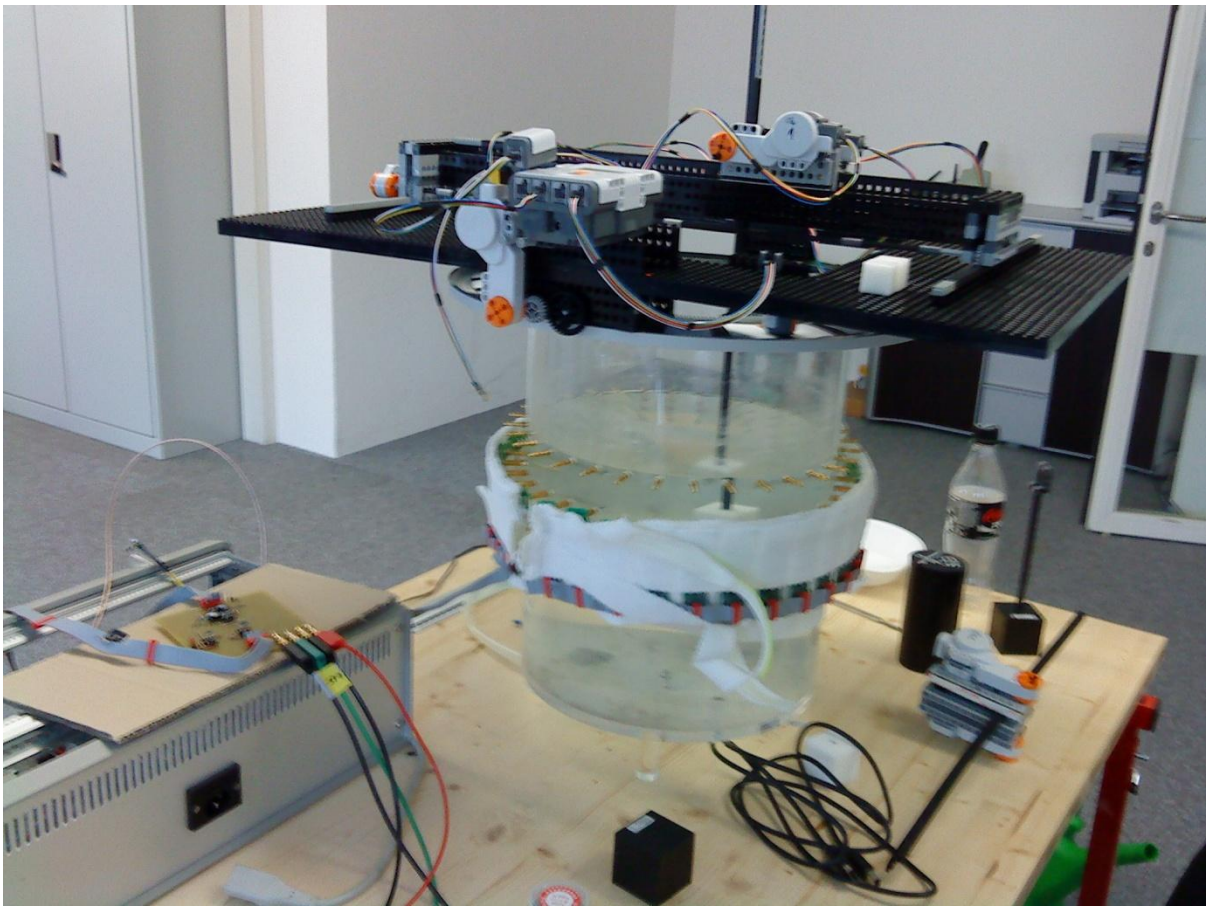


Abbildung 17

Das Resultat dieser Arbeit ist ein Positioning-System, das die Requirements erfüllt. Die Hard- und die Software erlauben das benutzerfreundliche Positionieren von verschiedenen Objekten in einem zylinderförmigen Glastank. Um das System bezüglich Präzision zu kategorisieren, habe ich verschiedene Tests durchgeführt, die im folgenden Kapitel erläutert werden.

3.1 Genauigkeit

3.1.1 xy-Bewegung

Um die Genauigkeit der Bewegung in x- und y-Richtung zu testen, habe ich folgenden Aufbau verwendet.

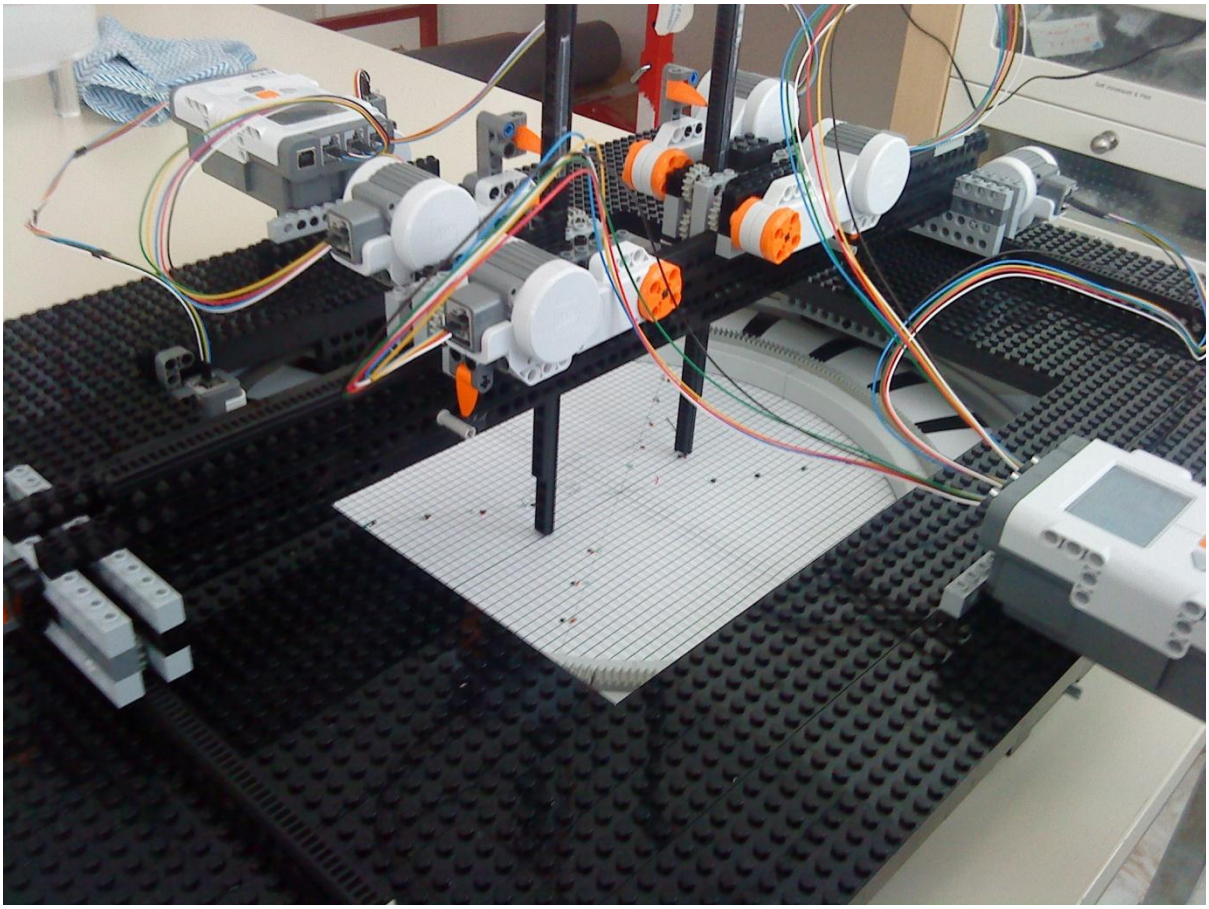


Abbildung 18

Das Positioning-System befindet sich auf einem Tisch, auf dem ein Koordinatensystem aufgeklebt ist. In diesem Koordinatensystem sind bestimmte Zielpositionen eingezeichnet. Im ersten Test befinden sich diese auf der x-Achse (grüne Kreuze), im zweiten Test parallel zur y-Achse (grüne Kreuze). Im dritten und letzten Test befinden sich die Zielpositionen verteilt im Koordinatensystem (blaue Kreuze), jedoch so, dass nur eine Bewegung in x- und y-Richtung und keine Rotation nötig ist (was bedeutet, dass die y-Koordinaten jeweils identisch sind). Die entsprechenden Protokolle werden dann abgefahren und die tatsächliche Position mit roter Farbe markiert.

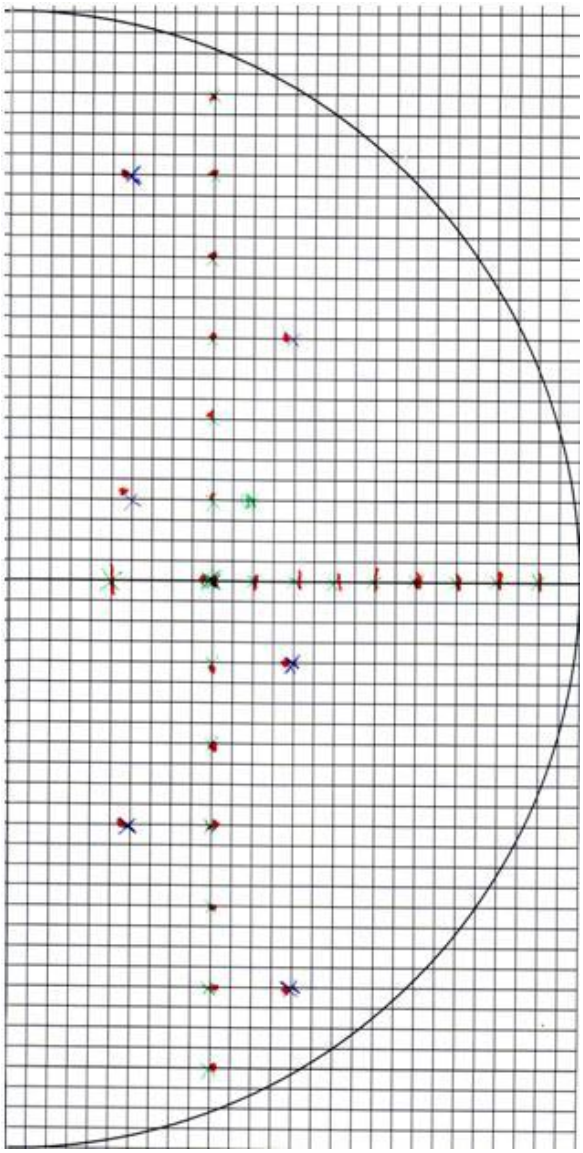


Abbildung 19

Es wird ersichtlich, dass sowohl im ersten als auch im zweiten Test die Genauigkeit sehr hoch ist. Es gibt nur wenige Abweichungen, und diese sind nicht grösser als ein Millimeter. Beim dritten Test ist ersichtlich, dass zwei Motoren im Einsatz sind. Dies erhöht die grösste Abweichung auf etwa zwei Millimeter.

3.1.2 z-Bewegung

Um die Genauigkeit der Bewegung in z-Richtung zu testen, habe ich folgenden Aufbau verwendet.

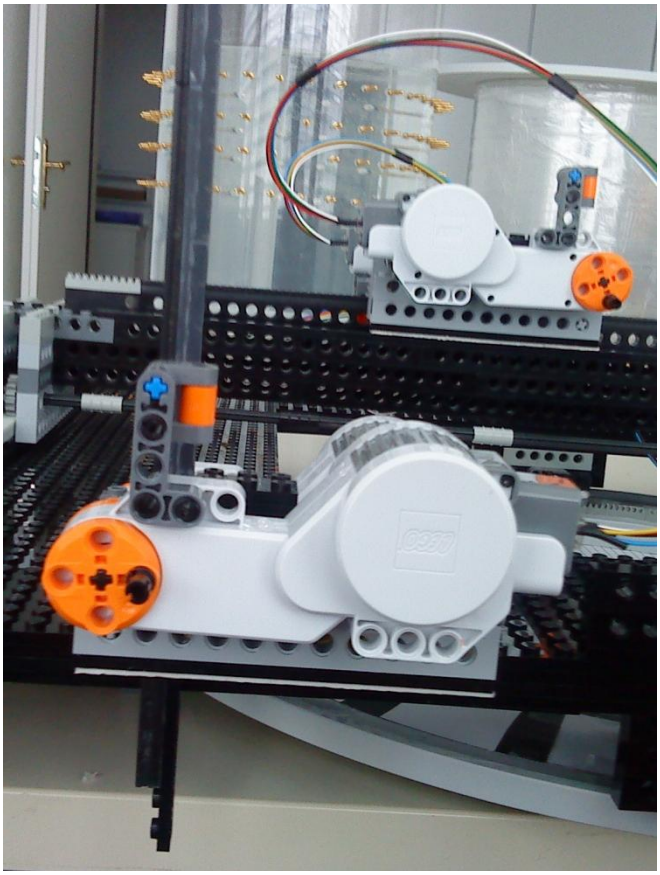


Abbildung 20

Die beiden Schlitten werden auf der Grundplatte befestigt und die Zahnstangen mit einem Streifen mit Markierungen in einem Zentimeter Abstand versehen. Nun lasse ich drei verschiedene Protokolle abfahren, bei denen jeweils nur die z-Koordinaten verändert werden. Folgende Abbildung ist das Resultat eines Tests und zeigt die Genauigkeit der z-Bewegung, da keine sichtbare Abweichung existiert.

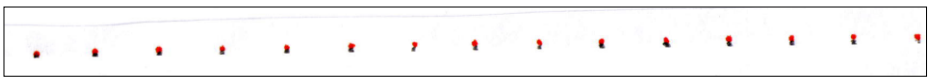


Abbildung 21

3.1.3 Rotation

Um die Genauigkeit der Rotation zu testen, habe ich denselben Aufbau wie für die xy-Bewegung verwendet. Hier ein Auszug aus einem Koordinatensystem für einen Test der Rotation. Bei den grünen Kreuzen handelt es sich um die Zielpositionen, die anderen Punkte sind die tatsächlichen Positionen. Deutlich zu sehen sind Abweichungen von einigen Millimetern.

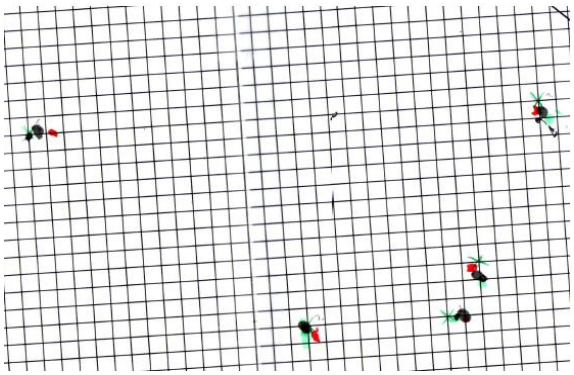


Abbildung 22

3.2 Diskussion

Die geringen Abweichungen von der Zielposition in x-, y, und z-Richtung in den vorangehenden Tests setzten sich im Wesentlichen aus zwei Aspekten zusammen. Zum einen rechnet der Algorithmus aufgrund der Unfähigkeit von NXC, mit Kommastellen zu arbeiten, auf einen Millimeter genau. Folglich können Rundungsfehler bis zu einem halben Millimeter entstehen. Ausserdem arbeiten die NXT-Motoren mit einem geringen Fehler von einem Grad Abweichung bei einer ganzen Umdrehung. Der Grund, weshalb die z-Bewegung genauer ist, sehe ich im Gewicht des Schlittens. Denn wenn an der z-Zahnstange geringe Gewichte befestigt werden, kommt es zu Abweichungen von etwa einem Millimeter. Der Hauptgrund für die signifikante Ungenauigkeit in der Rotation folgt jedoch noch aus einem dritten Aspekt. Die Zahnstangen auf dem grauen Ring sind gerade statt gebogen.

Abbildung 23

Mithilfe von Trigonometrie lässt sich hier eine maximale Abweichung von rund 1.5 Grad ausrechnen. Dazu kommt maximal 0.5 Grad Rundungsfehler. Dies verursacht eine mit dem radialen Abstand vom Ursprung zunehmende Ungenauigkeit. Da das System jedoch bei jeder neuen Zielposition mit dem Lichtsensor neu kalibriert, kumuliert sich dieser Fehler nicht. Gesamthaft bedeutet das, dass die Positionierung von einem Objekt mit einer Genauigkeit von zwei Millimetern, die Positionierung von zwei Objekten mit einer Genauigkeit von 5 Millimetern vonstatten geht. Ausserdem erwähnenswert ist die Tatsache, dass die Positionierung von Objekten mit einem Gewicht von 500 Gramm oder mehr durch die Wahl von LEGO eingeschränkt ist.