„Automatisierung der Ausrichtung von Satellitenbodenstationen“

Diplomarbeit

durchgeführt von

Rene Unterluggauer

Institut für Kommunikationsnetze und Satellitenkommunikation
der Technischen Universität Graz

Leiter: Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka

Begutachter: Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka
Betreuer: DI Michael Trieb

Graz, im März 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..

(Unterschrift)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitely marked all material which has been quotes either literally or by content from the used sources.

…….. ……..

date (signature)

Danksagung

Ich möchte mich an dieser Stelle bei meinem Betreuer Herrn DI Michael Trieb sowie bei Herrn Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka für die Unterstützung und Bereitstellung der Laboreinrichtung bedanken.
Danken möchte ich auch der Firma B&R welche mich im Laufe dieser Diplomarbeit telefonisch sowie durch Seminare und Workshops unterstützt hat.
Des Weiteren gilt großer Dank meiner Freundin (Christina Mohr) die mich immer mental unterstützt und motiviert hat, sowie meinen Eltern (Maria und Erwin Unterluggauer) die immer für mich da waren und mir finanziell das ganze Studium unter die Arme gegriffen haben.

Zusammenfassung

Um der Anforderung zu genügen eine Antenne auf einen Satelliten unabhängig vom aktuellen Standort und von der aktuellen Lage auszurichten, wird eine Versuchsanordnung mit zwei unabhängig voneinander ansteuerbaren Achsen entwickelt.

Als Unterkonstruktion dient ein Tisch aus *ITEM*-Profilen mit höhenverstellbaren Füßen. Als Sende- bzw. Empfangseinheit ist ein Offsetspiegel der Firma *TechniSat* in Verwendung. Globalen Variablen wie Längen- und Breitengrad vom aktuellen Ort und der Position des Satelliten sowie Neigungswinkel Alpha und Beta der Versuchsanordnung stellen die Grundlagen zur Berechnung dar. Mit Hilfe der orthogonalen Koordinatentransformation im Raum werden die entsprechenden Werte für die x- und y-Achse errechnet.

Um die Motoren von Bernecker und Rainer aus dem Zentrallabor der TU Graz in Betrieb zu nehmen, erfolgte die Einarbeitung in das Automation Studio – der Software für die Programmierung und Parametrisierung der zwei Motoren.
Um die proprietäre Programmiersprache „Strukturierter“ Text von *B&R* vorzustellen, wird diese für die Berechnung des Azimutwinkels gewählt. Der Programmcode für die Elevation ist in der bekannten Programmiersprache „C“ geschrieben.

Abstract

In order to be able to point an antenna to a satellite, independently from the current position and the current location, a test setup of two independently controllable axes was developed.

For the substructure, a table made from *ITEM*-profiles with height adjustable feet is used. An offset antenna by Technisat is utilised. Global variables, such as degree of longitude and latitude of the current position and longitude position of the satellite, as well as the angle of inclination Alpha and Beta of the test arrangement, are the basics for the calculation. By means of orthogonal coordinate transformation, the values for the azimuth direction angle and the elevation can be calculated.

In order to be able to use the motors by B&R in the central laboratory of Graz University of Technology, the software (Automation studio) was used. To present the proprietary software „structured text“ by B&R, it is chosen for the calculation of the azimuth angle. For the determination of the elevation the popular programming language „C” is used.

Inhaltsverzeichnis

1 Einleitung

Im Falle von geostationären Satelliten und der Verwendung von kleinen Antennen ist eine einmalige Ausrichtung der Antenne ausreichend.
Abhängig von der Position des Satelliten und den Koordinaten der Bodenstation ergeben sich der Azimut- und Elevationswinkel der Antenne.
In den meisten Fällen erfolgt die Ausrichtung manuell. Ziel der vorliegenden Arbeit war, eine automatische Ausrichtung mit Hilfe einer Industrial Ethernet-Infrastruktur aufzubauen.

1.1 Hintergrund

Das Institut für Kommunikationsnetze und Satellitenkommunikation der TU Graz beschäftigt sich unter anderem mit bidirektionalem Datentransfer zwischen Satelliten und Bodenstationen.

Für das Empfangen und Senden der Daten ist eine Satellitenbodenstation zwingend notwendig. Zwei mobile Bodenstationen sind an der TU Graz für Forschungszwecke vorhanden. Diese können manuell auf die jeweilige Zielposition eingestellt werden.

Abb. 01: Satellitenbodenstation

(Quelle: http://www.iks.tugraz.at/gallery/2008-05-airborne-firefighting-exercise/879/view)

Aus Gründen der Effizienz und der Komfortsteigerung ist eine Automatisierung im Bereich der Ausrichtung erstrebenswert.
Es gibt bereits Systeme am Markt, welche in der Lage sind, automatisch die Antenne auf den Satelliten auszurichten. Die Firma *ALDEN Loisirs et Techniques* mit Sitz in Huttenheim in Frankreich ist ein Hersteller von automatischen TV/Internet Satelliten-Anlagen.

Abb. 02: ALDEN SAT-NET® Mondo 90 Platinum
(Quelle: http://www.sat-welt.com/Zoom/mondo.html)

Mit dem Produkt Mondo 90 Platinum präsentiert die Firma *Alden* eine Breitband-Internetübertragung für das Reisemobil. Zum ersten Mal gibt es eine Möglichkeit Fernseh- und Internetnutzung zu kombinieren und das zum gewohnten Komfort von zuhause (vgl. Alden Loisirs et Techniques, S.19).
Alden bietet gegen eine Prepaid-Gebühr auch Tarife für die Internetnutzung an. Die Datenraten bewegen sich zwischen maximal 756 bis 2.048 kbps im Download- sowie maximal 128 kbps im Upload-Bereich (vgl. Alden IPcopter).
Diese Sat-Anlage kann jedoch Lage-Koordinaten nicht berücksichtigen. Das heißt, sie muss auf waagrechtem Untergrund stehen.

1.2 Zielsetzung

Ziel dieser Diplomarbeit ist es, ein automatisches Pointingsystem für Satellitenbodenstationen zu entwickeln.

Aufgrund einer durch GPS ermittelten Position sollen unter Berücksichtigung der aktuellen dreidimensionalen Ausrichtung (Lage der Versuchsanordnung) Parameter für die Ansteuerung der Motoren einer Satellitenbodenstation ermittelt werden.

Mit diesen Parametern sollen zwei Motoren der Firma *B&R* angesteuert und ein Satellit gefunden werden.

2 Industrial Ethernet

Industrial Ethernet stellt die Basis für eine effiziente Automatisierung dar. Die Anforderungen an Kommunikationsnetze in der Industrie sind sehr hoch und müssen sehr leistungsfähig sein. Die Basistechnologie „Industrial Ethernet“ hat sich hierfür etabliert (vgl. Siemens).
Um Kollisionen in einem Ethernet-Netzwerk (LAN) zu vermeiden sind Mehrfachzugriffsprotokolle erforderlich. Dazu zählen zum Beispiel ALOHA, CSMA oder POWERLINK im Fall von B&R.

2.1 ALOHA

Das ALOHA-System wurde um 1970 an der Universität Hawaii entwickelt.

2.1.1 Reines ALOHA

Beim reinen ALOHA ist keine globale Zeitsynchronisation notwendig. Es wird ein Rahmen mit Informationen gesendet und auf eine Bestätigung gewartet. Kommt es zu einer Kollision mit einem anderen Rahmen werden beide zerstört und es erfolgt keine Bestätigung. Daraufhin wartet die sendende Station eine zufällig gewählte Zeit und sendet erneut. (vgl. Tanenbaum 1997, S. 264).
Da beim reinen ALOHA die Sendestation vor der Übertragung den Kanal nicht auf Benützung abfragt, hat sie keine Möglichkeit zu sehen ob schon ein Rahmen zu einer andern Station unterwegs ist. (vgl. Tanenbaum 1997, S. 266)
Aufgrund der vielen Kollisionen beträgt die Kanalbelegung beim reinen ALOHA nur 18 Prozent. (vgl. Tanenbaum 1997, S. 267)

2.1.2 Unterteiltes ALOHA

1972 wurde das unterteilte ALOHA entworfen. Die Idee war, die Zeit in Intervalle einzuteilen. Ein Intervall entspricht der Länge eines Rahmens. Ein Sender darf nach einer Kollision nicht sofort senden sondern muss auf den nächsten Zeitschlitz warten. Dadurch kommt es zu weniger Kollisionen und die Effizienz wird verdoppelt. (vgl. Tanenbaum 1997, S. 267)

2.2 CSMA (Carrier Sense Multiple Access)

Beim CSMA-Protokoll handelt es sich um ein Trägererkennungsprotokoll, welches vor dem Senden den Kanal abhört. Ist der Kanal besetzt, muss der Sender warten und beginnt zu senden, sobald der Kanal frei ist. Im Falle einer Kollision wartet der Sender eine zufällige Zeitspanne und sendet erneut. Dieses Protokoll wird 1-Persistent CSMA genannt, da der Sender mit einer Wahrscheinlichkeit von 1 sendet, wenn ein Kanal frei ist. (vgl. Tanenbaum 1997, S. 268)
Es kann auch hier zu Kollisionen kommen. Zum Beispiel sendet Station 1 einen Rahmen der Station 2 aber noch nicht passiert hat. In diesem Fall sieht Station 2 einen freien Kanal und sendet. Oder Station 1 und 2 warten auf einen freien Kanal und beginnen gleichzeitig zu senden. (vgl. Tanenbaum 1997, S. 269)
Trotz dieser Möglichkeiten an Kollisionen ist das Protokoll 1-Persistent CSMA effektiver als das unterteilte ALOHA. Die Kanalauslastung liegt bei zirka 53 Prozent.

Die zweite Variante ist das Nonpersistent CSMA. Hier sendet die Station nicht sofort nach dem Freiwerden des Kanals sondern wartet nach der ersten Überprüfung eine gewisse Zeit und prüft dann erneut. Dies führt zu einer höheren Kanalauslastung und zu längeren Wartezeiten. (vgl. Tanenbaum 1997, S. 269)

Eine weitere Art ist das P-Persistent CSMA. Dabei hört der Sender den Kanal ab und sendet mit der Wahrscheinlichkeit p. Mit der Wahrscheinlichkeit q = p - 1 wartet die Station bis zum nächsten Zeitschlitz. Dort beginnt die Prozedur von vorne. Die Kanalauslastung wird bei dieser Variante höher wenn die Wahrscheinlichkeit zu senden geringer ist. (vgl. Tanenbaum 1997, S. 269)

2.3 CSMA mit Kollisionserkennung

Das Protokoll CSMA/CD (Carrier Sense Multiple Access with Collision Detection) beendet im Falle einer Kollision die aktuelle Übertragung des ohnehin zerstören Rahmens. Dies spart Zeit und Bandbreite. Das Protokoll wird meist bei LANs verwendet. Tritt eine Kollision auf beendet jede Station die Übertragung, wartet eine zufällige Zeit und beginnt erneut mit dem Senden. (vgl. Tanenbaum 1997, S. 270)
In Abb. 03 wird die Kanalauslastung der beschriebenen Protokolle dargestellt.

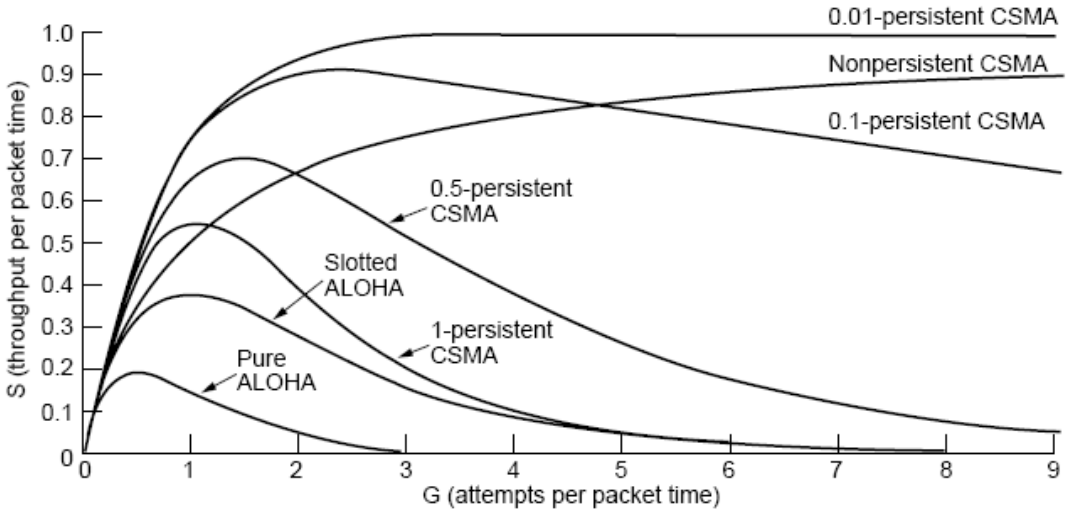

Abb. 03: Kanalauslastung verschiedener Protokolle
(Quelle: Tanenbaum S. 269)

2.4 POWERLINK

Bei B&R wird das offene Echtzeit Ethernet Protokoll POWERLINK für die Vernetzung der Automatisierungskomponenten verwendet. Es ist ein streng deterministisches Echtzeitprotokoll auf der Basis von Fast-Ethernet (100MBit). (vgl. Automation Studio B&R Help Explorer, 2010, POWERLINK)
In einem POWERLINK Netzwerk gibt es immer einen Managed Node (MN) und einen oder mehrere Controlled Nodes (CN). Der MN bestimmt die Abläufe im Netzwerk. Die Servoverstärker können nur CNs sein. (vgl. Automation Studio B&R Help Explorer, 2010, POWERLINK)
Das Grundprinzip basiert auf dem Prinzip der zyklischen Kommunikation. Innerhalb von zeitlich gleichbleibenden Zykluszeiten wird der Kommunikationsablauf immer wieder wiederholt. Ein POWERLINK-Zyklus beginnt mit einer SoC (Start of Cycle)-Nachricht. Dieser SoC wird als Multicast versendet, kann von allen CNs empfangen

werden und dient ausschließlich der Zeitsynchronisation. (vgl. Automation Studio B&R Help Explorer, 2010, POWERLINK)

Nach dem Senden des SoC spricht der MN jeden CN im Netzwerk mit einem PReq (PollRequest) an und die CNs antworten mit einem PRes (PollResponse). Der PReq kann auch als Unicast versendet werden. Wobei hier nur der angesprochene CN antwortet. Der PRes wird als Multicast versendet und dadurch können die CNs auch miteinander kommunizieren. Durch die direkte Querkommunikation werden die Zeiten für den Datenaustausch zwischen den Stationen verkürzt. Der CN sendet nur wenn er eine vom MN direkt adressierte Aufforderung (PReq) erhält. Durch dieses Verhalten werden Kollisionen im Netzwerk vermieden. (vgl. Automation Studio B&R Help Explorer, 2010, POWERLINK)

3 Winkelberechnung

Für sämtliche Berechnungen wird angenommen, dass die Erde eine Kugelform besitzt.

3.1 Konstanten

Folgende Konstanten werden verwendet:
Erdradius: r = 6.378 km
Höhe des Satelliten: h = 35.880 km
Breitengrad des Satelliten: 0 Grad
Längengrad des Satelliten: 19,2 Grad Ost

3.2 Längengrad und Breitengrad

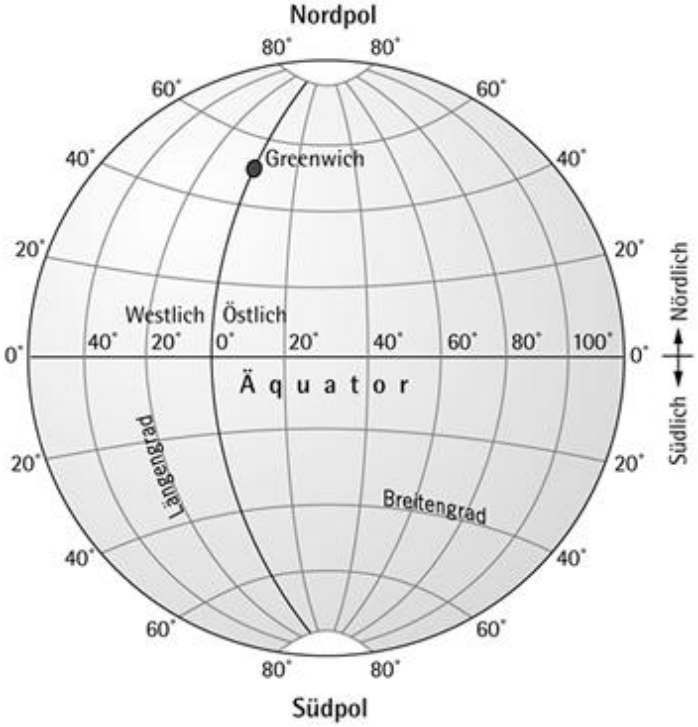

Abb. 04: Längen- und Breitengrade
(Quelle: http://www.astro.com/im/wiki/02_039.png)

Jeder Punkt der Erde kann durch den Längengrad, den Breitengrad und die Seehöhe eindeutig beschrieben werden.
Der Nullmeridian, welcher durch die Stadt Greenwich in England verläuft und der Äquator bilden die Referenzen zur Definition von Längen- und Breitengrad.

3 Winkelberechnung

Ausgehend vom Äquator, welcher senkrecht zur Erdachse liegt, gibt man von 0 bis 90 Grad die nördliche bzw. südliche Breite an. Die Längengrade werden vom Nullmeridian ausgehend in östlicher und westlicher Richtung angegeben. Die Längengrade erstrecken sich von 0 bis 180 Grad Ost (engl.: East) bzw. von 0 bis 180 Grad West (engl.: West). Negative Werte von Längen- und Breitengrad zeigen nach Westen bzw. Süden (vgl. Kowoma).
Um eine möglichst feine Auflösung bei der Angabe von Längen- und Breitengrad zu erhalten kann ein Grad in 60 Bogenminuten und eine Bogenminute wiederum in 60 Bogensekunden unterteilt werden.
Als Beispiel für die Darstellung von Längen- und Breitengrad werden die Koordinaten des Flughafens Graz aus Tab. 01 als Grundlage gewählt.

Tab. 01: Längen- und Breitengrad Flughafen Graz in Grad, Minuten, Sekunden

Längengrad:	15° 26' 24'' E
Breitengrad:	46° 59' 35'' N

(Quelle: http://www.skyscanner.at/flughaefen/grz/graz-flughafen.html)

Über die folgende Formel gelangt man zur Schreibweise in Grad:

$$15° + \frac{1}{60} \cdot \left(26° + \frac{1}{60} 24° \right) = 15{,}44° \text{ bzw. } 46° + \frac{1}{60} \cdot \left(59° + \frac{1}{60} 35° \right) = 46{,}993°$$

Rein in Grad angegeben hat der Längengrad einen Wert von 15,44° und der Breitengrad einen Wert von 46,993°.

Die Angaben zu Längen- und Breitengrad für den Flughaften Graz in Grad sind in Tab. 02 dargestellt.

Tab. 02: Längen- und Breitengrad Flughafen Graz in Grad

Längengrad:	15,44° E
Breitengrad:	46,993° N

Welcher Größenordnung ein Breitengrad in Kilometern entspricht kann mit folgender Berechnung gezeigt werden:

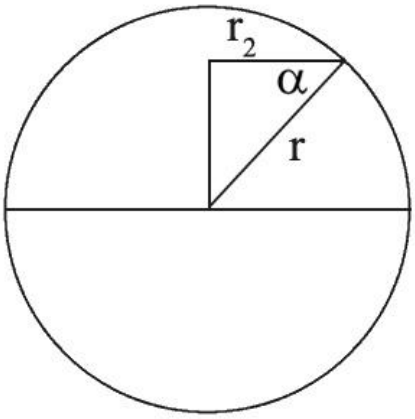

Abb. 05: Skizze zur Berechnung des Abstands zwischen zwei Längengraden

(gezeichnet von Unterluggauer)

$r =$ 6.378 km

$u = 2r\pi =$ 40.074,15589 km

$\frac{u}{360} =$ 111,317 km

Ein Breitengrad entspricht am Äquator zirka 111,32 km.

Mit steigendem Breitengrad rücken die Längengrade näher zusammen.

$r =$ 6.378 km

$\alpha =$ 46,993°

$r_2 = \cos\alpha \cdot r$

$r_2 =$ 4350,36 km

$u_2 = 2r_2\pi =$ 27.334,12 km

$\frac{u_2}{360} =$ 75,928 km

Am 47. Breitengrad entspricht ein Längengrad nach folgender Berechnung zirka 75,93 km.Um die Richtung zu einem Satelliten zu beschreiben müssen am Parabolspiegel zwei Winkel definiert werden.

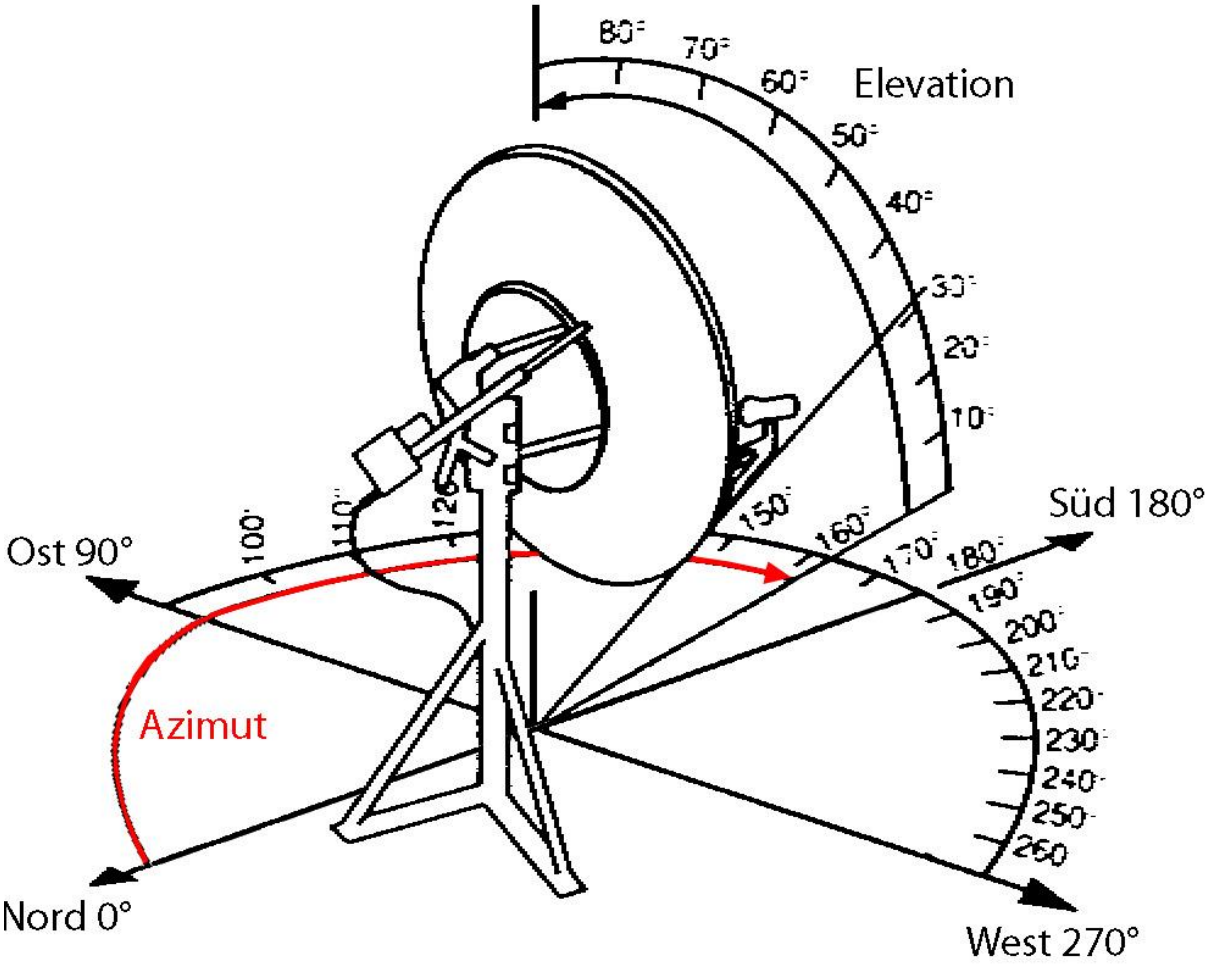

Abb. 06: Bereiche von Azimut- und Elevationswinkel in Grad

(Quelle: http://www.fen-net.de/satellitentechnik-online/images/richtung.gif)

3.3 Azimutwinkel

Der Azimutwinkel beschreibt den Winkel in horizontaler Ebene zwischen der tatsächlicher Position und der Nordrichtung. Mittels des Seitencosinussatzes, angewandt auf ein schiefwinkliges sphärisches Dreieck (wobei $\alpha = 90°$) kann dieser Winkel (hier γ) berechnet werden.

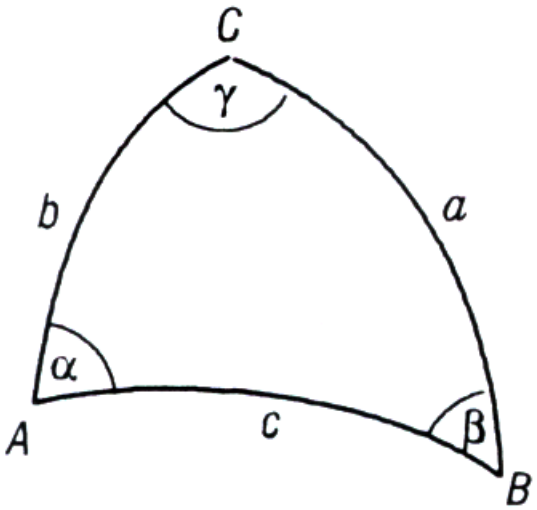

Abb. 07: Schiefwinkliges sphärisches Dreieck
(Quelle: Bartsch 1999, S. 200)

Abstände bezogen auf Erdoberfläche in Grad:

b = Ort zu Äquator ($Breitengrad_{Ort}$)

c = Ort zu Satellit auf Äquator ($Längengrad_{Ort} - Längengrad_{Sat}$)

a = Ort zu Satellit

$$b = Breitengrad_{Ort}$$

$$c = Längengrad_{Ort} - Längengrad_{Sat}$$

$$\cos a = \cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha$$

$$a = \arccos\left(\cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha\right)$$

$$\cos c = \cos a \cdot \cos b + \sin a \cdot \sin b \cdot \cos \gamma$$

$$\gamma = 180 - \arccos\left(\frac{\cos c - \cos a \cdot \cos b}{\sin a \cdot \sin b}\right)$$

3.3.1.1 Berechnung des Azimutwinkels für den Flughafen Graz

Unter der Berücksichtigung der Geo-Koordinaten vom Flughafen Graz aus Tab. 02 ergeben sich folgende Werte:

$b = 46{,}993°$

$c = 19{,}2° - 15{,}44° = 3{,}76°$

$\cos a = \cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha$

$a = \arccos(\cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha)$

$a = 47{,}108°$

$\cos c = \cos a \cdot \cos b + \sin a \cdot \sin b \cdot \cos \gamma$

$\gamma = 180 - \arccos\left(\frac{\cos c - \cos a \cdot \cos b}{\sin a \cdot \sin b}\right)$

$\gamma = 174{,}865°$

Ausgehend vom geografischen Norden (0°) muss ein Azimutwinkel von +185,135° eingestellt werden um den Satelliten zu finden.

Um die mechanischen Wege so gering wie möglich zu gestalten kann in diesem Fall auch um -174,865° gedreht werden.

Eine allgemeine Bedingung um den kürzesten Weg zu bestimmen kann durch $if(\gamma > 181)\ \{\gamma = \gamma - 360\}$ ausgedrückt werden.

3.3.1.2 Berechnung des Azimutwinkels für den Flughafen Innsbruck

Um die stationären Unterschiede bezüglich des Azimutwinkels zu verdeutlichen wird der Wert vom Flughafen Graz mit dem vom Flughafen Innsbruck verglichen. Der Flughafen Innsbruck liegt zirka 486 km westlich vom Flughafen Graz und auf ähnlichem Breitengrad. Die Werte für Längen- und Breitengrad werden von Tab. 03 entnommen.

Tab. 03: Längen- und Breitengrad Flughafen Innsbruck

Längengrad:	11° 20' 49'' E = 11,347° E
Breitengrad:	47° 15' 38'' N = 47,261° N

(Quelle: http://www.skyscanner.at/flughaefen/inn/innsbruck-flughafen.html)

$b = 47{,}458°$

$c = 19{,}2° - 11{,}347° = 10{,}648°$

$\cos a = \cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha$

$a = \arccos\left(\cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos \alpha\right)$

$a = 48{,}04°$

$\cos c = \cos a \cdot \cos b + \sin a \cdot \sin b \cdot \cos \gamma$

$$\gamma = 180 - \arccos\left(\frac{\cos c - \cos a \cdot \cos b}{\sin a \cdot \sin b}\right)$$

$\gamma = 168{,}464°$

Da der Flughafen Innsbruck gegenüber dem Satelliten weiter westlich liegt als der Flughafen Graz muss ein um 6,4° geringerer Azimutwinkel eingestellt werden.

3.4 Elevationswinkel

Senkrecht auf den Azimutwinkel steht der Elevationswinkel (Höhenwinkel). Der Elevationswinkel erreicht am Äquator sein Maximum und nimmt mit zunehmendem Breitengrad ab. Die Extremwerte liegen bei 0 Grad an den Polen und 90 Grad am Äquator.

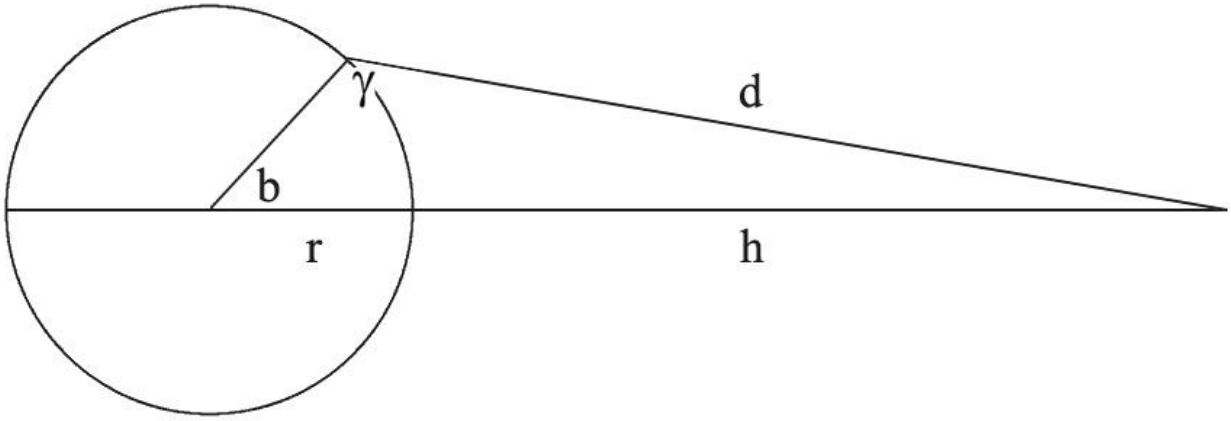

Abb. 08: Skizze zur Berechnung des Elevationswinkels

(gezeichnet von Unterluggauer)

r = Erdradius

h = Flughöhe des Satelliten

d = Entfernung Ort zu Satellit

b = Winkel vom Äquator zum Ort (Breitengrad)

Aus den gegebenen Größen (r, h und b) kann mittels des Cosinussatzes die Entfernung (d) von Ort zu Satellit berechnet werden. Daraus wird wiederum mit Hilfe des Cosinussatzes der gesuchte Elevationswinkel berechnet.

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

3.4.1.1 Berechnung des Elevationswinkels für den Flughafen Graz

Unter der Berücksichtigung der Geo-Koordinaten vom Flughafen Graz aus Tab. 02 ergeben sich folgende Werte:

h = 35.880 km

r = 6.378 km

b = 46,993°

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$d =$ 38.193,5 km

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

$$\delta = 35{,}993°$$

Der Elevationswinkel beträgt somit 35,993 Grad.

Auch beim Elevationswinkel werden zwei Orte mit ähnlichem Längengrad aber unterschiedlichem Breitengrad vergleichen. Hier wird der Flughafen Klagenfurt im Süden und der Flughafen Linz im Norden gewählt.

3.4.1.2 Berechnung des Elevationswinkels für den Flughafen Klagenfurt

Für die Berechnung wurden die Werte für Längen- und Breitengrad aus Tab. 04 herangezogen.

Tab. 04: Längen- und Breitengrad Flughafen Klagenfurt

Längengrad:	14° 20' 30" E = 14,342°
Breitengrad:	46° 38' 35" N = 46,643°

(Quelle: http://www.skyscanner.at/flughaefen/klu/klagenfurt-flughafen.html)

Daraus ergeben sich folgende Werte:

h = 35.880 km

r = 6.378 km

b = 46,643°

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$$d = 38.162{,}1 \text{ km}$$

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

$$\delta = 36{,}3773°$$

Der Elevationswinkel für den Flughafen Klagenfurt beträgt 36,38 Grad.

3.4.1.3 Berechnung des Elevationswinkels für den Flughafen Linz

Unter Zuhilfenahme der Werte aus Tab. 05 wurde der Elevationswinkel für den Flughafen Linz berechnet.

Tab. 05: Längen- und Breitengrad Flughafen Linz

Längengrad:	14° 11' 36" E = 14,193°
Breitengrad:	48° 14' 03" N = 48,234°

(Quelle: http://www.skyscanner.at/flughaefen/lnz/linz-flughafen.html)

h = 35.880 km

r = 6.378 km

b = 48,234°

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$d =$ 38.306,2 km

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

$$\delta = 34{,}6321°$$

Es ergibt sich für den Flughafen Linz ein Elevationswinkel von 34,63 Grad.

Da Klagenfurt weiter im Süden liegt als Linz und somit dem Äquator und der Umlaufbahn des Satelliten näher liegt, ist der Elevationswinkel um zirka 1,75° höher.

3.4.1.4 Einfluss der Seehöhe auf den Elevationswinkel

Die Seehöhe hat einen Einfluss auf den Elevationswinkel, da mit steigender Höhe der Winkel zum Satelliten abnimmt. Als Extreme werden für die Berechnung der Meeresspiegel sowie der höchste Berg verwendet. Der höchste Berg gemessen von der Meeresoberfläche ist mit 8.848 m der Mount Everest (vgl. Alpenverein Freistadt).

3 Winkelberechnung

Berechnung 1 (Meeresspiegel - 0 Meter Seehöhe):

h = 35.880 km + 0 km = 35.880 km

r = 6.378 km

b = 46,993°

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$$d = 38.193{,}5 \text{ km}$$

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

$$\delta = 35{,}993°$$

Berechnung 2 (Berg - 8848 Meter Seehöhe):

h = 35.880 km + 8,8 km = 35.888,8 km

r = 6.378 km

b = 46,993°

$$d^2 = r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b$$

$$d = \sqrt{r^2 + (h+r)^2 - 2r(h+r)\cdot\cos b}$$

$$d = 38.188{,}32 \text{ km}$$

$$(h+r)^2 = d^2 + r^2 - 2dr\cdot\cos\delta$$

$$\delta = \arccos\left(\frac{d^2 + r^2 - (h+r)^2}{2dr}\right) - 90°$$

$$\delta = 35{,}982°$$

Graz liegt 353 Meter über dem Meeresspiegel. Daraus ergibt sich ein Elevationswinkel von $\delta = 35{,}992°$.

Es wird ersichtlich, dass die Seehöhe einen Einfluss auf den Elevationswinkel hat, dieser jedoch so gering ist, dass er vernachlässigt werden kann. Der Höhenwinkel ist somit nur von der geografischen Breite und Länge abhängig.

3.5 Orthogonale Koordinatentransformation im Raum

Jeder Vektor bzw. Körper kann im Koordinatensystem elementar um die x-, y- und z-Achse beliebig gedreht werden.

Der Vektor vor der Drehung wird als $r_1 = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$ und der Vektor nach der Drehung als $r_2 = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ bezeichnet.

Bei der Drehung um die x-Achse ändert sich die x-Koordinate nicht.
Die Rotationsmatrix oder Drehmatrix sowie die Koordinaten nach der Drehung um die x-Achse werden wie folgt beschrieben.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_x(\alpha) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

(Bartsch 1999, S. 234).

Im Zuge der Drehung um die y-Achse bleiben die y-Koordinaten konstant und der Rest dreht sich in der x- und z-Ebene um den Winkel β.
Die Drehmatrix sowie die Koordinaten nach der Drehung um die y-Achse werden wie folgt beschrieben.

$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_y(\beta) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

(Bartsch 1999, S. 234).

Wird ein Körper um die z-Achse gedreht, bleiben die z-Koordinaten konstant. Die Drehmatrix sowie die Koordinaten nach der Drehung um die z-Achse werden wie folgt beschrieben.

$$R_z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_z(\gamma) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

(Bartsch 1999, S. 234).

Die Versuchsanordnung beschränkt sich auf die Drehung der x- und y-Achse da Änderungen im Bereich der z-Achse durch den Azimutwinkel berücksichtigt werden.

Um die Berechnung zu vereinfachen kann anstatt der getrennten Drehung um die x- und y-Achse eine Mehrfachdrehung durchgeführt werden. Dabei werden beide Drehungen in einer Rotationsmatrix zusammengefasst.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \left(R_x(\alpha) \cdot R_y(\beta)\right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$R_x(\alpha) \cdot R_y(\beta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} =$$

$$= \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ \sin(\alpha)\sin(\beta) & \cos(\alpha) & -\sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ \sin(\alpha)\sin(\beta) & \cos(\alpha) & -\sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

3.6 Ausgleichswinkel

Die reale Lage beschreibt den aktuellen Zustand der Versuchsanordnung. Sie kann zum Beispiel auf einem mit dem Winkel α bzw. β geneigtem schiefen Untergrund stehen. Die virtuelle Lage ist die horizontale Lage mit den Winkeln α und β gleich Null.

Für die Bestimmung des Winkels zwischen der realen und der neuen virtuellen Ebene soll der Vektor um den Winkel α in der y-z-Ebene sowie um den Winkel β in der x-z-Ebene gedreht werden.

$$r_1 = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$r_2 = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ \sin(\alpha)\sin(\beta) & \cos(\alpha) & -\sin(\alpha)\cos(\beta) \\ -\cos(\alpha)\sin(\beta) & \sin(\alpha) & \cos(\alpha)\cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Nun muss der Winkel zwischen dem ursprünglichen Vektor r_1 und dem Vektor nach der Drehung r_2 berechnet werden.

$$r_1 = \begin{pmatrix} u \\ v \\ w \end{pmatrix}, \quad r_2 = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Betrag der Ortsvektoren:

$$|r_1| = \sqrt{u^2 + v^2 + w^2}$$

$$|r_2| = \sqrt{x^2 + y^2 + z^2}$$

(Bartsch 1999, S. 209).

Der Winkel zwischen zwei Normalvektoren entspricht dem Winkel zwischen den beiden Ebenen.

$$\cos\varepsilon = \frac{r_1 \cdot r_2}{|r_1| \cdot |r_2|} = \frac{u \cdot x + v \cdot y + w \cdot z}{\left|\sqrt{u^2 + v^2 + w^2}\right| \cdot \left|\sqrt{x^2 + y^2 + z^2}\right|}$$

$$\varepsilon = \arccos\frac{r_1 \cdot r_2}{|r_1| \cdot |r_2|} = \arccos\frac{u \cdot x + v \cdot y + w \cdot z}{\left|\sqrt{u^2 + v^2 + w^2}\right| \cdot \left|\sqrt{x^2 + y^2 + z^2}\right|}$$

(Bartsch 1999, S. 237).

3.7 Elevationswinkel unter Berücksichtigung des Ausgleichswinkels

Der errechnete Ausgleichswinkel kann nicht direkt zum Elevationswinkel gerechnet werden. Je nach aktuellem Azimutwinkel variiert der Ausgleichswinkel zwischen 0 und ε Grad.

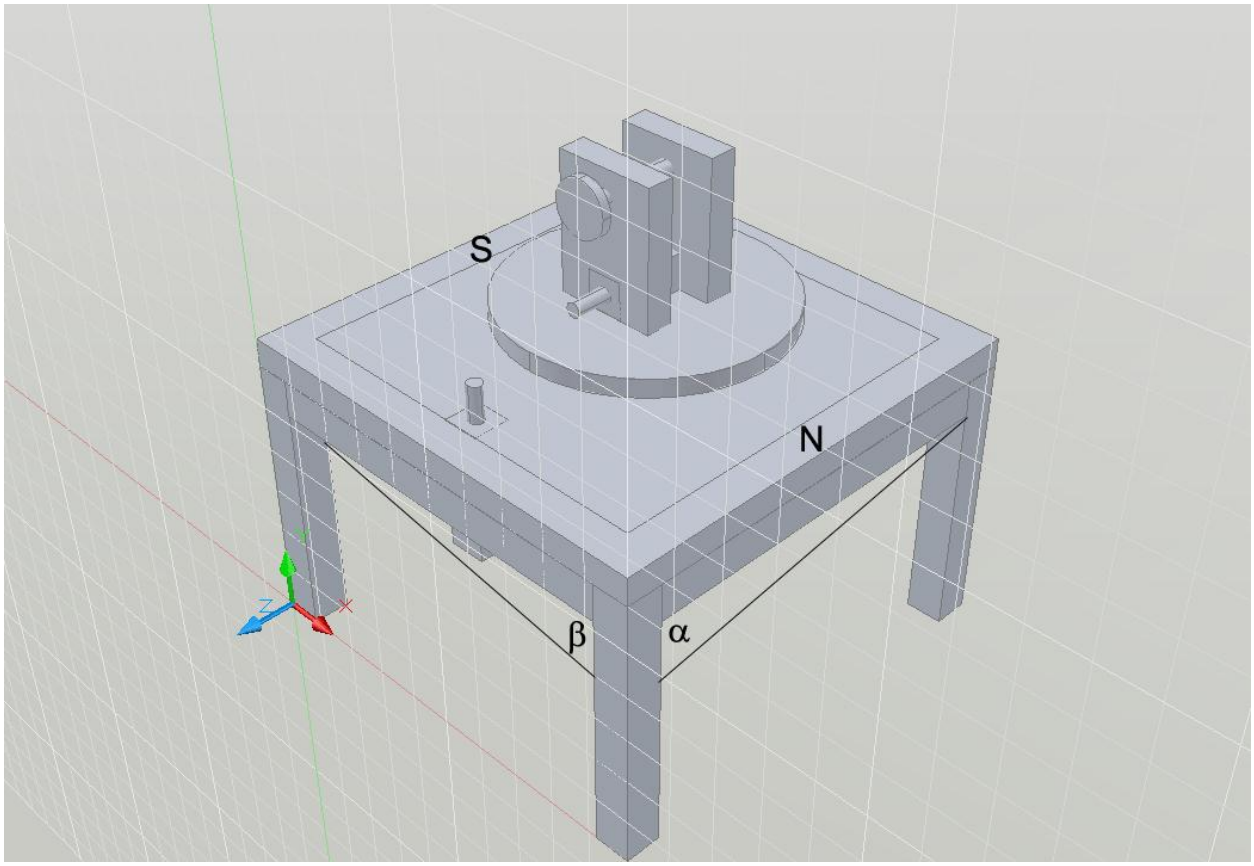

Abb. 09: Ausgleichswinkel

(gezeichnet von Unterluggauer)

Die im folgenden Teil beschriebenen Sonderfälle beziehen sich auf die in Abb. 09 dargestellte Versuchsanordnung mit Ausrichtung der Antennenhalterung in Nord-Süd-Richtung.

Sonderfall 1:

Wenn α ungleich Null und β gleich Null ist, beeinflusst der Ausgleichswinkel den Elevationswinkel nicht.

Sonderfall 2:

In diesem Fall ist der Winkel α gleich Null und der Winkel β ungleich Null. Der Ausgleichswinkel fließt zu 100% in den Elevationswinkel ein.

Sonderfall 3:

Für den Fall dass α und β gleich groß sind, wird die Hälfte des Ausgleichswinkels in die Berechnung mit einbezogen.

Allgemein gilt, dass der Einfluss des Ausgleichswinkels vom Verhältnis der Winkel α und β abhängt.

Im ersten Schritt wird der theoretische Azimutwinkel auf Basis von α und β berechnet bei welchem der Ausgleichswinkel zu 100% in den Elevationswinkel einfließt. 90 Grad von diesem Winkel abweichend geht der Einfluss des Ausgleichswinkels gegen Null. In diesem Bereich von 0 bis 90 Grad liegt der reale, bereits eingestellte Azimutwinkel. Jetzt muss noch berechnet werden, wie viel Prozent der reale Azimutwinkel dem theoretischen Azimutwinkel entspricht.

Genau diese Prozentzahl angewandt auf den Ausgleichswinkel wird dann zum Elevationswinkel dazugezählt oder abgezogen.

4 Software

Für die Programmierung der *B&R* Komponenten wird das Automation Studio (Abb. 10) in der Version 3.0.80.25 verwendet.

Um einen Überblick über die Produkte aus dem Hause B&R zu bekommen, sowie Einblicke in die Programmierung zu erlangen wurde ein Schulung in Anspruch genommen.

Diese Schulung setzt sich aus 3-wöchigen Trainings und Workshops aus dem Bereich „Automation Studio Control“ und „Automation Studio Motion“ zusammen.

Im Folgenden werden Produktbezeichnungen von Komponenten aus dem Labor des Instituts für Kommunikationsnetze und Satellitenkommunikation verwendet.

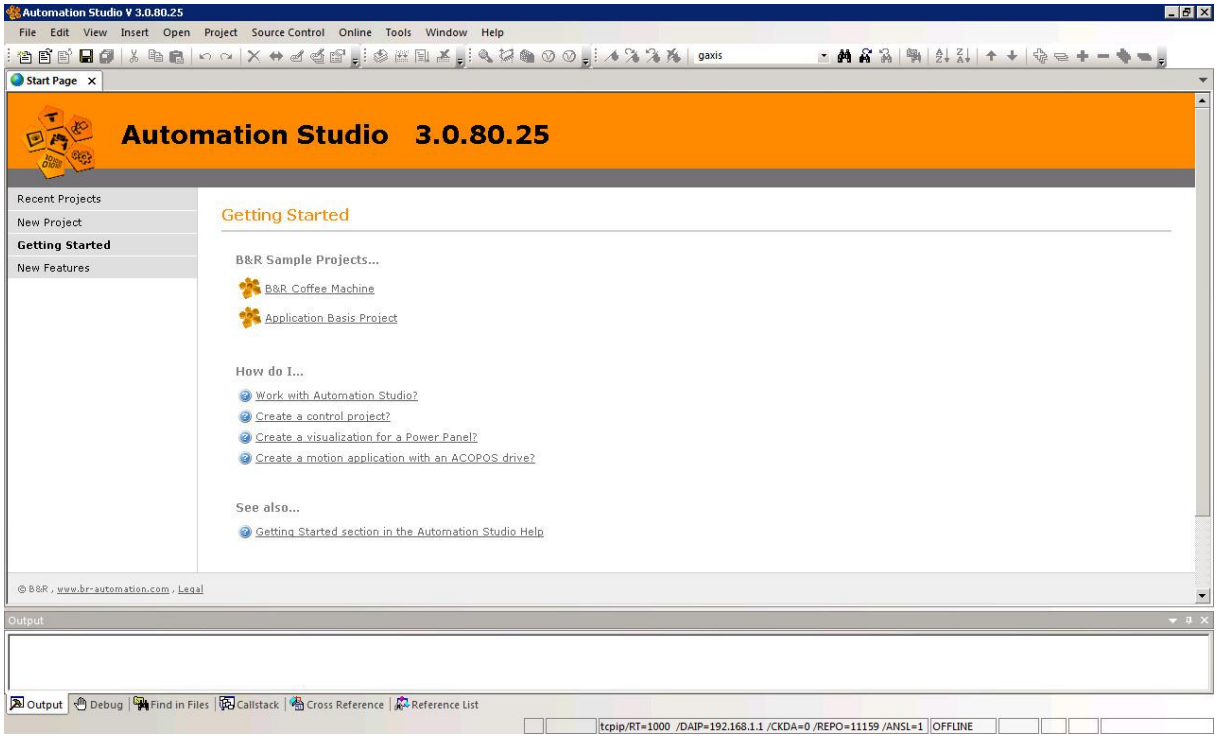

Abb. 10: Automation Studio

(Quelle: Automation Studio Version V 3.0.80.25)

4.1 Onlineverbindung und neues Projekt erstellen

Der erste, in Abb. 11 dargestellte, Schritt ist die Herstellung einer Onlineverbindung via Ethernet zur X20 CPU (X20CP3486). Dazu wird unter „Online Settings“ mit „Add TCPIP Connection“ eine neue TCPIP Verbindung hinzugefügt. Als IP wird im Falle der B&R – Ausstattung an der TU-Graz 192.168.1.1 eingetragen und anschließend via „Connect“ verbunden.

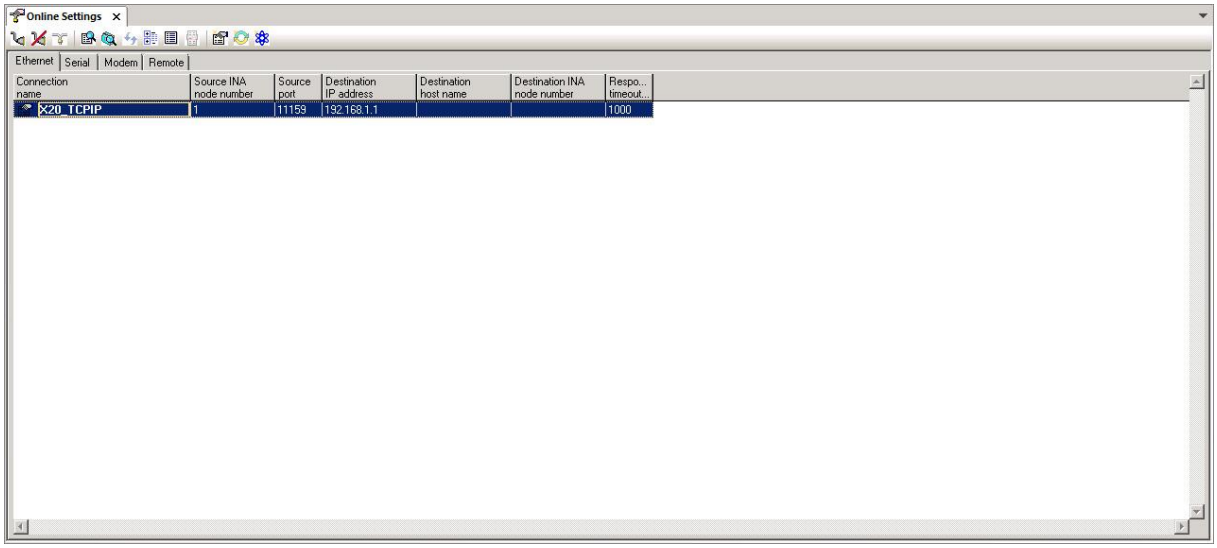

Abb. 11: Automation Studio Online Settings

(Quelle: Automation Studio Version V 3.0.80.25)

Am Notebook bzw. PC muss die IP 192.168.0.2 eingestellt werden. Die erfolgt unter den Windows Netzwerk- und Freigabeeinstellungen welche in Abb. 12 zu sehen sind.

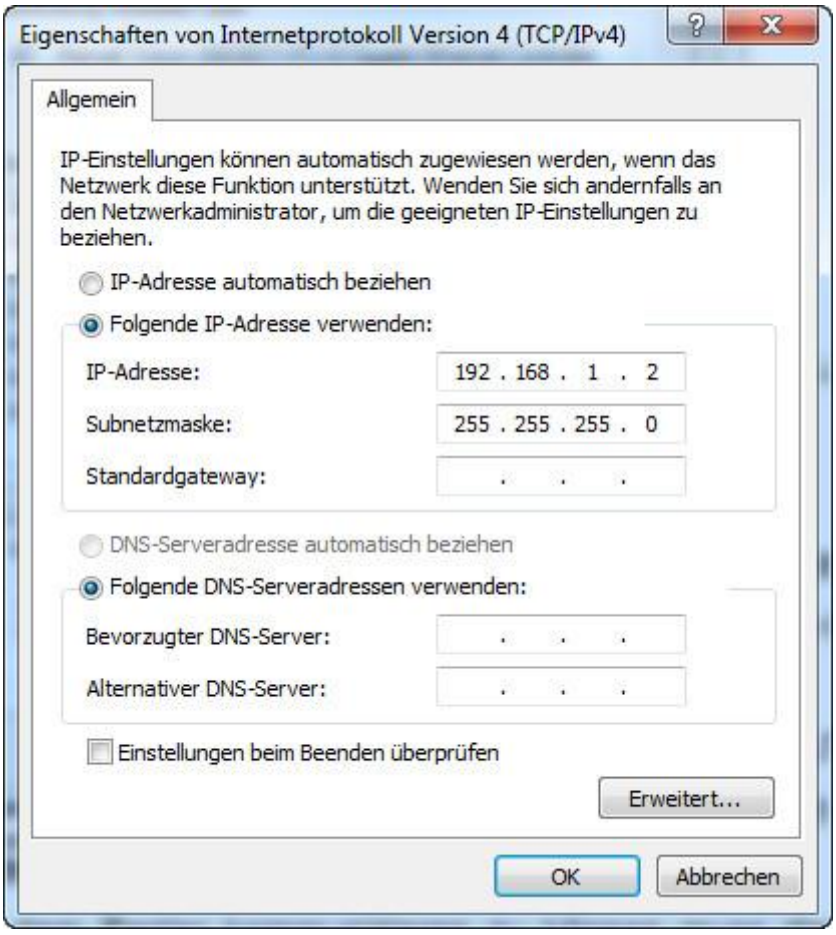

Abb. 12: Eigenschaften von Internetprotokoll Version 4(TCP/IPv4)

(Quelle: Netzwerk- und Freigabecenter Windows 7)

Anschließend wird ein neues Projekt erstellt. Es wird „AUTO-SAT“ genannt. Mit der Auswahl von „Identify control system online“ erkennt das Automation Studio automatisch die X20 CPU (X20CP3486).

Um mit dem Regler kommunizieren zu können muss dieselbe IP auch im Bereich „PLC1.CPU IF2 Ethernet Configuration“ eingetragen werden (Abb. 12).

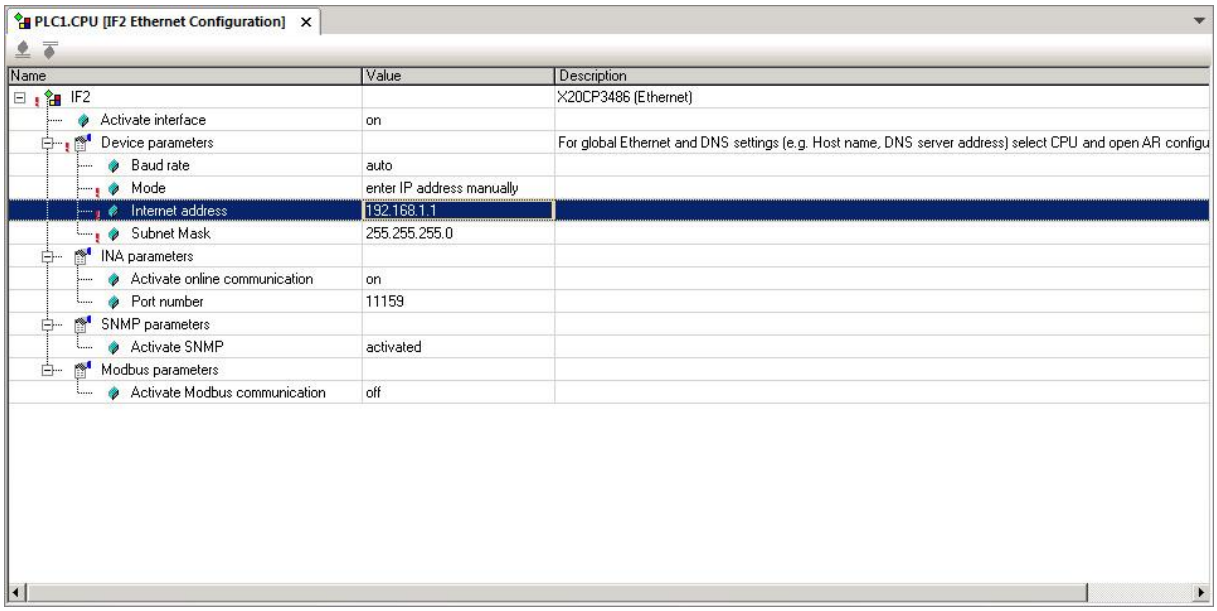

Abb. 13: Automation Studio IF2 Ethernet Configuration

(Quelle: Automation Studio Version V 3.0.80.25)

Jetzt fehlen noch die zwei ACOPOS – Servoverstärker welche über eine interne Logik verfügen. Diese Logik nennt sich NC-Betriebssystem, welche die Parameter am ACOPOS verwaltet und über alle Komponenten verfügt, um Positionierungen durchführen zu können (vgl. B&R 2007a, S. 9).

Im Reiter „PLC1.CPU POWERLINK“ wird mit „Insert“ der ACOPOS 8V1010.00-2 hinzugefügt. Als Knotennummer muss die Nummer 17 für Achse eins und 18 für die Achse zwei eingegeben werden. Weitere notwendige Einstellungen sind 8AC112.60-1 als Plug-in card (Powerlink V1) sowie der *B&R* – Motor 8MSA2M.E4-42 Revision D0. In Abb. 13 sieht man die zwei hinzugefügten ACOPOS.

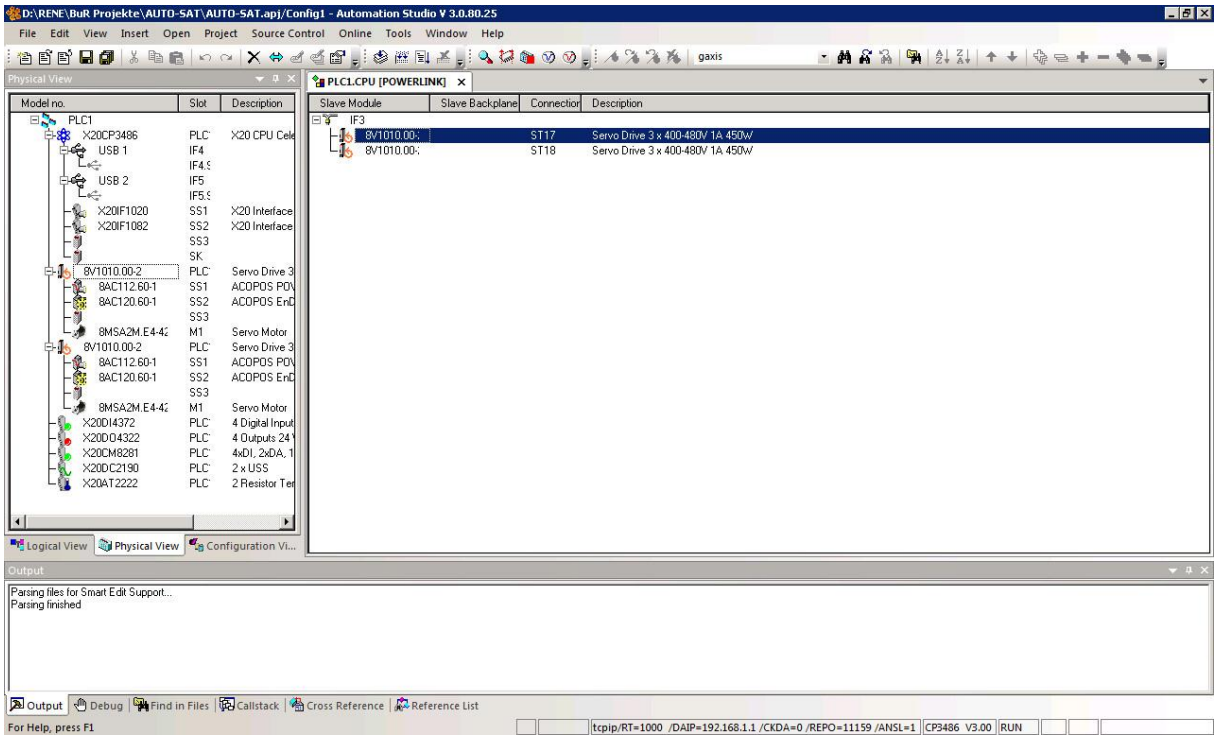

Abb. 14: Automation Studio POWERLINK

(Quelle: Automation Studio Version V 3.0.80.25)

4.2 Parametrisierung der Achsen

Nun sind beide Achsen gaxis01 für den Azimut und gaxis02 für die Elevation hinzugefügt und es kann mit der Parametrisierung begonnen werden.

In der Ansicht „Local View“ befindet sich die Datei gaxis01i.ax. Hier sind alle für die Achse 1 relevanten Parameter gespeichert. Wichtig in diesem Fall ist die Definition der Units. Die Anzahl der Units gibt an, mit wie vielen Schritten die Motorachse eine volle Umdrehung durchführt. Aus der Berechnung der Länge des Antriebsriemens zur Ansteuerung der Elevation ergibt sich ein Übersetzungsverhältnis von ca. 10,4. Das heißt 10,4 Umdrehungen am Motor entsprechen einer Umdrehung des Drehtellers.
Das Ziel war es, eine volle Umdrehung des Drehtellers mit 36.000 Units (für 360 Grad) gleich zu setzen. Somit wäre 1 Unit am Motor 0,01 Grad am Drehteller.

Um dies ganz genau einzustellen, wird eine Markierung am Drehteller angebracht und im Achsentestmodus (Abb. 15 und 17) des Automation Studios 36.000 als

Verfahrensweg eingestellt. Der Testmodus kann über den entsprechenden ACOPOS im „Physical View“ aufgerufen werden.

Danach werden die Units in der Datei gaxis01i.ax entsprechend angepasst und letztendlich ergab sich ein Wert von 3.450 Units (Abb. 14).
Das genaue Verhältnis von Achse zu Drehscheibe beträgt somit 3.450:36.000 bzw. 1:10,435.

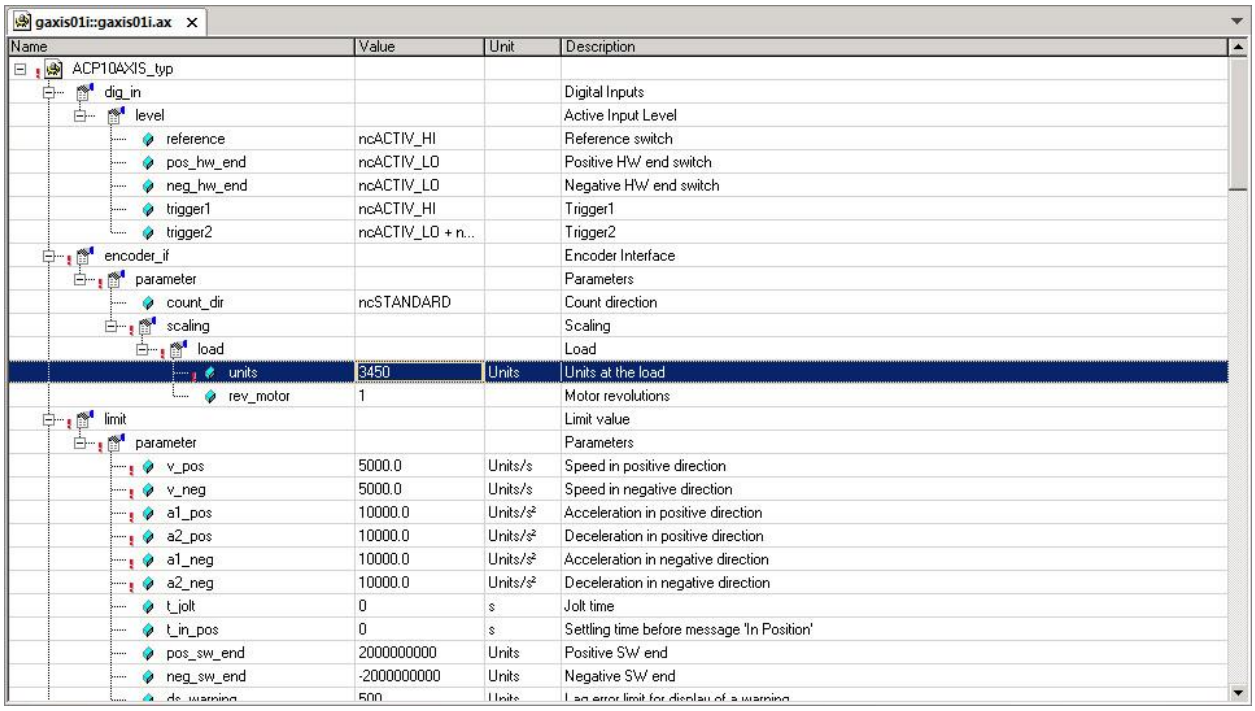

Abb. 15: Automation Studio gaxis01.ax
(Quelle: Automation Studio Version V 3.0.80.25)

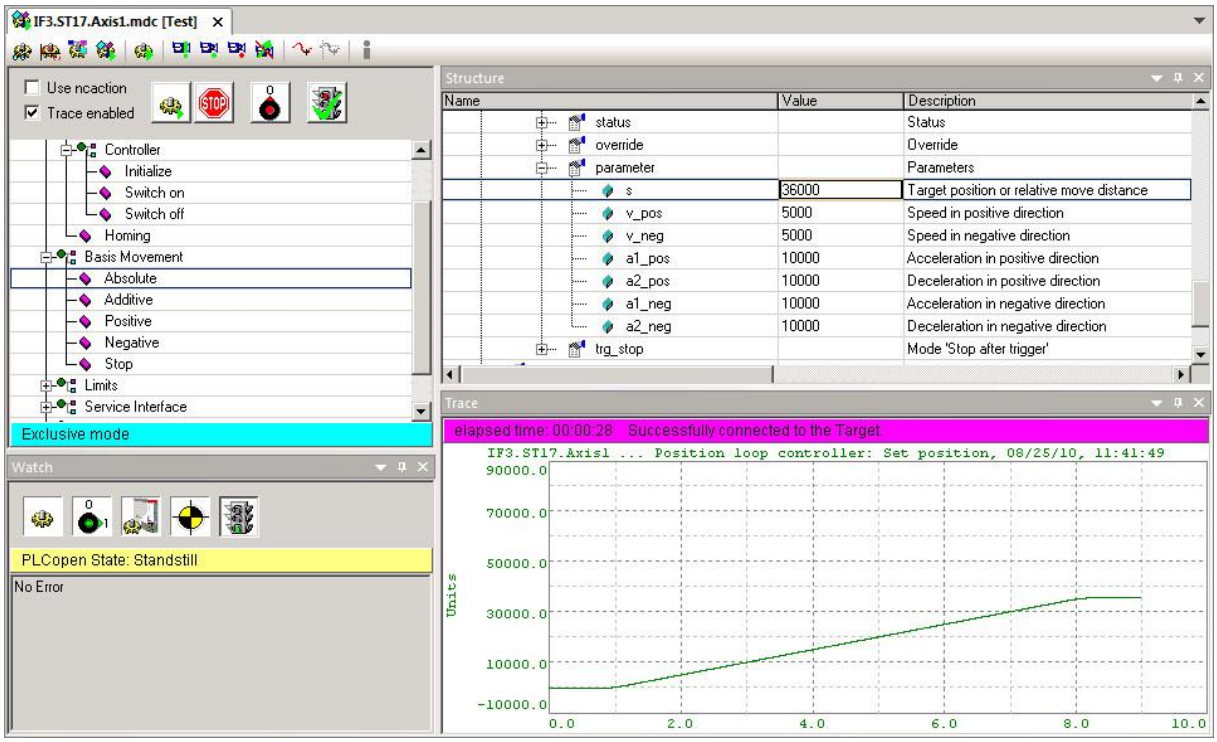

Abb. 16: Automation Studio Axis1.mdc Test

(Quelle: Automation Studio Version V 3.0.80.25)

In der Datei gaxis02i.ax werden die Units für eine Umdrehung der zweiten Achse definiert. Eine Umdrehung der Welle für die Änderung der Elevation wird mit 3.600 Units (für 360 Grad) festgelegt und dieser Wert im Testmodus als Verfahrensweg eingetragen. 0,1 Grad entspricht 1Unit. Auf Grund des Übersetzungsverhältnisses Motor zu Welle von zirka zwei ergab sich schlussendlich ein Unit-Wert von 1.684 in der Datei gaxis02i.ax (Abb. 16).

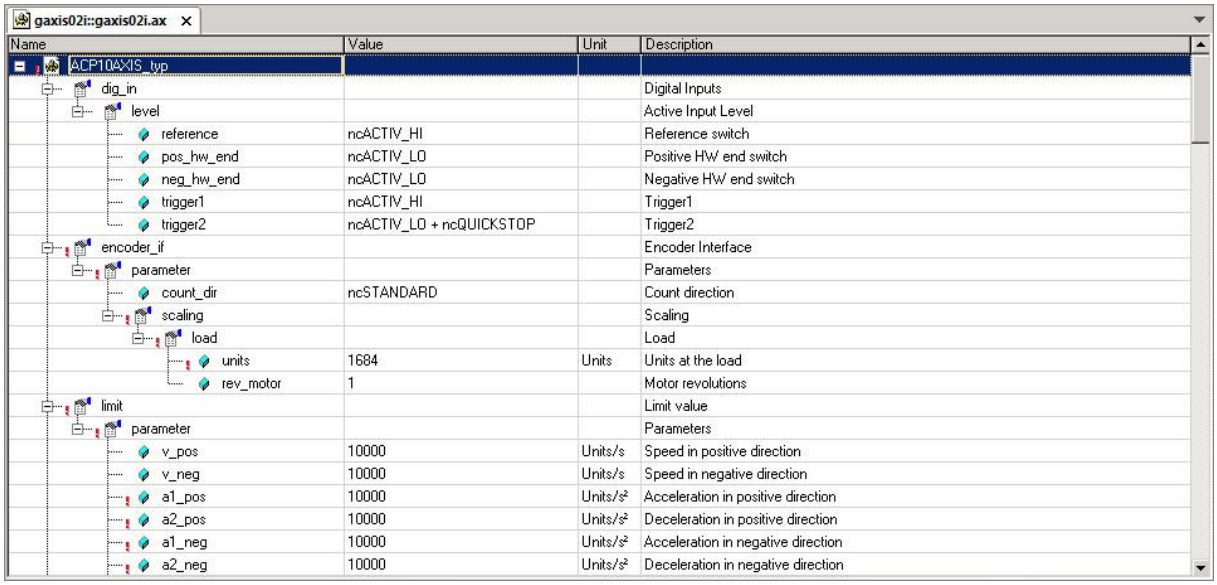

Abb. 17: Automation Studio gaxis02.ax

(Quelle: Automation Studio Version V 3.0.80.25)

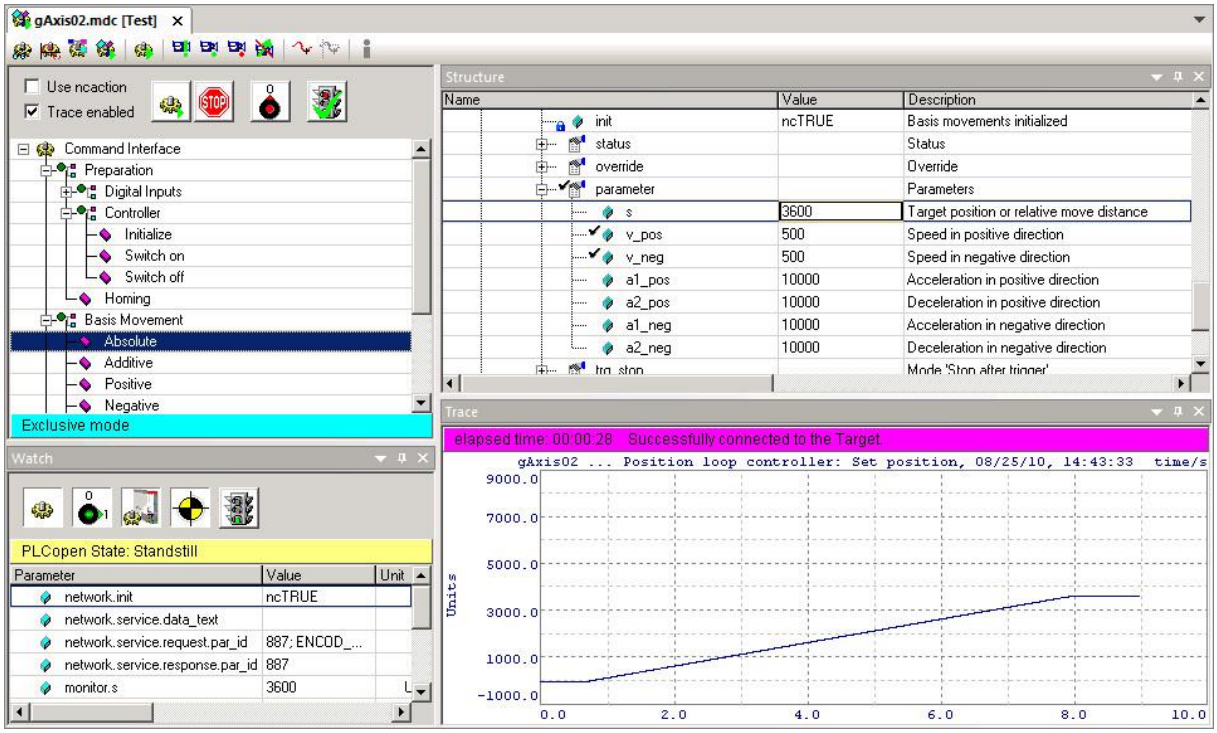

Abb. 17: Automation Studio Axis2.mdc Test

(Quelle: Automation Studio Version V 3.0.80.25)

4.3 Reglereinstellung

Mit Hilfe des Lagereglers (Position Controller) und dem Drehzahlregler (Speed Controller) kann der Regler nach den Wünschen und Anforderungen der Anwendung eingestellt werden.

Im Falle der Elevation der Antenne soll die Masse möglichst wenig durch zum Beispiel Wind bewegt werden können. Deshalb muss der Regler so hart wie möglich eingestellt werden.

Zum Einstellen des Reglers ist die Proportionalverstärkung (kv) der wichtigste Parameter. Dieser soll möglichst hoch gewählt werden, ohne dass das System zu schwingen beginnt (vgl. B&R 2007c, S. 30).

In Abb. 18 werden die Parameter zu niedrig gewählt. Aus diesem Grund setzt der Motor bei zu großer Belastung aus.

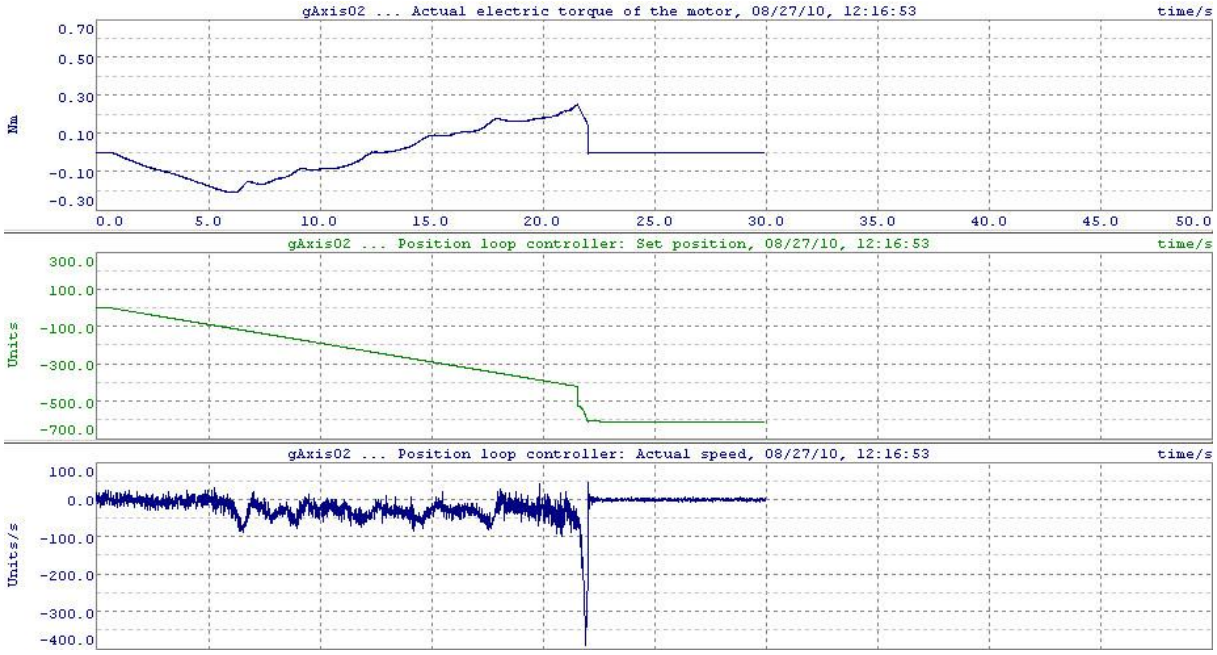

Abb. 18: Automation Studio position.kv = 100, speed.kv = 0,1

(Quelle: Automation Studio Version V 3.0.80.25)

In Abb. 19 sind die Parameter zu hoch gewählt und das System beginnt zu schwingen.

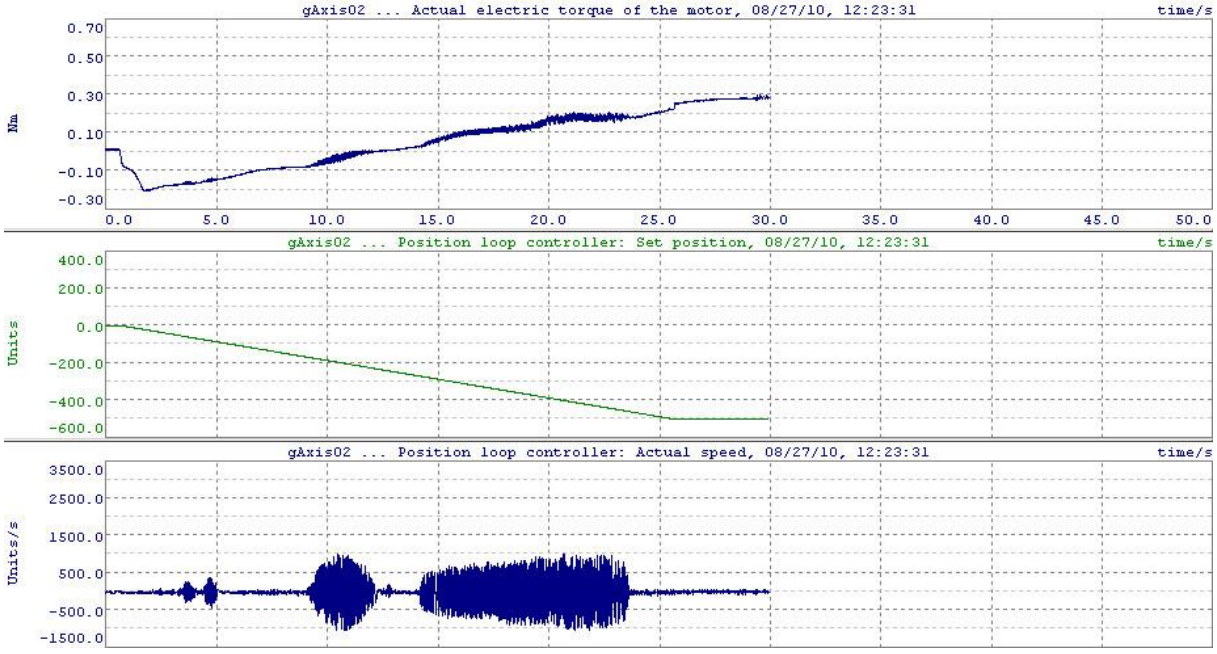

Abb. 19: Automation Studio position.kv = 2000, speed.kv = 0,2

(Quelle: Automation Studio Version V 3.0.80.25)

Anhand mehrerer Tests werden die Werte je nach Belastung für position.kv mit 800-1.500 und speed.kv mit 0.2 festgelegt. Der Verlauf ist in Abb. 20 zu sehen.

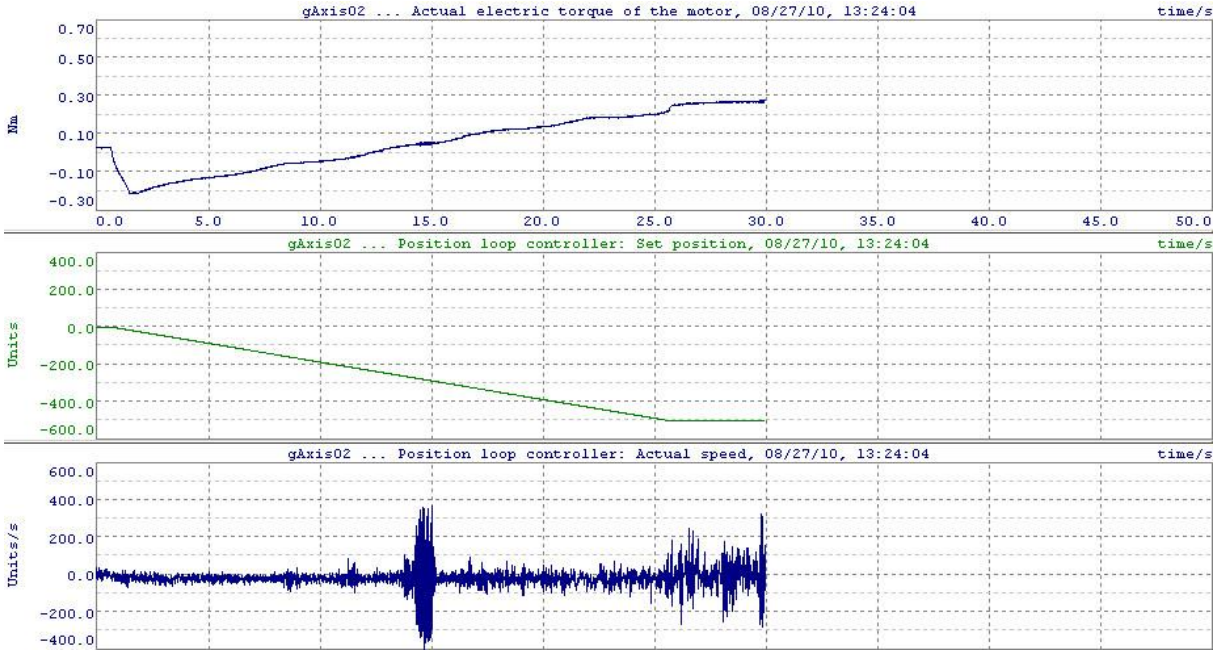

Abb. 20: Automation Studio position.kv = 800, speed.kv = 0,2

(Quelle: Automation Studio Version V 3.0.80.25)

4.4 Sollwertgenerator

Der Sollwertgenerator hat nach Befehlserteilung die Aufgabe, ein Bewegungsprofil zu erstellen (vgl. B&R 2007c, S. 11).

In Abb.21 ist zu sehen, dass die Beschleunigung (Acceleration) Sprünge besitzt. Diese werden als Ruck bezeichnet. Ein derartiges Verhalten ist sowohl für den Motor als auch für die Mechanik sehr belastend (vgl. B&R 2007c, S. 12).

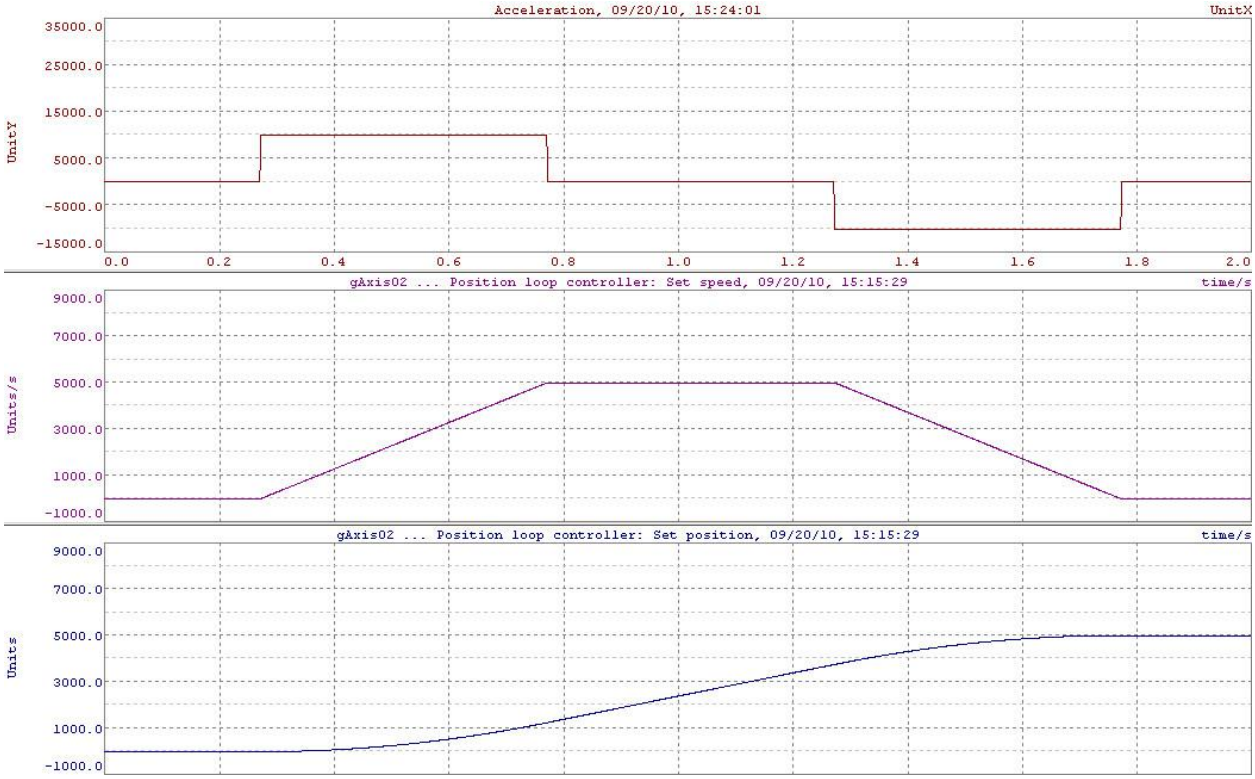

Abb. 21: Automation Studio t_jolt = 0s

(Quelle: Automation Studio Version V 3.0.80.25)

Ein solches Verhalten kann durch die Ruckbegrenzung vermieden werden. Dabei wird am Anfang die Geschwindigkeit langsamer erhöht und am Ende langsamer vermindert. Aus dem rechteckigen Beschleunigungsprofil (Abb. 21) wird nun ein trapezförmiges (Abb. 22). Mit Hilfe der Ruckfilterzeit wird diese Verzögerung eingestellt. In diesem Fall wird die Ruckfilterzeit (t_jolt) auf 0,13 Sekunden eingestellt (vgl. B&R 2007, S. 13).

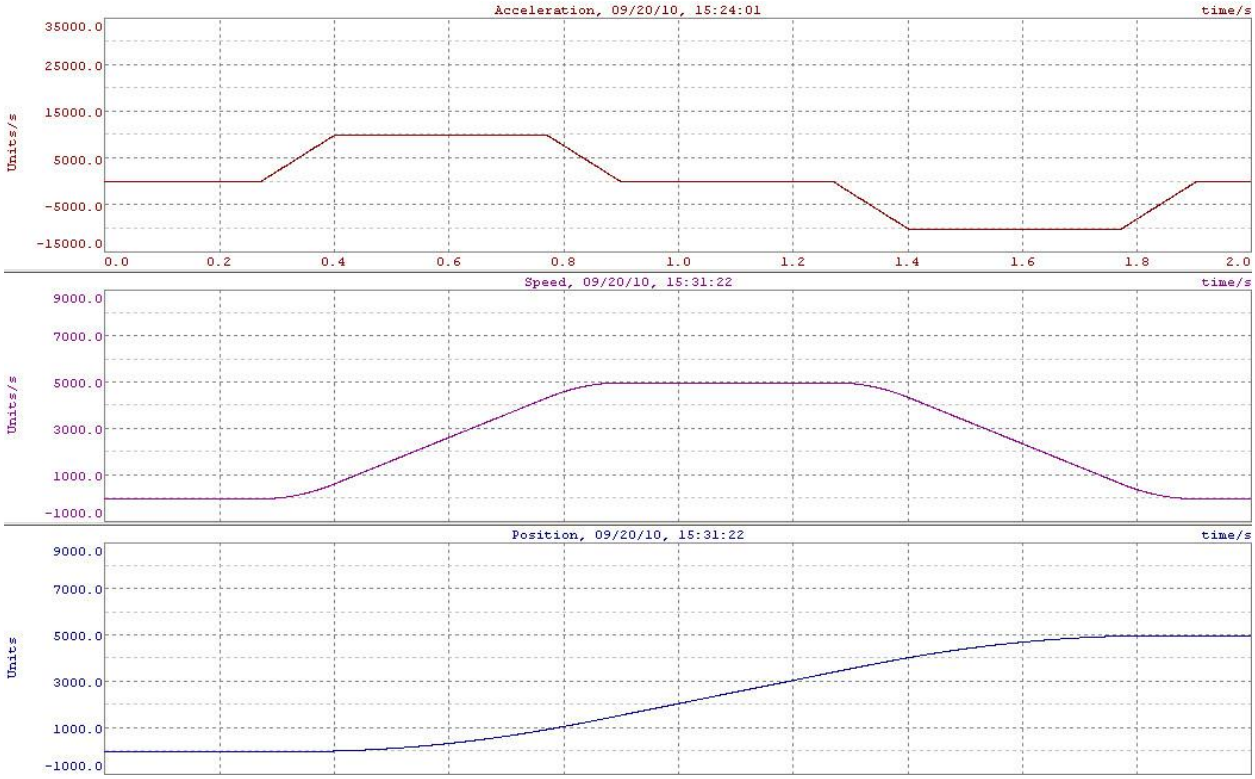

Abb. 22: Automation Studio t_jolt = 0,13s

(Quelle: Automation Studio Version V 3.0.80.25)

Name	Value	Unit	Description
limit			Limit value
parameter			Parameters
v_pos	5000.0	Units/s	Speed in positive direction
v_neg	5000.0	Units/s	Speed in negative direction
a1_pos	10000.0	Units/s²	Acceleration in positive direction
a2_pos	10000.0	Units/s²	Deceleration in positive direction
a1_neg	10000.0	Units/s²	Acceleration in negative direction
a2_neg	10000.0	Units/s²	Deceleration in negative direction
t_jolt	0.13	s	Jolt time
t_in_pos	0	s	Settling time before message 'In Position'
pos_sw_end	2000000000	Units	Positive SW end
neg_sw_end	-2000000000	Units	Negative SW end
ds_warning	500	Units	Lag error limit for display of a warning
ds_stop	1000	Units	Lag error limit for stop of a movement
a_stop	1.0e30	Units/s²	Acceleration limit for stop of a movement

Abb. 23: Automation Studio t_jolt

(Quelle: Automation Studio Version V 3.0.80.25)

4.5 Programmierung

Als Programmiersprachen werden „C“ sowie „Strukturierter Text (ST)“ gewählt.
„Strukturierter Text (ST)“ ist eine textuelle Hochsprache und wird im Bereich der Automatisierung angewandt. Leicht verständliche Standardkonstrukte stellen eine effiziente und schnelle Art der Programmierung dar (vgl. B&R 2007b, S.6).
Zu den Eigenschaften von „ST“ zählen selbsterklärende und flexible Anwendung sowie leichte Erlernbarkeit bei bereits Vorhandenem Wissen in „PASCAL“ oder „C“ (vgl. B&R 2007b, S.7).

4.5.1 Programmablauf

Der Programmablauf setzt sich aus den folgenden sieben Punkten zusammen.

1.) Eingabe von Längengrad und Breitengrad
2.) Eingabe der Nordrichtung in Grad
3.) Eingabe der Winkel von x- und y-Achse in Grad
4.) Berechnung von Azimutwinkel zum Zielpunkt
5.) Einführen einer virtuellen Ebene zur Bestimmung des Winkels zwischen den Normalvektoren der realen und virtuellen Ebene.
6.) Berechnung der Elevation zum Zielpunkt
7.) Ausgabe der Parameter in Grad.

4.5.2 Programmcode für den Azimutwinkel in „Strukturierter Text“

Der Progammcode zur Ansteuerung der Achse, welche für die Änderung des Azimutwinkels verantwortlich ist befindet sich im Anhang A.

4.5.3 Programmcode für die Elevation in C

Der Progammcode, welcher für die Ansteuerung der Azimut-Achse verantwortlich ist, befindet sich im Anhang B.

4.6 Ergebnisse aus dem Programm

Das Ergebnis aus der Berechnung für den Azimutwinkel kann direkt in die Motoreinheiten (Units) umgerechnet werden. 360 Grad entsprechen 36.000 Units.
Im Bereich der Elevation müssen zwei Begebenheiten berücksichtigt werden. Der Parabolspiegel hat einen Offset von 30 Grad und ist in der Ruheposition 12 Grad nach hinten geneigt.
Vom Ergebnis aus der Berechnung für die Elevation müssen also 30 Grad Offset sowie die 12 Grad zur vertikalen Stellung abgezogen werden.

4.7 Praxisbetrieb

Im Automation Studio werden nun zwei Programme zyklisch alle 500ms ausgeführt. In Abb. 24 kann man diese Programme sehen und wenn nötig die Zykluszeit zwischen 10ms und 1.000ms variieren. Werden zum Beispiel öfters sich unterscheidende Werte eingegeben, muss die Zykluszeit herabgesetzt werden.

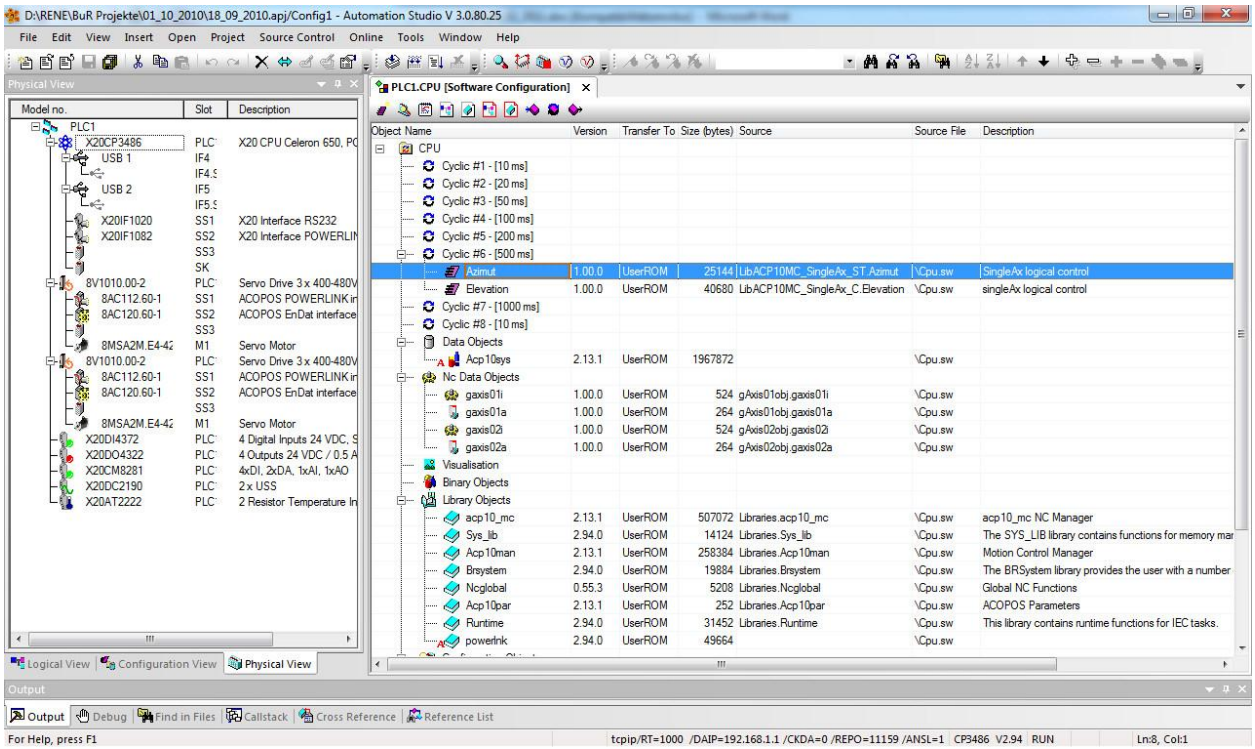

Abb. 24: Automation Studio Software Configuration

(Quelle: Automation Studio Version V 3.0.80.25)

Abb. 25 zeigt die Variablen sowie die Bewegungsbefehle welche eingegeben werden können.

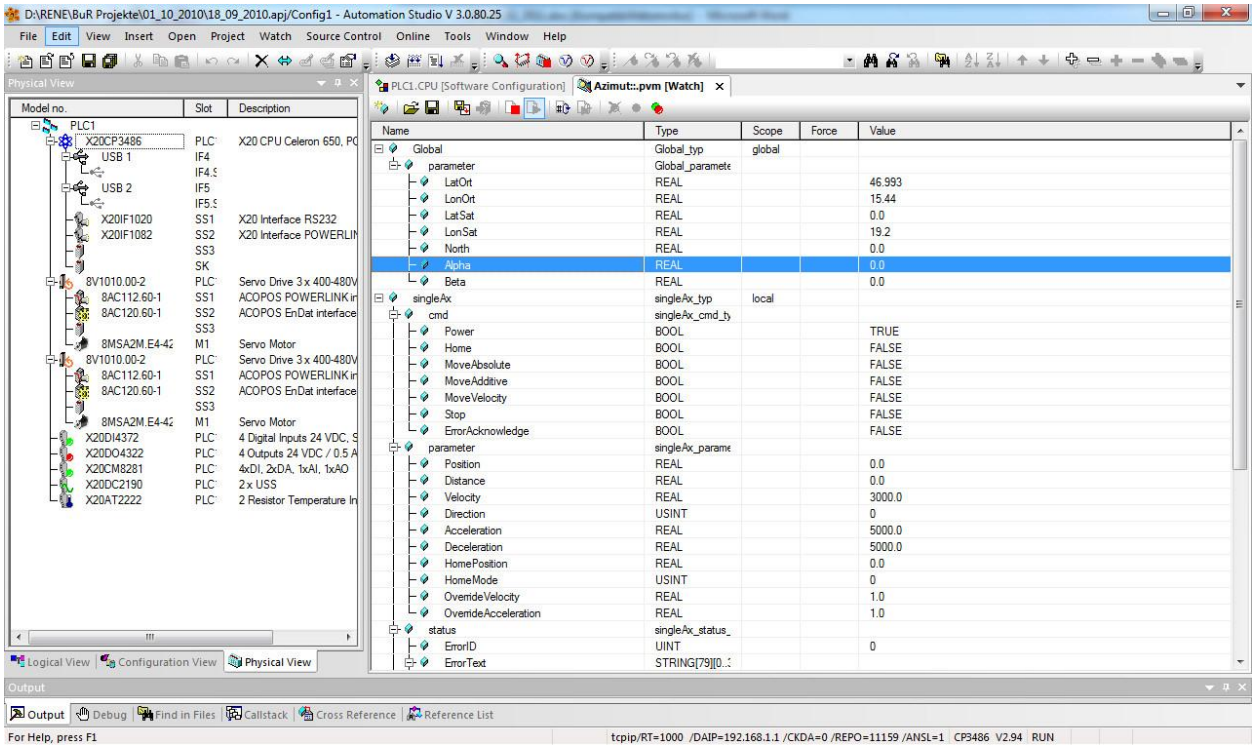

Abb. 25: Automation Studio Watch Azimut

(Quelle: Automation Studio Version V 3.0.80.25)

Die Variablen sind global angelegt und können so von beiden Programmen synchron verwendet werden. Eine Änderung wirkt sich also nach der Zykluszeit auf beide Programme aus und ändert somit die Position der beiden Motoren.

4.7.1 Ablauf der Eingabe und Ausführung

1.) Motor mittels „Power = 1“ einschalten
2.) Eingabe des Breiten- und Längengrads von Ort und Satellit
3.) Eingabe der Abweichung zum geografischen Norden
4.) Eingabe der Winkel Alpha und Beta
5.) Werte mittels „MoveAbsolute = 1“ an den Motor schicken

Das Resultat dieses Ablaufes ist das Drehen der Versuchsanordnung im Bereich des Azimuts bzw. der Elevation und somit die Ausrichtung auf den Satelliten.

4.7.2 Stoppen der Ausführung

Wird der Wert einer Variable falsch eingegeben oder der Motor führt eine Bewegung in eine kritische Richtung aus kann durch den Befehl „Stop = 1“ die aktuelle Bewegung sofort gestoppt werden.
Die Wiederaufnahme der Bewegung ist mittels „MoveAbsolute = 1“ möglich.

4.7.3 Beenden der Ausführung

Wenn die Versuchsanordnung nicht mehr verwendet oder ein Standortwechsel durchgeführt wird, darf der Motor nicht einfach stromlos geschaltet werden. Zuerst muss eine Ruheposition angefahren werden, welche durch den Befehl „Ruhe = 1“ eingeleitet wird. Diese setzt den Azimut und die Elevation auf eine vorher definierte Position. Ist diese Position erreicht, kann POWER auf 0 gestellt werden.
Wird dies nicht beachtet, fällt die Antenne unkontrolliert aus der aktuellen Position in eine mechanische Endposition und kann dadurch beschädigt werden.

5 Mechanik

Durch zwei unabhängig voneinander ansteuerbare Achsen kann jede zur Horizontalen abweichende Lage ausgeglichen werden. Des Weiteren kann jeder Punkt im Raum angesprochen werden.
Diese beiden Achsen sind im Falle dieses Versuchsaufbaus welcher in Abb. 26 dargestellt ist, der Drehteller (3) und die Welle (5), an welcher der Parabolspiegel befestigt wird.

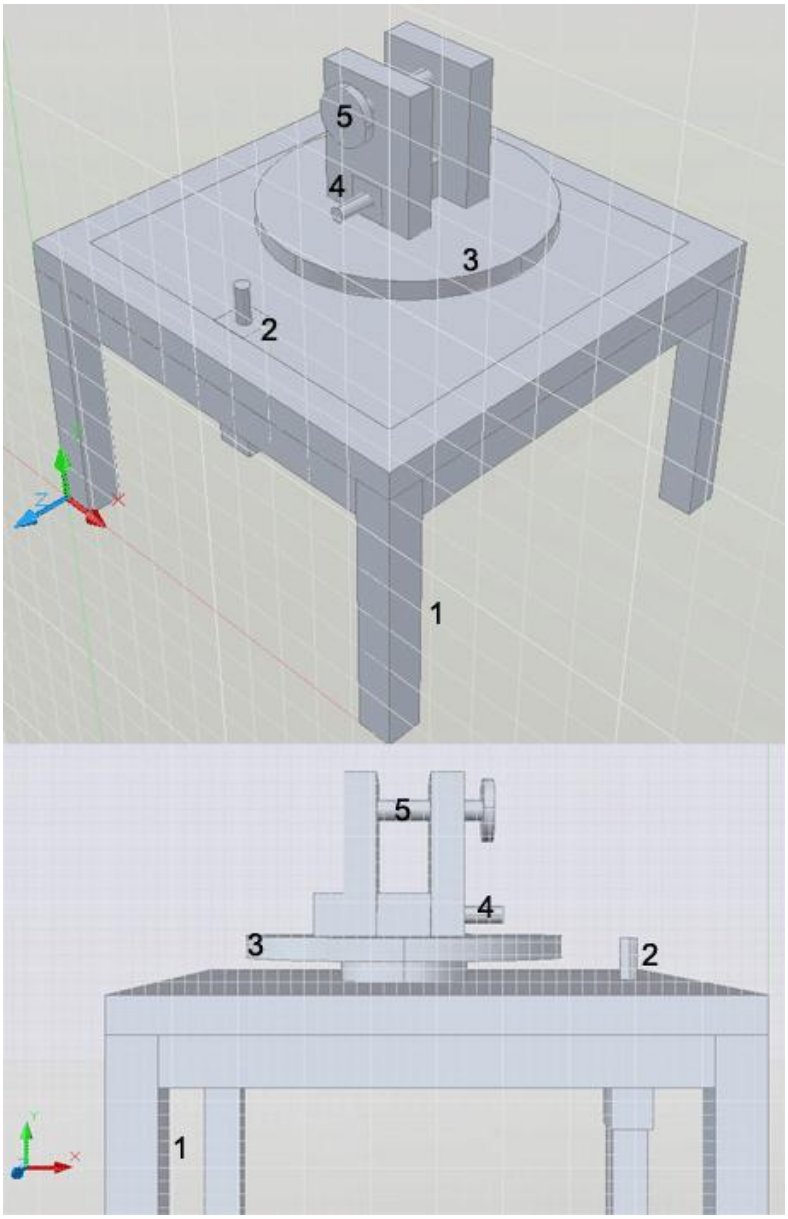

Abb. 26: Autocad-Modell der Versuchsanordnung
(gezeichnet von Unterluggauer)

1 … Tisch
2 … Motorwelle (Motor 1) mit Zahnscheibe 1
3 … Drehteller (Achse 1)
4 … Motorwelle (Motor 2) mit Zahnscheibe 2
5 … Welle für Parabolspiegel (Achse 2) mit Zahnscheibe 3

5.1 Wahl der Zahnscheiben

Um das Verhältnis der Zahnscheibe am Motor und der an der Achse möglichst groß zu gestalten, wird auf der Motorseite eine kleinstmögliche und an der Achse eine größtmögliche Zahnscheibe verwendet. Dadurch werden Änderungen an der Motorseite um ein Vielfaches geringer auf die Achse übertragen.
Im Azimutwinkel-Bereich steht dem Drehteller mit einem Durchmesser von 240 mm eine Zahnscheibe am Motor mit einem Durchmesser von 23 mm gegenüber. Durch diese große Übersetzung ist eine Änderung des Azimutwinkels in sehr kleinen Schritten möglich.
Zur Ansteuerung der Elevation wird aus Platzgründen eine Zahnscheibe mit einem Durchmesser von 51,94 mm gewählt (Zahnscheibe 3). Die dazugehörende Scheibe am Motor hat einen Durchmesser von 22,72 mm (Zahnscheibe 2).

5.1.1 Verhältnis im Bereich des Azimutwinkels

Folgende Werte werden als Grundlage der Berechnung herangezogen.
Die Zahnscheibe 1 (Zahnscheibe 5x10x15mm; RS-Compontens Bestell-Nr.: 744-974) hat ein Rastermaß von 5 mm,15 Zähne sowie einen gemessenen Durchmesser von 23 mm.

$$d_{zs1} = 23{,}00mm$$
$$r_{zs1} = \frac{d_{zs1}}{2} = 11{,}50mm$$
$$u_{zs1} = 2 \cdot \pi \cdot r_{zs1} = 72{,}26mm$$

Der Drehteller hat einen gemessen Durchmesser von 240 mm.

$$d_{drehteller} = 240{,}00mm$$
$$r_{drehteller} = \frac{d_{drehteller}}{2} = 120{,}00mm$$
$$u_{drehteller} = 2 \cdot \pi \cdot r_{drehteller} = 753{,}98mm$$

$$\frac{u_{drehteller}}{u_{zs1}} = \frac{753{,}98}{72{,}26} = 10{,}43$$

Daraus ergibt sich ein Verhältnis aus Zahnscheibe 1 zu Drehteller von 1:10,43.
Eine Umdrehung am Motor entspricht zirka 0,096 Umdrehungen (34,52 Grad) des Drehtellers.

5.1.2 Übersetzungsverhältnis im Bereich der Elevation

Für die Berechnung des Übersetzungsverhältnisses im Bereich der Elevation sind die Durchmesser der Zahnscheibe 2 und 3 notwendig.

Die Zahnscheibe 2 am Motor (Zahnscheibe XL 037 32 Zähne; RS-Compontens Bestell-Nr.: 182-638) hat einen gemessenen Durchmesser von 22 mm.

$$d_{zs2} = 22mm$$
$$r_{zs2} = \frac{d_{zs2}}{2} = 11{,}00mm$$
$$u_{zs2} = 2 \cdot \pi \cdot r_{zs2} = 69{,}12mm$$

Die Zahnscheibe 3 (Zahnscheibe XL 037 14 Zähne; RS-Compontens Bestell-Nr.: 182-672) hat einen gemessenen Durchmesser von 51,5 mm.

$$d_{zs3} = 51{,}5mm$$
$$r_{zs3} = \frac{d_{zs3}}{2} = 25{,}75mm$$
$$u_{zs3} = 2 \cdot \pi \cdot r_{zs3} = 161{,}79mm$$
$$\frac{u_{zs3}}{u_{zs2}} = \frac{161{,}79}{69{,}12} = 2{,}34$$

Daraus ergibt sich ein Verhältnis aus Zahnscheibe 2 zu Zahnscheibe 3 von 1:2,34.
Eine Umdrehung am Motor entspricht also zirka 0,43 Umdrehungen (153,85 Grad) der Achse zur Änderung der Elevation.

5.2 Kraftübertragung

Zur Kraftübertragung zwischen Motoren und Zahnscheiben werden Antriebsriemen verwendet. Durch die Zahnung der Zahnscheiben und der Antriebsriemen wird eine präzise Übertragung der Kraft gewährleistet und ein Durchrutschen vermieden.

5.2.1 Berechnung der Antriebsriemenlänge

Um zu vermeiden, dass zu viel bzw. unpassende Antriebsriemen bestellt werden wird eine Berechnung der Antriebsriemenlänge durchgeführt. Mit Hilfe der Durchmesser der beiden Zahnscheiben sowie deren Abstand kann die benötigte Länge berechnet werden.

Abb. 27: Antriebsriemen

(Quelle: http://at.rs-online.com/web/search/searchBrowseAction.html?method=searchProducts&searchTerm=474-9484&x=51&y=4#header)

Abb. 28: Zahnscheibe

(Quelle: http://img-europe.electrocomponents.com/largeimages/R744491-05.jpg)

5.2.2 Länge des Antriebsriemens zur Ansteuerung des Azimutwinkels

Für die Berechnung der Antriebsriemenlänge im Bereich des Azimutwinkels sind die Zahnscheibe 1 sowie der Drehteller ausschlaggebend.

Folgende Werte werden für die Berechnung herangezogen:

$r_{zs1} = 11{,}50mm$

$r_{drehteller} = 120{,}00mm$

Der zentrale Abstand zwischen Zahnscheibe 1 und Drehteller beträgt:

$d = 261{,}00mm$

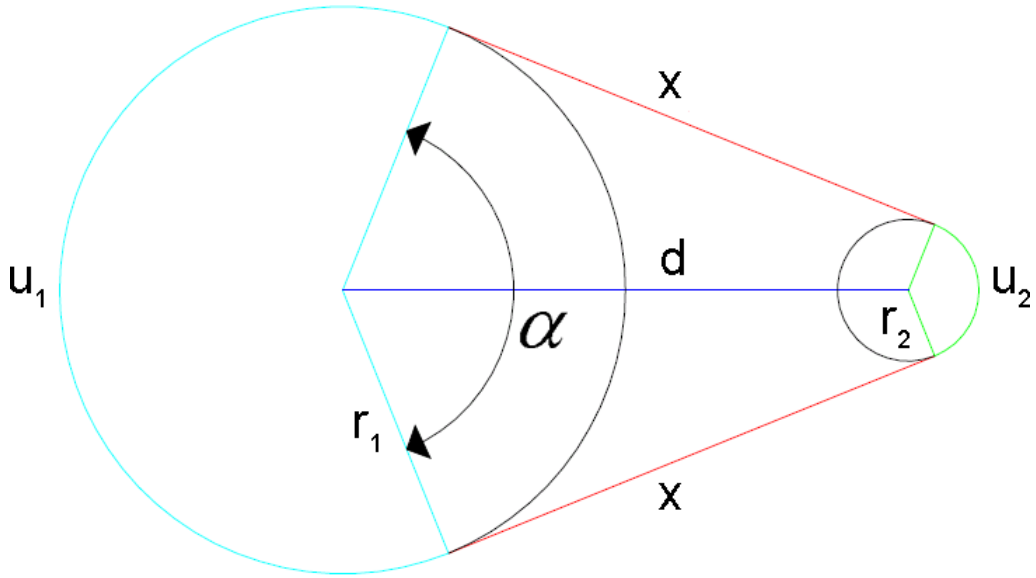

Abb. 29: Skizze Antriebsriemen Azimut

(gezeichnet von Unterluggauer)

In Abb. 29 sieht man die schematische Darstellung der wichtigen Komponenten zur Berechnung.

Unter Berücksichtigung der Werte für r_{zs1}, $r_{drehteller}$ und d wird die Berechnung wie folgt ausgeführt:

$r_1 = r_{drehteller}$
$r_2 = r_{zrs}$
$x = \sqrt{d^2 - (r_1 - r_2)^2} = 237{,}38mm$
$\alpha = 2 \cdot \cos^{-1}((r_1 - r_2)/d) = 130{,}87°$
$u_1 = [(2 \cdot r_1 \cdot \pi)/360] \cdot (360 - \alpha) = 479{,}89mm$
$u_2 = [(2 \cdot r_2 \cdot \pi)/360] \cdot \alpha = 26{,}28mm$
$l = 2 \cdot x + u_1 + u_2 = 980{,}93mm$

Es ergibt sich eine Länge für den Zahnriemen im Azimut-Bereich von zirka 990 mm. Aus diesem Grund wird das Produkt Zahnriemen 10x990 mm T5 (RS-Compontens Bestell-Nr.: 474-5836) gewählt und bestellt.

5.2.3 Länge des Antriebsriemens zur Ansteuerung der Elevation

Für die Berechnung im Bereich der Elevation sind die Durchmesser der Zahnscheiben 2 und 3 notwendig.
Folgende Werte werden dafür verwendet:

$r_{zs2} = 11{,}00mm$
$r_{rz3} = 25{,}75mm$

Der zentrale Abstand zwischen Zahnscheibe 2 und Zahnscheibe 3 beträgt:

$d = 105{,}50mm$

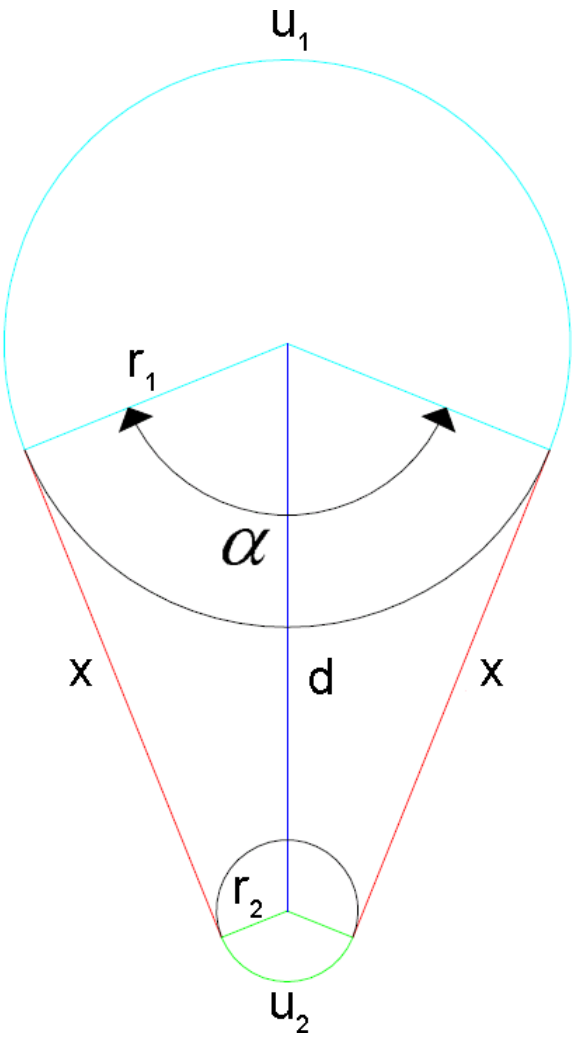

Abb. 30: Skizze Antriebsriemen Elevation

(gezeichnet von Unterluggauer)

In Abb. 30 sieht man die schematische Darstellung der wichtigen Komponenten zur Berechnung.

Unter Berücksichtigung der Werte für r_{zs2}, r_{zs3} und d wird die Berechnung wie folgt ausgeführt:

$$r_1 = r_{zs3}$$
$$r_2 = r_{zs2}$$
$$x = \sqrt{d^2 - (r_1 - r_2)^2} = 104{,}48mm$$
$$\alpha = 2 \cdot \cos^{-1}((r_1 - r_2)/d) = 164{,}08°$$
$$u_1 = [(2 \cdot r_1 \cdot \pi)/360] \cdot (360 - \alpha) = 88{,}80mm$$
$$u_2 = [(2 \cdot r_2 \cdot \pi)/360] \cdot \alpha = 32{,}53mm$$
$$l = 2 \cdot x + u_1 + u_2 = 330{,}29mm$$

Die Berechnung ergibt eine Länge für den Antriebsriemen von 330,29 mm oder 13 Zoll (1 Zoll = 25,4 mm).
Aus diesem Grund wird das Produkt Antriebsriemen 130 XL 037
(RS-Compontens Bestell-Nr.: 474-9484) gewählt.

5.3 Unterbau aus ITEM-Profilen

Abb. 31: ITEM-Profile

(Quelle: http://www.item24.com/de/produkte/mb_systembaukasten/grundelemente.html)

Durch die Verwendung eines Unterbaus kann die Versuchsanordnung auf eine bequeme Arbeitshöhe gebracht werden. Auch ist ein Einsatz im Gelände möglich. Aus Gründen der Stabilität und der Einfachheit der Verarbeitung werden 30 x 30 mm ITEM-Profile (siehe Abb. 31) gewählt, welche von der hauseigenen Werkstätte der TU Graz montiert wurden.

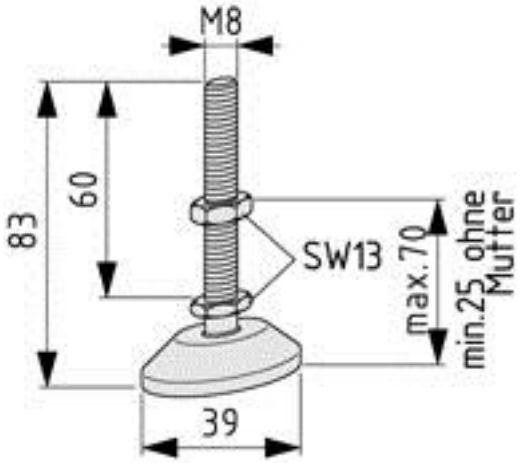

Abb. 32: Stellfuß

(Quelle: http://catalog.item.info/Onlinekatalog/web/DE/artikel/Teller-StellfueSSe_1000020894/26569)

Der Unterbau hat eine Höhe von 69 cm und kann mittels 4 Stellfüßen (siehe Abb. 32) geneigt werden. Die Höhe jedes Tischbeins kann im Bereich von 25 bis 70 mm stufenlos verstellt werden. Somit kann eine maximale Neigung von 5,7 Grad ohne Verwendung von zusätzlichen Hilfsmitteln simuliert werden.

Folgendes Material wird zur Realisierung verwendet:
Z27626 Profil 6 30 x 30 L
0.0.419.06 Zuschnitt maximal 6.000 mm
Notwendig: 5 m
Z27631 Standard-Verbindungssatz
6 0.0.419.14 1 Satz
Notwendig: 8 Stück
Z15672 Stellfuß 40 M8x80
0.0.265.69 1 Stück
Notwendig: 4 Stück

5.4 Sonstige Bauteile

Für die Realisierung der Versuchsanordnung werden noch weitere Bauteile benötigt. Diese werden in Folge beschrieben.

5.4.1.1 Holzkonstruktion

Die Holzkonstruktion wurde aus MDF-Platten hergestellt und besteht im Wesentlichen aus einer Tischplatte, dem Drehteller sowie zwei vertikalen Stücken, welche als Halterung für die Stahlwelle dienen.

5.4.1.2 Flanschlager

Zur Befestigung der 20 mm Stahlwelle werden zwei Flanschlager verwendet. Diese Flanschlager werden benötigt, damit der Motor die Welle und somit die Elevation des Offsetspiegels ändern kann. Auf eine qualitativ hochwertige Ausführung und damit verbundene Leichtgängigkeit des Lagers wurde bei der Auswahl der Komponente Wert gelegt.

Die Wahl fiel auf das in Abb. 33 dargestellte Produkt „Flanschlager 2-Loch FYTB20TF“ (RS Bestell-Nr. 339-8552).

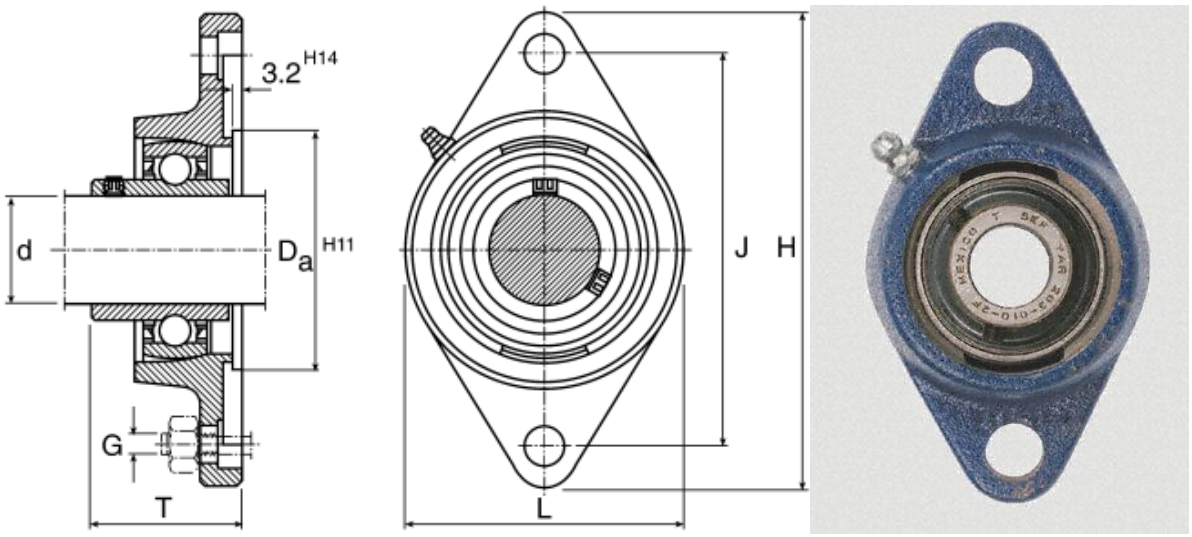

Abb. 33: Flanschlager

(Quelle: http://at.rs-online.com/web/search/searchBrowseAction.html?method=getProduct&R=3398552#header)

5.4.1.3 Offsetspiegel

Abb. 34: TechniSat DigiDish 33

(Quelle: http://www.technisat.de/index4e17.html?nav=Au%C3%9Fenanlagen,de,54-75)

Auf der 20 mm Stahlwelle wird der Offsetspiegel mittels einer Schraube montiert.

Kompakte Abmessungen sowie Leistungsstärke waren für die Wahl des Offsetspiegels von größter Bedeutung. Aus diesem Grund fiel die Wahl auf die „DigiDish 33“ von TechniSat (siehe Abb. 34).

Empfangstests von Seiten TechniSat in Deutschland, Österreich und der Schweiz zeigen, dass der kleine Offsetspiegel sehr leistungsstark und für den Empfang der digitalen Fernseh- und Radioprogramme von Astra 19,2° Ost bestens geeignet ist. Aufgrund der leistungsstärkeren Satellitensignale, digitaler Übertragung und erstklassiger Empfangsdaten des Offsetspiegels sind beim Empfang von ASTRA 19,2° Ost keine Qualitätseinbußen zu erwarten (vgl. TechniSat Digital GmbH 2010, S. 43).

6 Randbedingungen

Im Umgang mit der Versuchsanordnung muss auf Umwelteinflüsse geachtet werden. Diese werden folgend beschrieben.

6.1 Umwelteinfluss Wind

Da der Parabolspiegel eine Fläche von zirka 10 dm² einnimmt, bietet er eine große Angriffsfläche. Bei zu starkem Wind ist aus Sicherheitsgründen die Versuchsanordnung abzubauen da sonst Material und Motoren beschädigt werden können.

6.2 Umwelteinfluss Sonne

Sonneneinstrahlung kann zu einer Erwärmung der Motoren führen. Aus Sicherheitsgründen und zum Schutz der Motoren schaltet die *B&R* Software bei zu starker Erwärmung automatisch ab.

7 Ergebnisse und Schlussfolgerungen

Eine automatisch nachgeführte Antenne ist für den mobilen Einsatz sinnvoll und kann mit den vorhandenen Mitteln realisiert werden.

7.1 Probleme

Im Zuge der Projektarbeiten sind Probleme aufgetreten welche auf der einen Seite die Motoren und auf der anderen Seite die Mobilität betreffen.

7.1.1 Fehlerstrom-Schutzeinrichtung

Aufgrund der installationstechnischen Gegebenheiten im Zentrallabor der TU Graz ist es nicht möglich, beide Motoren parallel zu betreiben, da die Fehlerstrom-Schutzeinrichtung sofort nach Anschließen des zweiten Motors auslöst. Der Betrieb eines einzelnen Motors ist möglich.

Im ACOPOS Anwenderhandbuch ist der Grund für dieses Verhalten beschrieben.
„Bei den ACOPOS Servoverstärkern dürfen Fehlerstrom-Schutzeinrichtungen (RCD - residualcurrent-operated protective device) verwendet werden. Es ist jedoch folgendes zu beachten: Die ACOPOS Servoverstärker haben einen Netzgleichrichter. Bei einem Körperschluss kann ein glatter Fehlergleichstrom entstehen, der die Auslösung eines wechselstromsensitiven bzw. pulsstromsensitiven RCD (Typ A, bzw. AC) verhindert und somit die Schutzfunktion für alle daran angeschlossenen Verbraucher aufhebt.
*Bei ACOPOS Servoverstärkern sind Fehlerstrom-Schutzeinrichtungen mit einem Bemessungsfehlerstrom von ≥ 100 mA verwendbar“ (*B&R ACOPOS Anwenderhandbuch V 1.42, S. 187).

„Beispielsweise kann der allstromsensitive, 4-polige Fehlerstrom-Schutzschalter F 804 der Fa. ABB (Fehlerstrom: 300 mA; Nennstrom: 63 A) eingesetzt werden. Über diesen Fehlerstrom-Schutzschalter können ca. 5 ACOPOS 1022 (bzw. 1045, 1090) parallel angeschlossen werden“ (B&R ACOPOS Anwenderhandbuch V 1.42, S. 188).

Der Fehlerstrom-Schutzschalter *ABB Stotz S&J* F804B-63/0,03AP-R (Abb. 37) ist ein 4-poliger Schutzschalter mit einem Bemessungsstrom von 63A, einem Bemessungsfehlerstrom von 0,03A und der Auslöseeigenschaft vom Typ B.
Er erfüllt die Anforderungen aus dem B&R ACOPOS Anwenderhandbuch und kostet ca. 690,00 Euro.

Abb. 37: Fehlerstrom-Schutzschalter ABB Stotz S&J F804B-63/0,03AP-R

(Quelle: http://www.eibmarkt.com/cgi-bin/eibmarkt.storefront/4d4e9c9f015b32b4273d4debae380693/Product/View/NS0632784)

Ein vergleichbarer Fehlerstrom-Schutzschalter der Firma *SIEMENS* ist das Modell mit der Herstellernummer 5SM36464 (Abb. 38). Es handelt sich dabei um einen FI-Schutzschalter mit einer Auslöseempfindlichen von 300mA, einem Nennstrom von 63A und der Auslöseeigenschaft Typ B.
Der Preis für diesen 4-poligen Schutzschalter beläuft sich auf 592,00 Euro.

Abb. 38: Fehlerstrom-Schutzschalter SIEMENS 5SM36464

(Quelle: http://at.rs-online.com/web/search/searchBrowseAction.html?method=getProduct&R=6219985)

7.1.2 Mobilität

Durch das Problem mit dem Fehlerstrom-Schutzschalter sowie der Notwendigkeit einer 400V – Spannungsversorgung ist die Versuchsanordnung an den Standort des *B&R*-Equipments gebunden.

Ein abgesetzter Betrieb ist nur mit passender Kabellänge möglich. Der mobile Einsatz wäre nur mit einem Stromaggregat möglich.

7.2 Verbesserungen

Verbesserungspotential gibt es im Bereich der automatischen Erfassung von Umgebungsvariablen. Die Auswahl von alternativen Antriebskomponenten würde die Möglichkeit des mobilen Einsatzes verbessern.

7.2.1 Automatisierung

Um die Kosten der Versuchsanordnung in Grenzen zu halten, wird in dieser Version der Realisierung auf aufwendige und teure Sensoren verzichtet. Neigungswinkel der x- und y-Achsen und die Nordrichtung müssen vom Benutzer abgelesen und manuell in das Programm eingegeben werden. Um diese Angaben zu automatisieren wären ein digitaler Neigungssensor, ein digitaler Kompass sowie passende Schnittstellen notwendig.

7.2.2 Zweiachsiger Neigungssensor

Bei den Recherchen wurde einen digitalen Neigungssensor der Firma B+L Industrial Measurements GmbH gefunden.

*Der ETT 2000-45-90ext ist ein programmierbarer zweiachsiger Analog/Digital Neigungsaufnehmer. Der Sensor liefert parallel analoge, PWM und RS-232 Neigungsdaten von beiden Messachsen. Der Aufnehmer beinhaltet eine Linearkorrektur für den gesamten Messbereich beider Neigungsachsen und einen programmierbaren Schaltpunkt für jede Achse (*B+L Industrial Measurements GmbH).

Abb. 35: Digitaler Neigungssensor ETT 2000-45-90-ext

(Quelle: URL:http://www.variohm.de/de/files/ETT-deu-rev5.pdf)

Dieser in Abb. 35 gezeigte Neigungssensor verfügt über einen maximalen Messbereich von -90° bis +90°, eine integrierte Temperaturkompensation, ist vibrations- und schockresistent und kostet zirka 500 Euro. Die Messdaten werden via RS232-Schnittstelle dem verarbeitenden Rechner zugeführt.

7.2.3 Digitaler Kompass

Die Firma B+L Industrial Measurements GmbH ist auch der Hersteller des digitalen Kompass EZ-Compass-3A.

Abb. 36: Digitaler Kompass

(Quelle: URL:http://www.variohm.de/de/neigung.html)

Dieser in Abb. 36 dargestellte dreiachsige Kompass kann sowohl den Azimutwinkel von 0° bis 360° als auch Neigung von -90° bis +90° ermitteln. Die Auflösung beträgt 12 Bit und die Genauigkeit liegt bei +/- 0,5°.
Das Ausgangssignal kann über RS232, RS422 oder RS485 dem verarbeitenden Rechner zugeführt werden. Der Preis liegt bei zirka 1000 Euro.

7.2.4 Alternative Motoren

Auf der Suche nach alternativen Antriebskomponenten bin ich auf die Firma „CNC Profi Ltd“ aus Deutschland gestoßen. Dieses Unternehmen bietet Schrittmotoren und die dazu passende Steuerung und Stromversorgung an (Abb. 39).

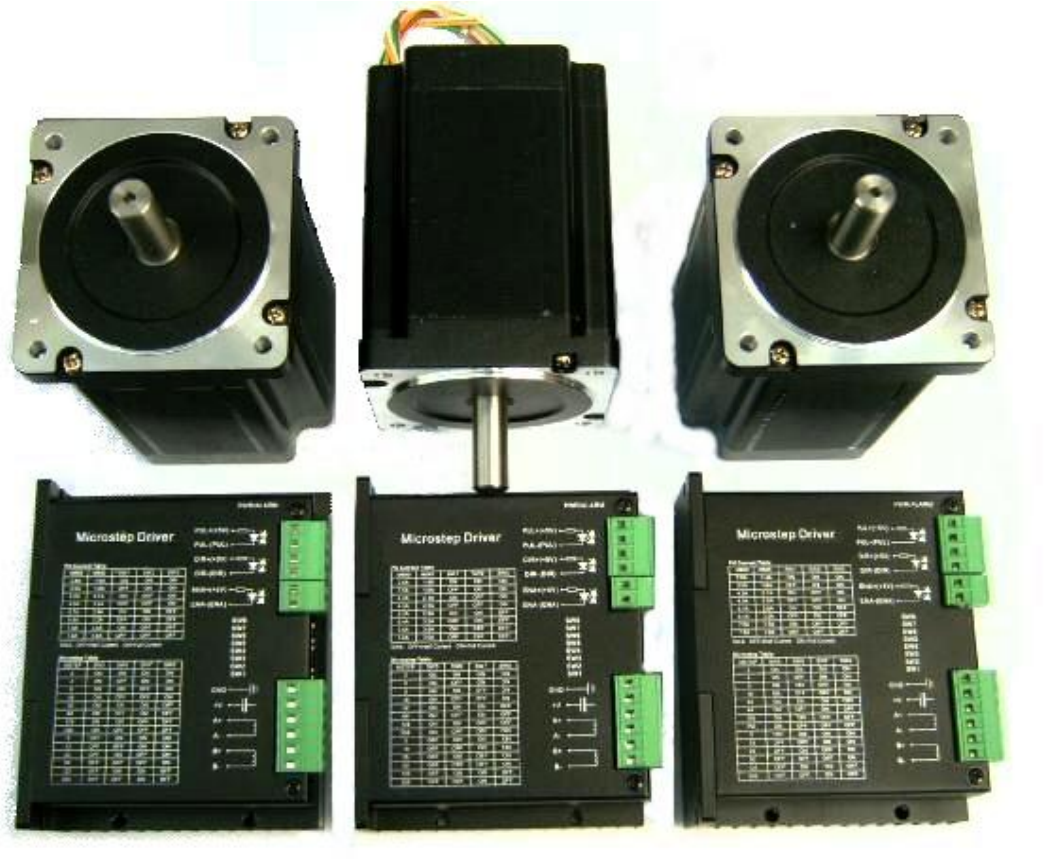

Abb. 39: Schrittmotor und Stromversorgung
(Quelle: URL: http://www.cncprofi.eu/index.php?action=Kategorie&kat=16)

Die Modellpalette reicht vom kleinen Schrittmotor mit einem Moment von 0,5 Nm bis hin zu großen Modellen mit einem Moment von 12,8 Nm.

Die Transformatoren stellen den Schrittmotoren eine Gleichspannung von 49 V zur Verfügung. Sie selbst benötigen eine Betriebsspannung von 230V. Dadurch wäre auch ein mobiler Einsatz leichter möglich. Die benötigte Spannung kann durch ein 230V Notstromaggregat erzeugt werden.

Vorschlag für mögliche Ausstattung:

- zwei Schrittmotoren mit einem Moment von 2 Nm
- eine Motorensteuerung
- ein Transformator für die Schrittmotoren
- ein Transformator für die Steuerung
- CNC-Kupfer-Kabel 4x1 mm²

Die Kosten würden sich bei dieser Variante auf zirka 400 Euro belaufen.

Literaturverzeichnis

Hans-Jochen Bartsch, 1999, Taschenbuch Mathematischer Formeln, Carl Hanser Verlag München Wien

Andrew S. Tanenbaum, 1997, Computernetzwerke, Prentice Hall Verlag GmbH

B&R, 2007a, TM410TRE.30-GER 0907
B&R, 2007b, TM246TRE.00-GER 0907
B&R, 2007c, TM450TRE.00-GER 0907

Internetrecherchen
ALDEN SAT-NET® Mondo 90 Platinum
http://www.sat-welt.com/Zoom/mondo.html
(Abruf am 08.03.10)

TechniSat Digital GmbH (2010): TechniTipp (03/10)
URL:http://www.technisat-daun-fs.de/downloads/magazine/TTM/partikel/TT03-10_S42-46_Empfangsanlagen.pdf
(Abruf am 09.03.10)

Alden Loisirs et Techniques
URL:http://www.sat-welt.com/Site%20Folder/pages_pdf/Katalogue.pdf
(Abruf am 09.03.10)

Alden IPcopter
URL:http://alden.ipcopter.com/de/shop/shop.php?catID=1000
(Abruf am 09.03.10)

Kowoma
URL:http://www.kowoma.de/gps/geo/laengenbreitengrad.htm
(Abruf am 18.03.10)

B+L Industrial Measurements GmbH
URL:http://www.variohm.de/de/files/ETT-deu-rev5.pdf
(Abruf am 21.04.10)

Bernecker und Rainer (B&R): ACOPOS Anwenderhandbuch V 1.42
URL:http://www.br-automation.com/cps/rde/xchg/br-productcatalogue/hs.xsl/services_164565_DEU_HTML.htm
(Abruf am 24.08.2010)

Alpenverein Freistadt
URL:http://www.alpenverein-freistadt.at/peaks.htm
(Abruf am 30.11.2010)

Siemens
URL:http://www.automation.siemens.com/mcms/industrial-communication/de/ie/Seiten/industrial-ethernet.aspx
(Abruf am 22.02.2011)

Software

Automation Studio B&R Help Explorer, 2010, Bernecker + Rainer

Abkürzungsverzeichnis

B&R	Bernecker und Rainer
kbit/s	Kilobit pro Sekunde
kbps	Kilobit pro Sekunde
Kbps	Kilobit pro Sekunde
Mbyte/s	Megabyte pro Sekunde
Nm	Newtonmeter
TV	Television
Univ.-Prof.	Universitätsprofessor
Dr.	Doktor
Dipl.-Ing.	Diplomingenieur
DI	Diplomingenieur
GPS	Global Positioning System
bzw.	beziehungsweise
engl.	englisch
Abb.	Abbildung
%	Prozent
CPU	Central processing unit
TCPIP	Transmission Control Protocol/Internet Protocol
PC	Personal Computer
IP	Internet Protocol
ST	Strukturierter Text
MDF	Mitteldichte Faserplatte
GmbH	Gesellschaft mit beschränkter Haftung
V	Volt
mm	Millimeter
cm	Zentimeter
km	Kilometer
FDM	Frequency Division Multiplexing (Frequenzmultiplexverfahren)
TDM	Time Division Multiplexing (Zeitmultiplexverfahren)
CSMA	Carrier Sense Multiple Access
PCM	Pulscodemodulation

LAN	Local Area Network
WAN	Wide Area Network
ACTS	Advanced Communication Technology Satellite

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A Programmcode Azimutwinkel

```
(*********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 *********************************************************************
 * PROGRAM: singleAx
 * File: singleAxCyclic.st
 * Author: Bernecker + Rainer
 * Created: May 07, 2009
 *********************************************************************
 * Implementation of Program singleAx
 *********************************************************************)
PROGRAM _CYCLIC

(*************************************************************
      Control Sequence
*************************************************************)

(**************** CHECK FOR GENERAL AXIS ERROR ******************)

   IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND (MC_ReadAxisError_0.Valid = TRUE)) THEN

      AxisStep := STATE_ERROR_AXIS;

(***************** CHECK IF POWER SHOULD BE OFF ******************)

   ELSIF (singleAx.cmd.Power = FALSE) THEN

      IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE)) THEN

         AxisStep := STATE_ERROR_RESET;
```

```
        ELSE
            AxisStep := STATE_WAIT;
        END_IF
    END_IF
CASE AxisStep OF

(******************** WAIT ************************)
    STATE_WAIT:  (* STATE: Wait *)
        IF (singleAx.cmd.Power = TRUE) THEN
            AxisStep := STATE_POWER_ON;
        ELSE
            MC_Power_0.Enable := FALSE;
        END_IF
        (* reset all FB execute inputs we use *)
        MC_Home_0.Execute := FALSE;
        MC_Stop_0.Execute := FALSE;
        MC_MoveAbsolute_0.Execute := FALSE;
```

```
        MC_MoveAdditive_0.Execute := FALSE;
        MC_MoveVelocity_0.Execute := FALSE;
        MC_ReadAxisError_0.Acknowledge := FALSE;
        MC_Reset_0.Execute := FALSE;
        (* reset user commands *)
        singleAx.cmd.Stop := FALSE;
        singleAx.cmd.Home := FALSE;
        singleAx.cmd.MoveVelocity := FALSE;
        singleAx.cmd.MoveAbsolute := FALSE;
        singleAx.cmd.MoveAdditive := FALSE;
        singleAx.status.ErrorID := 0;


(******************** POWER ON **********************)
    STATE_POWER_ON:  (* STATE: Power on *)
        MC_Power_0.Enable := TRUE;
        IF (MC_Power_0.Status = TRUE) THEN
            AxisStep := STATE_HOME; (*automatic homing after Power on*)
```

```
                    AxisStep := STATE_READY;

    END_IF

    (* if a power error occured go to error state *)

    IF (MC_Power_0.Error = TRUE) THEN

        singleAx.status.ErrorID := MC_Power_0.ErrorID;

        AxisStep := STATE_ERROR;

    END_IF


(******************** READY **********************)

    STATE_READY:  (* STATE: Waiting for commands *)

    IF (singleAx.cmd.Home = TRUE)THEN

        singleAx.cmd.Home := FALSE;

        AxisStep := STATE_HOME;



    ELSIF (singleAx.cmd.Stop = TRUE) THEN

        singleAx.cmd.Stop := FALSE;

        AxisStep := STATE_STOP;
```

```
        ELSIF (singleAx.cmd.MoveAbsolute = TRUE) THEN

            singleAx.cmd.MoveAbsolute := FALSE;

            AxisStep := STATE_MOVE_ABSOLUTE;



        ELSIF (singleAx.cmd.MoveAdditive = TRUE) THEN

            singleAx.cmd.MoveAdditive := FALSE;

            AxisStep := STATE_MOVE_ADDITIVE;



        ELSIF (singleAx.cmd.MoveVelocity = TRUE) THEN

            singleAx.cmd.MoveVelocity := FALSE;
            AxisStep := STATE_MOVE_VELOCITY;

        END_IF



(******************** HOME **********************)

    STATE_HOME:  (* STATE: start homing process *)

        MC_Home_0.Position := singleAx.parameter.HomePosition;

        MC_Home_0.HomingMode := singleAx.parameter.HomeMode;
```

```
MC_Home_0.Execute := TRUE;

IF (singleAx.cmd.Stop = TRUE) THEN

    singleAx.cmd.Stop := FALSE;

    MC_Home_0.Execute := FALSE;

    AxisStep := STATE_STOP;

ELSIF (MC_Home_0.Done = TRUE) THEN

    MC_Home_0.Execute := FALSE;

    AxisStep := STATE_READY;

END_IF

(* if a homing error occured go to error state *)

IF (MC_Home_0.Error = TRUE) THEN
    MC_Home_0.Execute := FALSE;

    singleAx.status.ErrorID := MC_Home_0.ErrorID;

    AxisStep := STATE_ERROR;

END_IF


(********************** STOP MOVEMENT ************************)
```

```
    STATE_STOP:  (* STATE: Stop movement *)
        MC_Stop_0.Deceleration := singleAx.parameter.Deceleration;
        MC_Stop_0.Execute := TRUE;
        (* if axis is stopped go to ready state *)
        IF (MC_Stop_0.Done = TRUE) THEN
            MC_Stop_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_Stop_0.Error = TRUE) THEN
            singleAx.status.ErrorID := MC_Stop_0.ErrorID;
            MC_Stop_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF


(********************* START ABSOLUTE MOVEMENT **********************)
    STATE_MOVE_ABSOLUTE:  (* STATE: Start absolute movement *)
```

```
            azimut_b := Global.parameter.LatOrt * DegreeToRad;
            azimut_c  :=  (Global.parameter.LonSat  -  Global.parameter.LonOrt)  *
DegreeToRad;
            azimut_a                                                          :=
ACOS(COS(azimut_b)*COS(azimut_c)+SIN(azimut_b)*SIN(azimut_c)*COS(90*Degr
eeToRad));

            azimut          :=          ((180*DegreeToRad)+ACOS((COS(azimut_c)-
(COS(azimut_a)*COS(azimut_b)))/(SIN(azimut_a)*SIN(azimut_b))))/DegreeToRad;
        //Seitencosinussatz

            azimut := azimut + (Global.parameter.North * 100);

        IF (azimut > 18100) THEN

            azimut := azimut - 36000;
            MC_MoveAbsolute_0.Direction     := 1;

        ELSE
            MC_MoveAbsolute_0.Direction     := 0;
        END_IF;

            MC_MoveAbsolute_0.Position      := azimut;
    MC_MoveAbsolute_0.Velocity      := singleAx.parameter.Velocity;

    MC_MoveAbsolute_0.Acceleration  := singleAx.parameter.Acceleration;

    MC_MoveAbsolute_0.Deceleration  := singleAx.parameter.Deceleration;

    MC_MoveAbsolute_0.Execute := TRUE;

    (* check if commanded position is reached *)
```

```
        IF (singleAx.cmd.Stop = TRUE) THEN
            singleAx.cmd.Stop := FALSE;
            MC_MoveAbsolute_0.Execute := FALSE;
            AxisStep := STATE_STOP;
        ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
            MC_MoveAbsolute_0.Execute := FALSE;
            AxisStep := STATE_MOVE_ABSOLUTE;
        END_IF
        (* check if error occured *)
        IF (MC_MoveAbsolute_0.Error = TRUE) THEN
            singleAx.status.ErrorID := MC_MoveAbsolute_0.ErrorID;
            MC_MoveAbsolute_0.Execute := TRUE;
            AxisStep := STATE_ERROR;
        END_IF


(********************* START ADDITIVE MOVEMENT **********************)
    STATE_MOVE_ADDITIVE:  (* STATE: Start additive movement *)
```

```
MC_MoveAdditive_0.Distance      := singleAx.parameter.Distance;

MC_MoveAdditive_0.Velocity      := singleAx.parameter.Velocity;

MC_MoveAdditive_0.Acceleration  := singleAx.parameter.Acceleration;

MC_MoveAdditive_0.Deceleration  := singleAx.parameter.Deceleration;

MC_MoveAdditive_0.Execute := TRUE;

(* check if commanded distance is reached *)

IF (singleAx.cmd.Stop = TRUE) THEN

    singleAx.cmd.Stop := FALSE;

    MC_MoveAdditive_0.Execute := FALSE;

    AxisStep := STATE_STOP;

ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN

    MC_MoveAdditive_0.Execute := FALSE;

    AxisStep := STATE_READY;

END_IF

(* check if error occured *)

IF (MC_MoveAdditive_0.Error = TRUE) THEN

    singleAx.status.ErrorID := MC_MoveAdditive_0.ErrorID;
```

```
            MC_MoveAdditive_0.Execute := FALSE;

            AxisStep := STATE_ERROR;

        END_IF



(******************** START VELOCITY MOVEMENT **********************)
    STATE_MOVE_VELOCITY:  (* STATE: Start velocity movement *)

        MC_MoveVelocity_0.Velocity      := singleAx.parameter.Velocity;

        MC_MoveVelocity_0.Acceleration  := singleAx.parameter.Acceleration;

        MC_MoveVelocity_0.Deceleration  := singleAx.parameter.Deceleration;

        MC_MoveVelocity_0.Direction     := singleAx.parameter.Direction;

        MC_MoveVelocity_0.Execute := TRUE;

        (* check if commanded velocity is reached *)

        IF (singleAx.cmd.Stop = TRUE) THEN

            singleAx.cmd.Stop := FALSE;

            MC_MoveVelocity_0.Execute := FALSE;

            AxisStep := STATE_STOP;

        ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
```

```
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_MoveVelocity_0.Error = TRUE) THEN
            singleAx.status.ErrorID := MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF


(******************** FB-ERROR OCCURED ************************)
    STATE_ERROR:  (* STATE: Error *)
        (* check if FB indicates an axis error *)
        IF (singleAx.status.ErrorID = 29226) THEN
            AxisStep := STATE_ERROR_AXIS;
        ELSE
            IF (singleAx.cmd.ErrorAcknowledge = TRUE) THEN
```

```
            singleAx.cmd.ErrorAcknowledge := FALSE;

            singleAx.status.ErrorID := 0;

            (* reset axis if it is in axis state ErrorStop *)

            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid
= TRUE)) THEN

               AxisStep := STATE_ERROR_RESET;

            ELSE

               AxisStep := STATE_WAIT;

            END_IF

         END_IF

      END_IF



(******************** AXIS-ERROR OCCURED ************************)

   STATE_ERROR_AXIS:  (* STATE: Axis Error *)

      IF (MC_ReadAxisError_0.Valid = TRUE) THEN

         IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN

            singleAx.status.ErrorID := MC_ReadAxisError_0.AxisErrorID;
```

```
        END_IF

        MC_ReadAxisError_0.Acknowledge := FALSE;

        IF (singleAx.cmd.ErrorAcknowledge = TRUE) THEN

            singleAx.cmd.ErrorAcknowledge := FALSE;

            (* acknowledge axis error *)

            IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN

                MC_ReadAxisError_0.Acknowledge := TRUE;

            END_IF

        END_IF

        IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN

            (* reset axis if it is in axis state ErrorStop *)

            singleAx.status.ErrorID := 0;

            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid
= TRUE)) THEN

                AxisStep := STATE_ERROR_RESET;

            ELSE

                AxisStep := STATE_WAIT;
```

```
            END_IF

        END_IF

    END_IF



(******************** RESET DONE ************************)

    STATE_ERROR_RESET:  (* STATE: Wait for reset done *)

        MC_Reset_0.Execute := TRUE;

        (* reset MC_Power.Enable if this FB is in Error*)

        IF (MC_Power_0.Error = TRUE) THEN

            MC_Power_0.Enable := FALSE;

        END_IF

        IF(MC_Reset_0.Done = TRUE)THEN

            MC_Reset_0.Execute := FALSE;

            AxisStep := STATE_WAIT;

        END_IF

(******************** SEQUENCE END ************************)

END_CASE
```

```
(*************************************************************

        Function Block Calls

*************************************************************)



(************************** MC_POWER ****************************)

MC_Power_0.Axis := Axis1Obj;  (* pointer to axis *)

MC_Power_0();



(************************** MC_HOME ****************************)

MC_Home_0.Axis := Axis1Obj;

MC_Home_0();



(********************** MC_MOVEABSOLUTE ************************)

MC_MoveAbsolute_0.Axis := Axis1Obj;

MC_MoveAbsolute_0();
```

```
(********** MC_MOVEADDITIVE **********)

MC_MoveAdditive_0.Axis := Axis1Obj;

MC_MoveAdditive_0();



(********** MC_MOVEVELOCITY **********)

MC_MoveVelocity_0.Axis := Axis1Obj;

MC_MoveVelocity_0();



(********** MC_STOP **********)

MC_Stop_0.Axis := Axis1Obj;

MC_Stop_0();



(********** MC_RESET **********)

MC_Reset_0.Axis := Axis1Obj;

MC_Reset_0();



(********** MC_READAXISERROR **********)
```

```
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);

MC_ReadAxisError_0.Axis := Axis1Obj;

MC_ReadAxisError_0.DataAddress := ADR(singleAx.status.ErrorText);

MC_ReadAxisError_0.DataLength := SIZEOF(singleAx.status.ErrorText);

MC_ReadAxisError_0.DataObjectName := 'english';

MC_ReadAxisError_0();



(************************ MC_READSTATUS ************************)

MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);

MC_ReadStatus_0.Axis := Axis1Obj;

MC_ReadStatus_0();

singleAx.axisState.Disabled          := MC_ReadStatus_0.Disabled;

singleAx.axisState.StandStill        := MC_ReadStatus_0.StandStill;

singleAx.axisState.Stopping          := MC_ReadStatus_0.Stopping;

singleAx.axisState.Homing            := MC_ReadStatus_0.Homing;

singleAx.axisState.DiscreteMotion    := MC_ReadStatus_0.DiscreteMotion;

singleAx.axisState.ContinuousMotion  := MC_ReadStatus_0.ContinuousMotion;
```

```
singleAx.axisState.SynchronizedMotion   := MC_ReadStatus_0.SynchronizedMotion;

singleAx.axisState.ErrorStop            := MC_ReadStatus_0.Errorstop;



(******************** MC_SETOVERRIDE **********************)

MC_SetOverride_0.Enable      :=      (NOT(MC_ReadStatus_0.StandStill)      AND
NOT(MC_SetOverride_0.Error) AND MC_Power_0.Status);

MC_SetOverride_0.Axis := Axis1Obj;

MC_SetOverride_0.VelFactor := singleAx.parameter.OverrideVelocity;

MC_SetOverride_0.AccFactor := singleAx.parameter.OverrideAcceleration;

MC_SetOverride_0();



(******************** MC_READACTUALPOSITION **********************)

MC_ReadActualPosition_0.Enable  :=  (NOT(MC_ReadActualPosition_0.Error)  AND
MC_Power_0.Status);

MC_ReadActualPosition_0.Axis := Axis1Obj;

MC_ReadActualPosition_0();

IF(MC_ReadActualPosition_0.Valid = TRUE)THEN

    singleAx.status.ActPosition := MC_ReadActualPosition_0.Position;
```

```
END_IF



(******************** MC_READACTUALVELOCITY **********************)

MC_ReadActualVelocity_0.Enable := (NOT(MC_ReadActualVelocity_0.Error) AND
MC_Power_0.Status);

MC_ReadActualVelocity_0.Axis := Axis1Obj;

MC_ReadActualVelocity_0();

IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN

    singleAx.status.ActVelocity := MC_ReadActualVelocity_0.Velocity;

END_IF

END_PROGRAM
```

Anhang B Programmcode Elevationswinkel

```
/********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 ********************************************************************
 * PROGRAM: singleAx
 * File: singleAx.c
 * Author: Bernecker + Rainer
 * Created: May 07, 2009
 ********************************************************************
 * Implementation of Program singleAx
 ********************************************************************/

#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif
#include <bur/plc.h>
#include <bur/plctypes.h>
#include <string.h>
#include <acp10man.h>
#include <acp10_mc.h>
#include <math.h>

/* defines of the state-constants */
#define horizont                    90

#define STATE_WAIT          0
#define STATE_POWER_ON      1
#define STATE_HOME          2

#define STATE_READY         10

#define STATE_STOP          11
```

```
#define STATE_JOG_POSITIVE      12
#define STATE_JOG_NEGATIVE      13
#define STATE_MOVE_ABSOLUTE     14
#define STATE_MOVE_ADDITIVE     15
#define STATE_MOVE_VELOCITY     16

#define STATE_ERROR_AXIS        100
#define STATE_ERROR             101
#define STATE_ERROR_RESET       102

_INIT void basic_init(void)
{

/* get axis object */
Axis1Obj = (UDINT)&gAxis02;

AxisStep = STATE_WAIT;

singleAx.parameter.Velocity             = 50;   /*velocity for movement*/
singleAx.parameter.Acceleration         = 100;  /*acceleration for movement*/
singleAx.parameter.Deceleration         = 100;  /*deceleration for movement*/
singleAx.parameter.Direction                    = 0;      /*direction for movement*/
singleAx.parameter.JogVelocity          = 400;   /*velocity for jogging */
singleAx.parameter.OverrideVelocity        = 1.0;    /*multiplication factor for override velocity*/
singleAx.parameter.OverrideAcceleration   = 1.0;    /*multiplication factor for override acceleration*/

Global.parameter.LatOrt    = 46.993;
Global.parameter.LonOrt   = 15.440;
Global.parameter.LatSat    = 0;
Global.parameter.LonSat   = 19.2;
}
```

```
_CYCLIC void basic_cyclic(void)
{

/************************************************************
        Control Sequence
************************************************************/

/***************** CHECK FOR GENERAL AXIS ERROR ******************/
    if ((MC_ReadAxisError_0.AxisErrorID != 0) && (MC_ReadAxisError_0.Valid == 1))
    {
        AxisStep = STATE_ERROR_AXIS;
    }
/****************** CHECK IF POWER SHOULD BE OFF ******************/
    else if (singleAx.cmd.Power == 0)
    {
        if ((MC_ReadStatus_0.Errorstop == 1) && (MC_ReadStatus_0.Valid == 1))
        {
            AxisStep = STATE_ERROR_RESET;
        }
        else
        {
          AxisStep = STATE_WAIT;
        }
    }

switch(AxisStep)
{
/********************* WAIT ************************/
    case STATE_WAIT:  /* STATE: Wait */
        if (singleAx.cmd.Power == 1)
        {
            AxisStep = STATE_POWER_ON;
        }
```

```
        else
        {
                        MC_Power_0.Enable = 0;
        }
        /* reset all FB execute inputs we use */
        MC_Home_0.Execute = 0;
        MC_Stop_0.Execute = 0;
        MC_MoveAbsolute_0.Execute = 0;
        MC_MoveAdditive_0.Execute = 0;
        MC_MoveVelocity_0.Execute = 0;
        MC_ReadAxisError_0.Acknowledge = 0;
        MC_Reset_0.Execute = 0;
        /* reset user commands */
        singleAx.cmd.Ruhe = 0;
        singleAx.cmd.Stop = 0;
        singleAx.cmd.Home = 0;
        singleAx.cmd.MoveJogPos = 0;
        singleAx.cmd.MoveJogNeg = 0;
        singleAx.cmd.MoveVelocity = 0;
        singleAx.cmd.MoveAbsolute = 0;
        singleAx.cmd.MoveAdditive = 0;
        singleAx.status.ErrorID = 0;
    break;

/******************** POWER ON **********************/
    case STATE_POWER_ON:  /* STATE: Power on */
        MC_Power_0.Enable = 1;
        if (MC_Power_0.Status == 1)
        {
            AxisStep = STATE_HOME; /*automatic homing after Power on*/
                        AxisStep = STATE_READY;
        }
        /* if a power error occured go to error state */
        if (MC_Power_0.Error == 1)
```

```
        {
            singleAx.status.ErrorID = MC_Power_0.ErrorID;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** READY **********************/
    case STATE_READY:  /* STATE: Waiting for commands */
                if (singleAx.cmd.Ruhe == 1)
        {
            AxisStep = STATE_MOVE_ABSOLUTE;
        }
                else if (singleAx.cmd.Home == 1)
        {
            singleAx.cmd.Home = 0;
            AxisStep = STATE_HOME;
        }
        else if (singleAx.cmd.Stop == 1)
        {
            singleAx.cmd.Stop = 0;
            AxisStep = STATE_STOP;
        }
        else if (singleAx.cmd.MoveJogPos == 1)
        {
            AxisStep = STATE_JOG_POSITIVE;
        }
        else if (singleAx.cmd.MoveJogNeg == 1)
        {
            AxisStep = STATE_JOG_NEGATIVE;
        }
        else if (singleAx.cmd.MoveAbsolute == 1)
        {
            singleAx.cmd.MoveAbsolute = 0;
            AxisStep = STATE_MOVE_ABSOLUTE;
```

```
        }
        else if (singleAx.cmd.MoveAdditive == 1)
        {
            singleAx.cmd.MoveAdditive = 0;
            AxisStep = STATE_MOVE_ADDITIVE;
        }
        else if (singleAx.cmd.MoveVelocity == 1)
        {
            singleAx.cmd.MoveVelocity = 0;
            AxisStep = STATE_MOVE_VELOCITY;
        }
    break;

/******************** HOME **********************/
    case STATE_HOME:  /* STATE: start homing process */
        MC_Home_0.Position = singleAx.parameter.HomePosition;
        MC_Home_0.HomingMode = singleAx.parameter.HomeMode;
        MC_Home_0.Execute = 1;
        if (singleAx.cmd.Stop == 1)
        {
            singleAx.cmd.Stop = 0;
            MC_Home_0.Execute = 0;
            AxisStep = STATE_STOP;
        }
        else if (MC_Home_0.Done == 1)
        {
            MC_Home_0.Execute = 0;
            AxisStep = STATE_READY;
        }
        /* if a homing error occured go to error state */
    break;

/************************ STOP MOVEMENT ************************/
    case STATE_STOP: /* STATE: Stop movement */
```

```
        MC_Stop_0.Deceleration = singleAx.parameter.Deceleration;
        MC_Stop_0.Execute = 1;
        /* if axis is stopped go to ready state */
        if (MC_Stop_0.Done == 1)
        {
            MC_Stop_0.Execute = 0;
            AxisStep = STATE_READY;
        }
        /* check if error occured */
        if (MC_Stop_0.Error == 1)
        {
            singleAx.status.ErrorID = MC_Stop_0.ErrorID;
            MC_Stop_0.Execute = 0;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** START JOG MOVEMENT POSITVE **********************/
    case STATE_JOG_POSITIVE:  /* STATE: Start jog movement in positive direction
*/
        MC_MoveVelocity_0.Velocity      = singleAx.parameter.JogVelocity;
        MC_MoveVelocity_0.Acceleration  = singleAx.parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  = singleAx.parameter.Deceleration;
        MC_MoveVelocity_0.Direction     = mcPOSITIVE_DIR;
        MC_MoveVelocity_0.Execute = 1;
        /* check if jog movement should be stopped */
        if (singleAx.cmd.MoveJogPos == 0)
        {
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_STOP;
        }
        /* check if error occured */
        if (MC_MoveVelocity_0.Error == 1)
        {
```

```
            singleAx.status.ErrorID = MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** START JOG MOVEMENT NEGATIVE **********************/
    case STATE_JOG_NEGATIVE:    /* STATE: Start jog movement in negative direction */
        MC_MoveVelocity_0.Velocity      = singleAx.parameter.JogVelocity;
        MC_MoveVelocity_0.Acceleration  = singleAx.parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  = singleAx.parameter.Deceleration;
        MC_MoveVelocity_0.Direction     = mcNEGATIVE_DIR;
        MC_MoveVelocity_0.Execute = 1;
        /* check if jog movement should be stopped */
        if (singleAx.cmd.MoveJogNeg == 0)
        {
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_STOP;
        }
        /* check if error occured */
        if (MC_MoveVelocity_0.Error == 1)
        {
            singleAx.status.ErrorID = MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** START ABSOLUTE MOVEMENT **********************/
    case STATE_MOVE_ABSOLUTE:  /* STATE: Start absolute movement */


          MC_MoveAbsolute_0.Direction     = singleAx.parameter.Direction;
```

```
	double alpha;
	double beta;
	double alpha_betrag;
	double beta_betrag;
	double epsilon;
	double epsilon_vektor;
	double epsilon_prozent;
	double elevation_korrektur;
	double a;
	double b;
	double c;

	b = Global.parameter.LatOrt * DegreeToRad;
	c = (Global.parameter.LonSat - Global.parameter.LonOrt) * DegreeToRad;
	a = acos(cos(b) * cos(c) + sin(b) * sin(c) * cos(90 * DegreeToRad));
	azimut = ((180 * DegreeToRad) + acos((cos(c) - (cos(a) * cos(b))) / (sin(a) *
sin(b)))) / DegreeToRad;
	azimut = azimut - Global.parameter.North;

	if(azimut>181)
		{
			azimut=azimut-360;
		}

	alpha=Global.parameter.Alpha*DegreeToRad;
	beta=Global.parameter.Beta*DegreeToRad;

	double r_matrix [3] [3] = {
							{cos(beta),0,sin(beta)},
							{sin(alpha)*sin(beta),cos(alpha),-
sin(alpha)*cos(beta)},
							{-
cos(alpha)*sin(beta),sin(alpha),cos(alpha)*cos(beta)}
```

```
                                                    };
        double vektor_v [3] [1] = { {0}, {0}, {1} };

        double vektor_n [3] [1] = {
        {((r_matrix[0][0]*vektor_v[0][0])+(r_matrix[0][1]*vektor_v[0][1])+(r_matrix[0][2]*v
ektor_v[0][2]))},
        {((r_matrix[1][0]*vektor_v[0][0])+(r_matrix[1][1]*vektor_v[0][1])+(r_matrix[1][2]*v
ektor_v[0][2]))},
        {((r_matrix[2][0]*vektor_v[0][0])+(r_matrix[2][1]*vektor_v[0][1])+(r_matrix[2][2]*v
ektor_v[0][2]))} };

        double                                                          zaehler
=((vektor_v[0][0]*vektor_n[0][0])+(vektor_v[1][0]*vektor_n[1][0])+(vektor_v[2][0]*vektor
_n[2][0]));
        double nenner_1 = pow(( (vektor_v[0][0] * vektor_v[0][0]) +
                                        (vektor_v[1][0] * vektor_v[1][0]) +
                                        (vektor_v[2][0] * vektor_v[2][0]) ), 1.0/2.0);
        double nenner_2 = pow(( (vektor_n[0][0] * vektor_n[0][0]) +
                                        (vektor_n[1][0] * vektor_n[1][0]) +
                                        (vektor_n[2][0] * vektor_n[2][0]) ), 1.0/2.0);
        epsilon_vektor = (acos(zaehler / (nenner_1 * nenner_2))/DegreeToRad);
        epsilon = 0;

        if(alpha<0) { alpha_betrag=alpha*(-1); } else { alpha_betrag=alpha; }
        if(beta<0) { beta_betrag=beta*(-1); } else { beta_betrag=beta; }
        if(alpha_betrag>0 && beta_betrag==0)
      {
         epsilon = 90;
      }
        if(alpha_betrag==beta_betrag)
                {
                        epsilon = 45;
                }
        if(alpha_betrag==0 && beta_betrag>0)
```

```
	{
		epsilon = 0;
	}
if(alpha_betrag>beta_betrag)
	{
	double y = beta_betrag/alpha_betrag;
	epsilon = 90-(y*45);
	}
if(beta_betrag>alpha_betrag)
	{
	double y = alpha_betrag/beta_betrag;
	epsilon = y*45;
	}

double azimut_temp;
double azimut_epsilon;
azimut_epsilon=azimut;

if(azimut>(epsilon+90)) { azimut_epsilon=azimut-180; }
if(azimut<(epsilon-90)) { azimut_epsilon=azimut+180; }

azimut_temp=azimut_epsilon-epsilon;
if(azimut_temp<0)
{
	azimut_temp=azimut_temp*(-1);
}

epsilon_prozent=100-(azimut_temp*(1.11111111111111));

if((alpha+beta)<0)
{
elevation_korrektur=epsilon_prozent*epsilon_vektor/100*(-1);
}
else
```

```
        {
        elevation_korrektur=epsilon_prozent*epsilon_vektor/100;
        }

        double d;
        d                                                                    =
sqrt((RadiusErde*RadiusErde+(HoeheSat+RadiusErde)*(HoeheSat+RadiusErde))-
(2*RadiusErde*(HoeheSat+RadiusErde)*cos(b)));
        elevation=(acos((d*d+RadiusErde*RadiusErde-
(HoeheSat+RadiusErde)*(HoeheSat+RadiusErde))/(2*d*RadiusErde))/DegreeToRad
)-horizont;
        elevation=elevation+elevation_korrektur;
        elevation=Ruheposition+elevation-Offset;
        elevation=elevation*10;/* Anpassen auf die Units der Motoren */

                MC_MoveAbsolute_0.Position              = elevation;
      MC_MoveAbsolute_0.Velocity      = singleAx.parameter.Velocity;
      MC_MoveAbsolute_0.Acceleration  = singleAx.parameter.Acceleration;
      MC_MoveAbsolute_0.Deceleration  = singleAx.parameter.Deceleration;
      MC_MoveAbsolute_0.Execute               = 1;

      /* check if commanded position is reached */
       if (singleAx.cmd.Ruhe == 1)
      {
                        MC_MoveAbsolute_0.Position = 0;
                        MC_MoveAbsolute_0.Execute = 1;
                        if (MC_MoveAbsolute_0.Done == 1)
                        {
                                MC_MoveAbsolute_0.Execute = 0;
                                AxisStep = STATE_READY;
                        }
      }
      else
              {
```

```
        if (MC_MoveAbsolute_0.Done == 1)
        {
                MC_MoveAbsolute_0.Execute = 0;
                AxisStep = STATE_MOVE_ABSOLUTE;
        }
      }

     /* check if commanded position is reached */

                if (singleAx.cmd.Stop == 1)
      {
         singleAx.cmd.Stop = 0;
         MC_MoveAbsolute_0.Execute = 0;
         AxisStep = STATE_STOP;
      }
      else if (MC_MoveAbsolute_0.Done == 1)
      {
         MC_MoveAbsolute_0.Execute = 0;
         AxisStep = STATE_MOVE_ABSOLUTE;
      }
                /* check if error occured */
      if (MC_MoveAbsolute_0.Error == 1)
      {
         singleAx.status.ErrorID = MC_MoveAbsolute_0.ErrorID;
         MC_MoveAbsolute_0.Execute = 0;
         AxisStep = STATE_ERROR;
      }
   break;

/******************** START ADDITIVE MOVEMENT **********************/
   case STATE_MOVE_ADDITIVE:  /* STATE: Start additive movement */
      MC_MoveAdditive_0.Distance      = singleAx.parameter.Distance;
      MC_MoveAdditive_0.Velocity      = singleAx.parameter.Velocity;
      MC_MoveAdditive_0.Acceleration  = singleAx.parameter.Acceleration;
```

```
        MC_MoveAdditive_0.Deceleration  = singleAx.parameter.Deceleration;
        MC_MoveAdditive_0.Execute = 1;
        /* check if commanded distance is reached */
        if (singleAx.cmd.Stop == 1)
        {
            singleAx.cmd.Stop = 0;
            MC_MoveAdditive_0.Execute = 0;
            AxisStep = STATE_STOP;
        }
        else if (MC_MoveAdditive_0.Done == 1)
        {
            MC_MoveAdditive_0.Execute = 0;
             AxisStep = STATE_READY;
        }
        /* check if error occured */
        if (MC_MoveAdditive_0.Error == 1)
        {
            singleAx.status.ErrorID = MC_MoveAdditive_0.ErrorID;
            MC_MoveAdditive_0.Execute = 0;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** START VELOCITY MOVEMENT **********************/
    case STATE_MOVE_VELOCITY:  /* STATE: Start velocity movement */
        MC_MoveVelocity_0.Velocity      = singleAx.parameter.Velocity;
        MC_MoveVelocity_0.Acceleration  = singleAx.parameter.Acceleration;
        MC_MoveVelocity_0.Deceleration  = singleAx.parameter.Deceleration;
        MC_MoveVelocity_0.Direction     = singleAx.parameter.Direction;
        MC_MoveVelocity_0.Execute = 1;
        /* check if commanded velocity is reached */
        if (singleAx.cmd.Stop == 1)
        {
            singleAx.cmd.Stop = 0;
```

```
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_STOP;
        }
        else if (MC_MoveVelocity_0.InVelocity == 1)
        {
            MC_MoveVelocity_0.Execute = 0;
             AxisStep = STATE_READY;
        }
        /* check if error occured */
        if (MC_MoveVelocity_0.Error == 1)
        {
            singleAx.status.ErrorID = MC_MoveVelocity_0.ErrorID;
            MC_MoveVelocity_0.Execute = 0;
            AxisStep = STATE_ERROR;
        }
    break;

/******************** FB-ERROR OCCURED ************************/
    case STATE_ERROR:  /* STATE: Error */
        /* check if FB indicates an axis error */
        if (singleAx.status.ErrorID == 29226)
        {
            AxisStep = STATE_ERROR_AXIS;
        }
        else
        {
            if (singleAx.cmd.ErrorAcknowledge == 1)
            {
                singleAx.cmd.ErrorAcknowledge = 0;
                singleAx.status.ErrorID = 0;
                /* reset axis if it is in axis state ErrorStop */
                if ((MC_ReadStatus_0.Errorstop == 1) && (MC_ReadStatus_0.Valid == 1))
                {
                    AxisStep = STATE_ERROR_RESET;
```

```
            }
            else
            {
                AxisStep = STATE_WAIT;
            }
        }
    }
    break;

/******************** AXIS-ERROR OCCURED ************************/
    case STATE_ERROR_AXIS:  /* STATE: Axis Error */
        if (MC_ReadAxisError_0.Valid == 1)
        {
            if (MC_ReadAxisError_0.AxisErrorID != 0)
            {
                singleAx.status.ErrorID = MC_ReadAxisError_0.AxisErrorID;
            }
            MC_ReadAxisError_0.Acknowledge = 0;
            if (singleAx.cmd.ErrorAcknowledge == 1)
            {
                singleAx.cmd.ErrorAcknowledge = 0;
                /* acknowledge axis error */
                if (MC_ReadAxisError_0.AxisErrorID != 0)
                {
                    MC_ReadAxisError_0.Acknowledge = 1;
                }
            }
            if (MC_ReadAxisError_0.AxisErrorCount == 0)
            {
                singleAx.status.ErrorID = 0;
                /* reset axis if it is in axis state ErrorStop */
                if ((MC_ReadStatus_0.Errorstop == 1) && (MC_ReadStatus_0.Valid == 1))
                {
                    AxisStep = STATE_ERROR_RESET;
```

```
            }
            else
            {
                AxisStep = STATE_WAIT;
            }
        }
    }
    break;

/******************** RESET DONE ************************/
    case STATE_ERROR_RESET:  /* STATE: Wait for reset done */
        MC_Reset_0.Execute = 1;
        /* reset MC_Power.Enable if this FB is in Error*/
        if(MC_Power_0.Error == 1)
        {
            MC_Power_0.Enable = 0;
        }
        if(MC_Reset_0.Done == 1)
        {
            MC_Reset_0.Execute = 0;
            AxisStep = STATE_WAIT;
        }
    break;
/******************** SEQUENCE END ************************/
}

/*****************************************************************
        Function Block Calls
*****************************************************************/

/************************** MC_POWER ****************************/
MC_Power_0.Axis = Axis1Obj;  /* pointer to axis */
MC_Power(&MC_Power_0);
```

```
/************************* MC_HOME ****************************/
MC_Home_0.Axis = Axis1Obj;
MC_Home(&MC_Home_0);

/********************** MC_MOVEABSOLUTE ************************/
MC_MoveAbsolute_0.Axis = Axis1Obj;
MC_MoveAbsolute(&MC_MoveAbsolute_0);

/********************** MC_MOVEADDITIVE ************************/
MC_MoveAdditive_0.Axis = Axis1Obj;
MC_MoveAdditive(&MC_MoveAdditive_0);

/********************** MC_MOVEVELOCITY ************************/
MC_MoveVelocity_0.Axis = Axis1Obj;
MC_MoveVelocity(&MC_MoveVelocity_0);

/************************* MC_STOP ****************************/
MC_Stop_0.Axis = Axis1Obj;
MC_Stop(&MC_Stop_0);

/************************* MC_RESET ****************************/
MC_Reset_0.Axis = Axis1Obj;
MC_Reset(&MC_Reset_0);

/************************ MC_READAXISERROR **********************/
MC_ReadAxisError_0.Enable = !MC_ReadAxisError_0.Error;
MC_ReadAxisError_0.Axis = Axis1Obj;
MC_ReadAxisError_0.DataAddress = (UDINT)&(singleAx.status.ErrorText);
MC_ReadAxisError_0.DataLength = sizeof(singleAx.status.ErrorText);
strcpy((void*)&MC_ReadAxisError_0.DataObjectName,"english");
MC_ReadAxisError(&MC_ReadAxisError_0);

/************************ MC_READSTATUS ************************/
MC_ReadStatus_0.Enable = !MC_ReadStatus_0.Error;
```

```
MC_ReadStatus_0.Axis = Axis1Obj;
MC_ReadStatus(&MC_ReadStatus_0);
singleAx.axisState.Disabled            = MC_ReadStatus_0.Disabled;
singleAx.axisState.StandStill          = MC_ReadStatus_0.StandStill;
singleAx.axisState.Stopping            = MC_ReadStatus_0.Stopping;
singleAx.axisState.Homing              = MC_ReadStatus_0.Homing;
singleAx.axisState.DiscreteMotion      = MC_ReadStatus_0.DiscreteMotion;
singleAx.axisState.ContinuousMotion    = MC_ReadStatus_0.ContinuousMotion;
singleAx.axisState.SynchronizedMotion  = MC_ReadStatus_0.SynchronizedMotion;
singleAx.axisState.ErrorStop           = MC_ReadStatus_0.Errorstop;
/******************** MC_SETOVERRIDE *********************/
MC_SetOverride_0.Enable            =           (!MC_ReadStatus_0.StandStill           &
!MC_SetOverride_0.Error & MC_Power_0.Status);
MC_SetOverride_0.Axis = Axis1Obj;
MC_SetOverride_0.VelFactor = singleAx.parameter.OverrideVelocity;
MC_SetOverride_0.AccFactor = singleAx.parameter.OverrideAcceleration;
MC_SetOverride(&MC_SetOverride_0);

/******************** MC_READACTUALPOSITION *********************/
MC_ReadActualPosition_0.Enable      =      (!MC_ReadActualPosition_0.Error      &
MC_Power_0.Status);
MC_ReadActualPosition_0.Axis = Axis1Obj;
MC_ReadActualPosition(&MC_ReadActualPosition_0);
if(MC_ReadActualPosition_0.Valid == 1)
{
   singleAx.status.ActPosition = MC_ReadActualPosition_0.Position;
}

/******************** MC_READACTUALVELOCITY *********************/
MC_ReadActualVelocity_0.Enable      =      (!MC_ReadActualVelocity_0.Error      &
MC_Power_0.Status);
MC_ReadActualVelocity_0.Axis = Axis1Obj;
MC_ReadActualVelocity(&MC_ReadActualVelocity_0);
if(MC_ReadActualVelocity_0.Valid == 1)
```

```
{
    singleAx.status.ActVelocity = MC_ReadActualVelocity_0.Velocity;
}
}
```