



Fakultät für Ingenieurwissenschaften -
Studiengang Mechatronik

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering (B. Eng.)

Echtzeitregelung der Bahn einer Kugel auf einer Platte mit verstellbaren Neigungswinkeln

Autor: Vadim Dammer

Erstprüfer: Prof. Dr.-Ing. Peter Zentgraf, M.Sc.

Zweitprüfer: Dipl.-Ing. (FH) Martin Heigl

Datum der Abgabe: 17.04.2014

Danksagung

An dieser Stelle möchte ich mich recht herzlich bei der Hochschule Rosenheim bedanken, welche mir die Möglichkeit, die notwendigen Unterlagen und das nötige Vertrauen zur Ausarbeitung meiner Bachelorarbeit zur Verfügung stellte, sowie bei allen Personen, die mich tatkräftig unterstützt haben.

Ich danke Herrn Prof. Dr.-Ing. Peter Zentgraf, M.Sc. und Herrn Dipl.-Ing. (FH) Martin Heigl, die mich bei Fragen und Anliegen zu meiner Bachelorarbeit jederzeit unterstützt haben.

Im Besonderen danke ich Herrn Dipl.-Ing. (FH) Peter Crämer für die Unterstützung mit zahlreichen Tipps und Anmerkungen rund um das Thema „DLL“.

Kurzfassung

Diese Bachelorarbeit besteht hauptsächlich aus der Entwicklung und Realisierung eines Prinzips, bei dem es sich um die Regelung der schwenkbaren phyCARD Plattform mit zwei Servomotoren handelt. Eine Stahlkugel, die auf das Touchpanel vom phyCARD gelegt wird, soll eine zuvor aufgenommene oder vorprogrammierte Bahn durch die Schwenkbewegungen der X- und Y-Achse fahren. Dies wird dadurch erreicht, dass das Simulinkmodell des Reglers in Form einer DLL-Datei in die Software, die auf der phyCARD Plattform läuft, integriert wird. Desweiteren besteht diese wissenschaftliche Arbeit aus der Einarbeitung einer dafür passenden mechanischen Konstruktion mit möglichst kleinem Spiel während des Betriebes. Eine der wichtigsten Rollen in der Bachelorarbeit spielt der Aufbau und Test der Schnittstelle zwischen dem Simulink und dem Hauptprogramm der phyCARD Plattform.

Die Hochschule Rosenheim will mit der Themenstellung dieser Bachelorarbeit eine kontinuierliche Entwicklung und Optimierung von Simulationsprozessen fördern. Dadurch erhofft man sich, das Spektrum der Simulationstechniken für Studenten sowie Mitarbeiter der Hochschule Rosenheim noch weiter zu verbreiten. Anhand dieser Bachelorarbeit können weitere Ideen und Projekte entwickelt werden, in denen das Thema automatische Stabilisierung der Lage eingesetzt werden kann.

Abstract

This thesis mainly consists of the development and realization of a principle, which is comes to the regulation of the pivotal phyCARD platform with two servo motors. A steel ball, placed on the touch panel from phyCARD, should drive a pre-recorded or pre-programmed path through the pivoting movements of the X - and Y -axis. This is achieved by integrating Simulink model of the controller is integrated in the form of a DLL file in the software, that runs on the phyCARD platform. Further, there is this scientific work from the discovery of a for suitable mechanical design with the smallest possible game during operation. One of the most important parts in the thesis is the construction and testing of the interface between Simulink and the main program of the phyCARD platform.

The reason for the Rosenheim University of Applied Sciences to publish this bachelor thesis is a continuous development and optimization of simulation processes and their applications. It is hoped the range of simulation techniques for the students and employees of the University of Rosenheim to spread even further. On the basis of this thesis, further ideas and projects can be developed in which the topic of automatic stabilization of the situation can be used.

Inhaltsverzeichnis

Danksagung	2
Kurzfassung.....	3
Abstract.....	4
Inhaltsverzeichnis	5
Abbildungsverzeichnis.....	7
Tabellenverzeichnis	8
Abkürzungsverzeichnis	9
Formelsammlung	10
Fremdwörter und Fachbegriffe	11
1 Einleitung	12
1.1 Entwicklung von mechanischen und elektrischen Reglern	12
1.2 Eingebettete Systeme.....	14
2 Aufgabenstellung dieser Bachelorarbeit	16
3 Analyse des Ist-Zustandes.....	17
4 Konzepterstellungphase	18
5 Auswahl von geeigneten Komponenten.....	19
5.1 Aktuatoren	19
5.2 Programme	19
6 Konstruktionsphase.....	24
7 Montage und Test der Konstruktion	28
7.1 Montage.....	28
7.2 Test	28
8 Verbesserung der Konstruktion.....	29
9 Entwicklungsphase	30
9.1 Erstellung und Einbindung der DLL für Desktop Systeme.....	30
9.1.1 Erstellung der DLL im Matlab	30
9.1.2 Erstellung der DLL im Simulink	31
9.1.3 Einbindung der im Matlab erstellten DLL im Visual Studio 2005	33

9.1.4	Einbindung der im Simulink erstellten DLL im Visual Studio 2005	34
9.2	Erstellung und Einbindung der DLL für eingebettete Systeme.....	35
9.2.1	Erstellung von C++ Code im Simulink	35
9.2.2	Erstellung der DLL aus dem C++ Code im Visual Studio 2005	36
9.2.3	Einbindung der DLL im Visual Studio 2005	37
10	Test der Schnittstelle	40
10.1	Benutzung der SQL Datenbank.....	40
10.2	Vergleich von PhyCARD und Simulink Ergebnissen.....	41
10.3	Sprungantwort des Systems.....	47
11	Resümee der Bachelorarbeit	49
	Anhang.....	50
	Literaturverzeichnis	62
	Erklärung	63

Abbildungsverzeichnis

Abbildung 1, Dampfmaschine von James Watt, Quelle: http://idahospudsblog.blogspot.de	12
Abbildung 2. Zentrifugal-Regulator, Quelle: http://www.visualphotos.com	13
Abbildung 3 Stabilisierung eines Tischtennisballs mittels Bildverarbeitung	17
Abbildung 4 Matlab, Hauptfenster	20
Abbildung 5 Matlab, Plot 3D.....	21
Abbildung 6 Simulink, Library Browser	22
Abbildung 7 Simulink, unterstützte Compiler	22
Abbildung 8 Visual Studio 2010, Hauptfenster	23
Abbildung 9 PhyCARD Plattform	24
Abbildung 10 Zwischenrahmen für die PhyCARD Platinen	25
Abbildung 11 PhyCARD Plattform	26
Abbildung 12 Gehäuse aus Acrylglas.....	27
Abbildung 13 Komplette Konstruktion	27
Abbildung 14 Matlab, Definition der Eingangsvariablen.....	31
Abbildung 15 Dependency Walker, Auflistung der externen Funktionen	31
Abbildung 16 Simulinkmodell mit der Addition und Multiplikation.....	32
Abbildung 17 Simulink, empfohlene Einstellungen.....	32
Abbildung 18 Dependency Walker, Externe Funktionen der DLL	33
Abbildung 19 VS 2005, Platz zum Einfügen des Codes	34
Abbildung 20 Simulink-Modell, P-Regler.....	35
Abbildung 21 VS 2005, Projektordner	36
Abbildung 22 IP-Einstellungen	40
Abbildung 23 PhyCARD Datenbank.....	41
Abbildung 24 Eingangswerte für die X-Achse des Reglers	42
Abbildung 25 Ausgangswerte des Reglers für die X-Achse	43
Abbildung 26 Eingangswerte für die Y-Achse des Reglers	44
Abbildung 27 Ausgangswerte des Reglers für die Y-Achse	44
Abbildung 28 Simulink-Modell zum Vergleich von Ergebnissen.....	45
Abbildung 29 Simulink-Modell, das in der DLL getestet wurde	45
Abbildung 30 Ausgangswerte des Simulinkmodells für die X-Achse	46
Abbildung 31 Ausgangswerte des Simulinkmodells für die Y-Achse	47
Abbildung 32 Sprungantwort des Systems für die X-Achse	48
Abbildung 33 Sprungantwort des Systems für die Y-Achse	48

Tabellenverzeichnis

Tabelle 1: PhyCARD, Technische Daten, Quelle: http://www.phytec.de15
Tabelle 2: Savöx SC-1257TG, Technische Daten19

Abkürzungsverzeichnis

z.B.	zum Beispiel
bzw.	beziehungsweise
Abb.	Abbildung
ggf.	gegebenenfalls
u.U.	unter Umständen
sog.	sogenannten
zw.	Zwischen
evtl.	eventuell
max.	maximal
ms.	Millisekunde
kont.	kontinuierlich
Nr.	Nummer
ca.	circa
engl.	englisch

Formelsammlung

Bezeichnung	Symbol	Einheiten
Drehmoment	M	Newton/ Zentimeter
Frequenz	f	Herz
Spannung	U	Volt
Masse	m	Kilogramm
Winkelgeschwindigkeit	ω	Radian/Sekunde
Radius	r	Meter
Erdbeschleunigung	g	Meter/sec ²

Fremdwörter und Fachbegriffe

PWM-Signal	Bei der Pulsweitenmodulation (engl. Pulse Width Modulation , abgekürzt PWM) wird die Ein- und Ausschaltzeit eines Rechtecksignals bei fester Grundfrequenz variiert.
Servomotor:	Als Servomotor werden elektrische Motoren verschiedener Bauart bezeichnet, die mit einem Servoregler in geschlossenen Regelkreisen betrieben werden. Durch das PWM-Signal können sie schrittweise gesteuert werden.
DLL:	Eine DLL (D ynamic L ink L ibrary) ist eine Bibliothek, die Code und Daten enthält. Sie kann gleichzeitig von mehreren Programmen verwendet werden.

1 Einleitung

1.1 Entwicklung von mechanischen und elektrischen Reglern

In letzter Zeit gewinnt die Simulation komplexer Systeme immer mehr an Bedeutung. Teilweise liegt es daran, dass es immer schwieriger wird, komplexe Systeme „per Hand“ auszulegen bzw. oft ist es sogar ohne Rechnerunterstützung gar nicht mehr möglich. Zusätzlich wird die Zeit, über die man verfügt, um neue Systeme zu entwickeln und zu testen, wird immer knapper. Ein anderer wichtiger Grund ist die Möglichkeit, die Ergebnisse der Simulation in anderen Rechnergestützten Programmen zu speichern und zu bearbeiten. Ein solcher Prozess ist heutzutage ein Muss für große Konzerne und Firmen, um Geld und Zeit während der Entwicklungsphase zu sparen. Es betrifft auch Organisationen, in denen man ständig in Bereichen Forschung und Entwicklung tätig ist. Zu diesen gehören ohne Zweifel solche Institutionen wie Hochschulen und Universitäten. Das Thema Simulation ist sehr stark mit dem Thema Regelungstechnik verbunden. Zu den Zeiten, als es noch keine Rechner gegeben hat, haben Wissenschaftler und Ingenieure versucht, Regelungen durch unterschiedliche Mechanismen zu realisieren. Ein gutes Beispiel hierfür ist die Dampfmaschine von James Watt (Abb. 1).

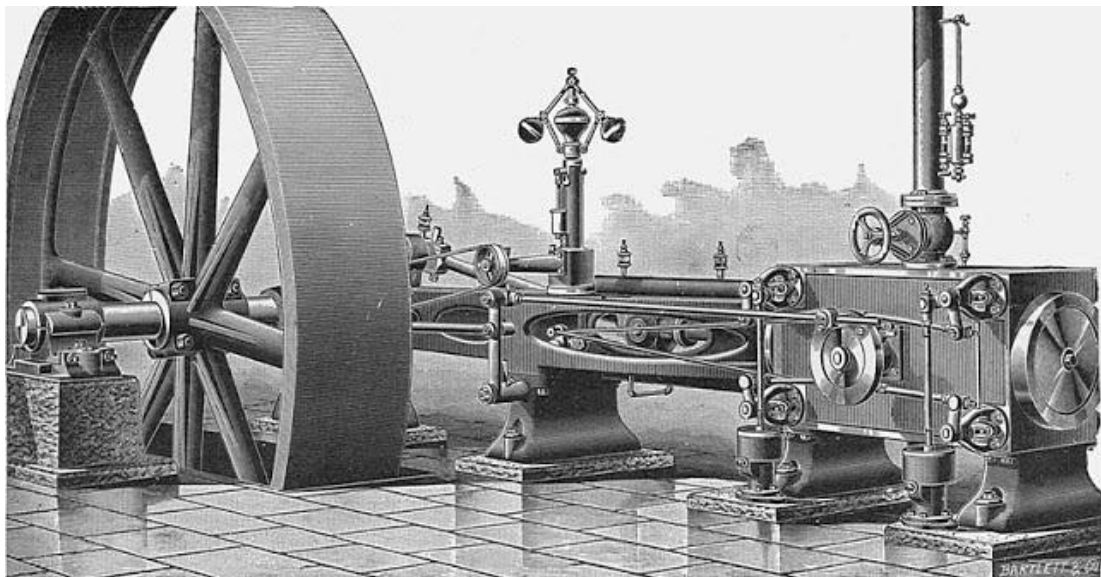


Abbildung 1, Dampfmaschine von James Watt, Quelle:
<http://idahospudsblog.blogspot.de>

Die wattsche Dampfmaschine besteht aus einem Kessel, der mit Kohle geheizt wird, und dem Kolbenapparat mit Pleuelstange und Schwungrad, der den Dampfdruck in eine Drehbewegung umwandelt. In der Dampfleitung zwischen Kessel und Kolben befindet sich ein Ventil. Wird der Dampfmaschine eine höhere mechanische Leistung abverlangt, so wird die Drehzahl geringer. Mit genau diesem Problem hatten die Maschinisten der Dampfmaschinen zu kämpfen. Sie mussten Kohle in den Kessel schaufeln und gleichzeitig die Drehzahl der Dampfmaschine überwachen sowie das Ventil zwischen dem Kessel und Kolben einstellen, damit die Drehzahl konstant blieb. Um das Problem mit dem Einstellen vom Ventil zu lösen und den Vorgang zu automatisieren, hat James Watt im Jahr 1788 den „Zentrifugal-Regulator“ (Abb. 2) erfunden.

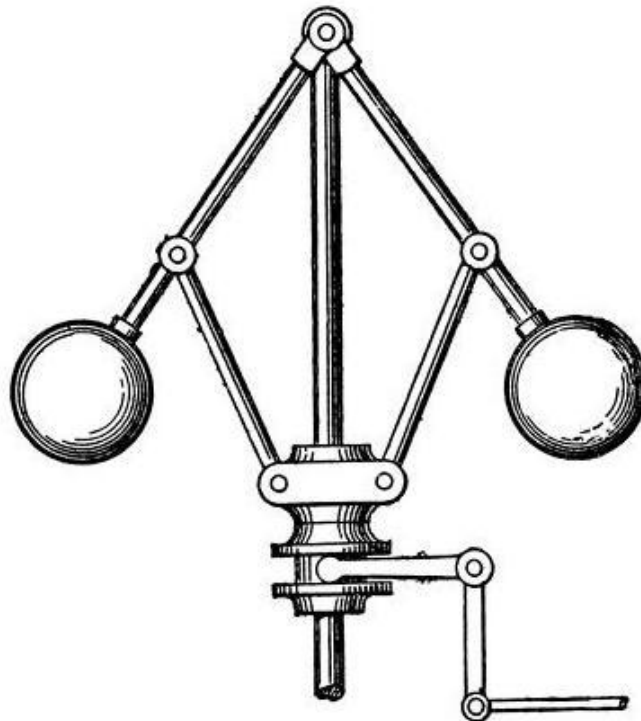


Abbildung 2. Zentrifugal-Regulator, Quelle: <http://www.visualphotos.com>

Wie man in der Abbildung sieht, ist das Funktionsprinzip ziemlich einfach und leicht nachvollziehbar. Die Kugelgewichte werden durch die vertikale Welle angetrieben, die über einen Riemen (hier z.B. eine Kurbel) mit der Dampfmaschine verbunden ist. Auf die Pendel (Masse) wirken die Gewichtskraft mg (als Sollwert) sowie die Fliehkraft $m\omega^2$ (als Messwert), die das Produkt aus der Masse m , dem Radius r und dem Quadrat der Winkelgeschwindigkeit ω ist. Wenn nun die Drehzahl zu groß wird, dann bewegen sich die beiden Pendel nach außen. Über den Hebelmechanismus bewirkt die Pendelauslenkung, dass sich die Stange nach unten bewegt und das Drosselventil für den Dampf schließt, woraufhin die Dampfmaschine weniger Dampfzufuhr erhält und damit langsamer dreht. Der Regelkreis ist damit geschlossen und die Wirkung des Regelkrei-

ses klar: Drehzahl zu hoch → Auslenkung des Pendelapparates → Drosselventil zu → Drehzahl wird geringer. Der Regler vom James Watt war damals so revolutionär und neu, dass es keine Zweifel gab, in welche Richtung die Menschheit gehen wird. Allerdings wiesen mechanische Regler Schwächen wie Verschleiß und geringe Frequenz auf. Mit der Entwicklung von Computergestützten Systemen konnten diese Schwächen behoben werden. Nun konnte man schnellere und vor allem komplexere Regler entwerfen. Ein weiteres Problem war jedoch das, dass man sich mit den spezifischen Sprachen wie Assembler sehr gut auskennen sollte, um einen Regler zu entwerfen. Eine Erleichterung hat die Einführung der C Sprache im Jahr 1970 mit sich gebracht. Die C Sprache war wesentlich einfacher als Assembler und trotzdem war das Auslegen von den Reglern immer noch umständlich. Ende der 1970er Jahre wurde das Programm Matlab entwickelt, das auch heute vor allem für numerische Simulation sowie Datenerfassung, Datenanalyse und -auswertung eingesetzt wird. Nun musste man keine einzige Zeile C-Code schreiben, sondern man sollte nun nur die sogenannte „mathematische“ Sprache beherrschen. Ein wichtiges Teil des Programms Matlab ist das Simulink. Wie man schon aus dem Namen erraten kann, ist dieses Programm für die Simulationsvorgänge zuständig. Das Simulink stellt hierarchische Modellierung mit Hilfe grafischer Blöcke dar. Zwischen den Blöcken wird der Datenfluss grafisch über die Verbindungslinien realisiert. Ein zu prüfendes System kann dann innerhalb von Simulink erstellt und simuliert werden. Mit solchen mächtigen Werkzeugen können Regler fast unbegrenzter Komplexität entworfen werden. Nach dem der Regler ein im Simulink „richtiges“ Verhalten bekommen hat, könnte man ihn in Form vom C Code oder einer DLL auf ein Eingebettetes System laden.

1.2 Eingebettete Systeme

Unter dem Begriff „eingebetteten Systeme“ versteht man Hard- und Softwaresysteme, die komplexe Steuerungs-, Regelungs- und Datenverarbeitungsaufgaben übernehmen und in umgebende technische Systeme eingebettet sind. Durch die Miniaturisierung der Komponenten wird das Anwendungsspektrum eingebetteter Systeme weiter erweitert und gleichzeitig die Leistung und Ausstattung durch die Unterstützung unterschiedlicher Schnittstellen erhöht. Die steigende Produktion und weltweite Verwendung von eingebetteten Systemen sorgen dafür, dass solche Systeme immer mehr an Bedeutung im Alltag gewinnen und ihre Kosten kontinuierlich sinken. Die eingebetteten Mikrosysteme werden typischerweise in den Bereichen Steuerung und Regelung von Produktionsprozessen, Automobilindustrie sowie Telekommunikation und Raum- und Luftfahrt eingesetzt. Im Hinblick auf die Entwicklung moderner Methoden zum Regler Entwerfen, ist es möglich geworden, mit solchen Programmen wie Simulink unterschiedliche Mikrocontroller zu programmieren. Dieser Schritt hat eine sehr große Bedeutung für Entwickler und Forscher, da man nun den mühsamen Schritt mit dem Programmieren

von Mikrocontrollern „umgehen“ kann. Das einzige, worauf nun geachtet werden sollte, ist ein richtig funktionierender Regler. Dessen Konvertierung und anschließendes Laden (engl. flashen) auf den Mikrocontroller übernimmt das Programm Simulink. Im Rahmen dieser Bachelorarbeit wurde entschieden, die Schnittstelle zwischen dem Simulink und dem Evaluation Board der Firma PHYTEC aufzubauen und zu testen. Die WinCE-phyCARD-i.MX35x Plattform entspricht dem heutigen Stand der Technik und hat neben der Echtzeitfähigkeit ein resistives Touchdisplay.

Tabelle 1: PhyCARD, Technische Daten, Quelle: <http://www.phytec.de>

Software	
Betriebssystem	WinCE 6.0
Echtzeit	ja
BSP / Image	ja
Bootloader	Eboot
Toolchain	Platform Builder
Compiler	Visual Studio
Debug-Interface	JTAG (auf Modul)
CPU	Freescale i.MX35
Architektur	ARM11
Hersteller	Freescale
Taktfrequenz	532 MHz
Speicher	
NAND Flash	128 MB
DDR RAM	128 MB
EEPROM	4 kB

Dank der leistungsfähigen Prozessoren und schnellem DDR Arbeitsspeicher, die in den eingebetteten Systemen verwendet werden, stellen die Simulationen und Messungen im Echtzeitbetrieb heutzutage kein Problem dar.

2 Aufgabenstellung dieser Bachelorarbeit

Im Rahmen dieser wissenschaftlichen Arbeit sollte es gezeigt werden, dass alle gestellten Ziele durch die während des Studiums erlernten Fähigkeiten und selbständiges und strukturiertes Denken erledigt werden konnten. Man sollte ebenfalls als Ingenieur den gesamten Aufwand einschätzen können, um die Zeit und Ressourcen sinnvoll einzuplanen.

Es wurden folgende Ziele gesetzt:

1. Entwurf eines Konzeptes für die schwenkbare PhyCARD Plattform
2. Suche nach passenden Aktuatoren im System.
3. Konstruktion des Konzeptes im CAD Programm Solid Edge.
4. Fertigung und Kontrolle der gefertigten Teile.
5. Zusammenbau der Konstruktion und anschließender Test.
6. Eventuelle Nachbesserungen der Konstruktion/ Aktuatoren.
7. Das System soll tragbar sein und eine eigene Energiequelle haben.
8. Das Gesamtgewicht des Systems darf 5 Kg nicht überschreiten.
9. Ein Gehäuse zum besseren Aufbewahren sollte ebenfalls gebaut werden.
10. Auslegen eines einfachen Reglers für Testzwecke.
11. Suche eines optimalen Weges zum Konvertieren und Laden des entworfenen Reglers auf die PhyCARD Plattform.
12. Aufbau der Schnittstelle zwischen dem Simulink und der PhyCARD Plattform.
13. Aufbau der Schnittstelle zwischen dem Simulink und der PhyCARD Plattform.
14. Testen der Schnittstelle und eventuelle Nachbesserungen.
15. Endgültiger Test des Konzeptes.
16. Vergleich der Ergebnisse auf der PhyCARD Plattform mit den Ergebnissen im Simulink.

3 Analyse des Ist-Zustandes.

Die ersten Recherchen haben gezeigt, dass es generell viele Systeme gibt, die ähnliche Funktionsweisen aufweisen, wie das gesuchte System. An der Hochschule Rosenheim ist sogar ein ähnliches Projekt bei dem Herrn Prof. Dr.-Ing. Birger Mysliwetz vorhanden, in dem die Studenten einen Tischtennisball mittels Bildbearbeitung, einer Platte, zwei Servomotoren und einem Mikrocontroller mit entsprechender Software auf die Mitte der Platte stabilisiert haben (Abb. 3).

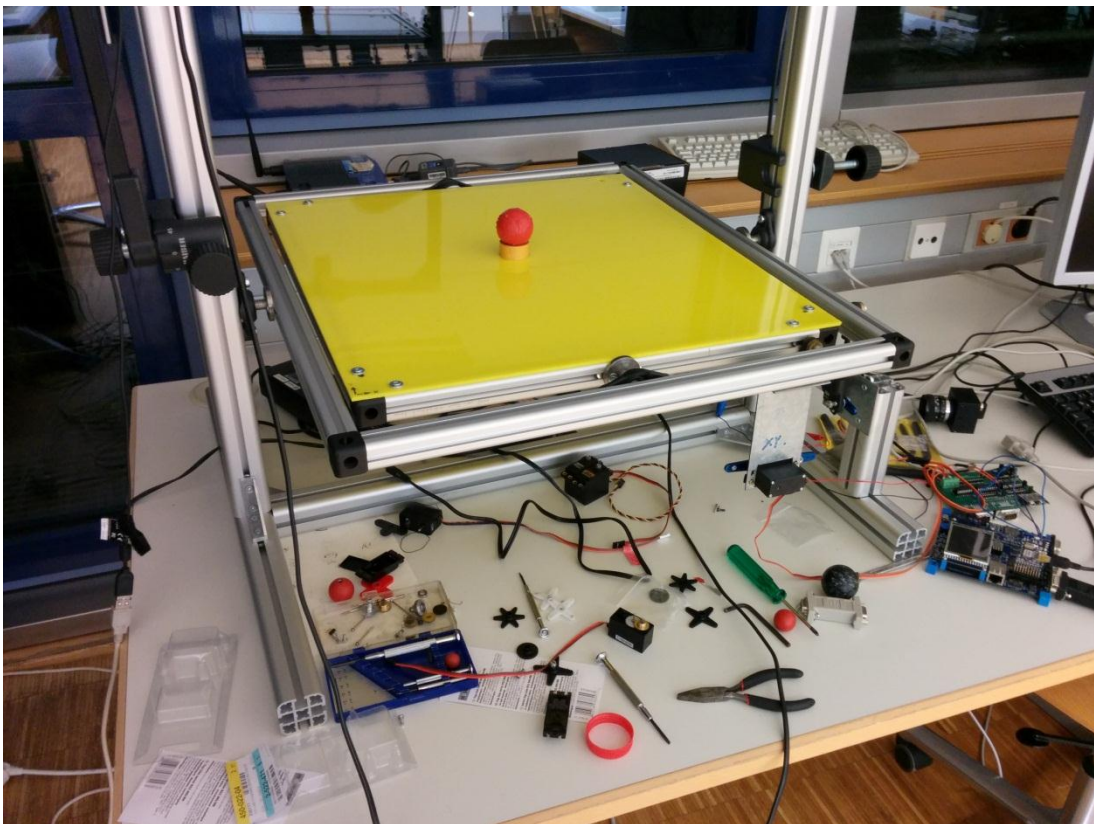


Abbildung 3 Stabilisierung eines Tischtennisballs mittels Bildverarbeitung

Nach dem Treffen mit dem Herrn Prof. Dr.-Ing. Birger Mysliwetz und recherchieren von Videos von ähnlichen Projekten im Internet war klar, dass solche Eingebettete Systeme mit Reglern wirklich realisierbar sind.

4 Konzepterstellungsphase

Wie man auf der Abbildung 3 erkennen kann, befinden sich die Servomotoren am Rand der Platte und sind mit der mittels Koppelstangen verbunden. Da durch solche Verbindungen oft ein unnötiges Spiel in der gesamten mechanischen Konstruktion entstehen kann, das dann auf die Regelgenauigkeit einen negativen Einfluss hat, wurde beschlossen, die beiden Servomotoren an den Drehachsen des Touchdisplays der PhyCARD Plattform zu befestigen. Folglich sind die Drehmittelpunkte der Servomotoren und die Drehachsen des Displays kollinear. Da die mechanische Konstruktion aus mehreren Komponenten bestehen sollte, sollte man sich vorher schon Gedanken machen, wie die Reihenfolge beim Zusammenbau erfolgen wird, damit nur geringes oder gar kein Spiel im System entsteht und sich das System trotzdem ohne Schwierigkeiten montieren lässt. Das System wird in Zukunft eventuell auch anderen Studenten präsentiert, daher waren die kleinen Abmaßen sowie geringes Gewicht empfohlen. Um das Gewicht zu reduzieren und trotzdem genügende Festigkeit zu haben, sollten alle Teile für die mechanische Konstruktion aus Aluminium gefräst werden. Um die Anforderungen an geringen Abmaßen einzuhalten, war die Position und Befestigung der Servomotoren sehr gut zu überlegen. Ein anderer wichtiger Punkt war die Befestigung der PhyCARD selbst. Da sie aus zwei Platinen besteht, war ein Zwischenrahmen notwendig, der diese beiden Platinen miteinander verbindet und gleichzeitig recht stabil ist.

5 Auswahl von geeigneten Komponenten

5.1 Aktuatoren

Nach einer langen Suche nach den geeigneten Servomotoren im Internet und dem Vergleich vieler Parameter, hat man sich für die digitalen Servomotoren der Firma Savöx entschieden (an dieser Stelle sollte darauf hingewiesen werden dass diese Servomotoren später gegen Motoren der Firma Futaba ersetzt wurden).

Tabelle 2: Savöx SC-1257TG, Technische Daten

Betriebsspannung	4,8 Volt	6,0 Volt
Stellzeit 60° (sek)	0,09	0,07
Stellkraft (Ncm)	80	100
Stellkraft (Kg)	8	10
Betriebsspannung (V)	4,8 bis 6,0	
Mittenimpuls ca. (μ s)	1500	
Max. Impulsfolge (Hz)	333	
Anzahl Kugellager	2	
Servomotor	Glockenankermotor	
Servogetriebe	Titangetriebe	
Servogehäuse	Teilweise Alu	

Die kurze Stellzeit der Servomotoren, sowie eine akzeptable Genauigkeit und das Titangetriebe sorgen für eine zuverlässige und schnelle Arbeit des Systems.

5.2 Programme

Von Anfang an hat man den Schwerpunkt auf die Integrierung des Reglers durch die Einbindung von C-Code gelegt und dann, sobald diese Lösung funktionierte, konnte man sie durch die Erstellung und Einbindung der DLL-Datei ersetzen. Die DLL-Dateien sollten sowohl vom Matlab als auch vom Simulink erstellt werden können. Auf der PhyCARD Plattform läuft das Betriebssystem Windows CE 6.0 x86. Um die Kompatibilität zwischen Systemen zu gewährleisten bzw. Probleme bei der Erstellung bzw. Portierung der DLL-Dateien auf die PhyCARD Plattform zu vermeiden, hat man bei der

Entwicklung der Schnittstelle zwischen dem Simulink und der PhyCARD Windows 7 Professional x86 verwendet.

Folgende Programme wurden während der Bachelorarbeit benutzt:

- Matlab/ Simulink
- Microsoft Visual Studio 2005 Professional + PhyCARD Plattform Builder
- Microsoft Visual Studio 2010 Professional
- Solid Edge
- Dependency Walker

Mit Hilfe vom Programm Dependency Walker war es möglich, die Funktionen, die in einer DLL-Datei als externe Funktionen deklariert wurden, anzusehen. Das gilt aber nur für die DLL-Dateien, die für die Desktop Version erstellt wurden. Das heißt, die DLL-Dateien für die PhyCARD Plattform können auf diese Art und Weise aufgrund der Kompatibilitätsprobleme nicht geöffnet werden.

Das Programm Matlab ist heutzutage ein Standard Programm für alle möglichen Berechnungen. Das Hauptfenster des Programms (Abb. 4) ist in fünf bis sechs Teile gegliedert.

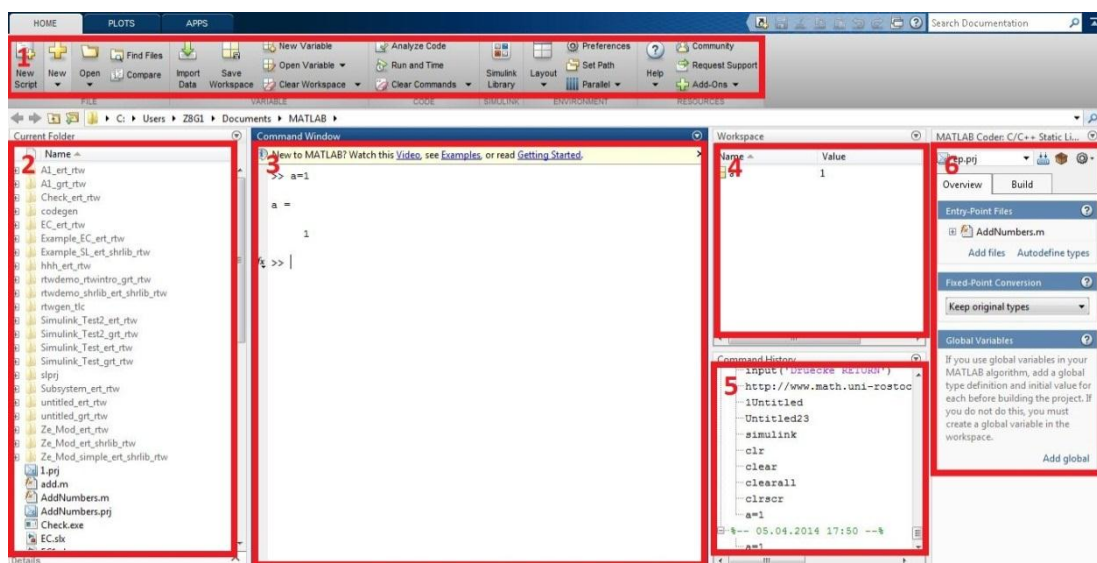


Abbildung 4 Matlab, Hauptfenster

In der Abbildung 4 werden die wichtigsten Menüs und Fenster angezeigt. Im rot markierten Bereich mit Nummer 1 sind die wichtigsten Befehle sowie Menüs platziert. Im Bereich zwei befindet sich das Teil „Current Folder“ – hier werden alle relevanten Dateien im Ordner Matlab angezeigt. Der dritte Bereich ist die Commandozeile, in der alle Befehle eingegeben werden. Im „Command History“ (fünfter Bereich) kann man alle zuvor eingegebenen Befehle ansehen. Um das sechste Fenster anzeigen zu lassen, sollte man in der Commandozeile den Befehl „Coder“ eingeben. Dies wird benötigt, um z.B. aus einer Scriptdatei den C Code oder eine DLL-Datei generieren zu lassen.

Folgendes Beispiel sollte noch mal zeigen, wie sich das Programmieren und vor allem die Darstellung von Diagrammen mit Hilfe vom Matlab vereinfacht haben. Den unten stehenden Code könnte man direkt in der Commandozeile ausführen oder aus einer Scriptdatei ausführen.

```
t = [0:pi/50:10*pi];  
x = sin(t);  
y = cos(t);  
z = t;  
plot3(x,y,z,'r-')  
input('Druecke RETURN')
```

Das Ergebnis sieht dann folgendermaßen aus (Abb. 5).

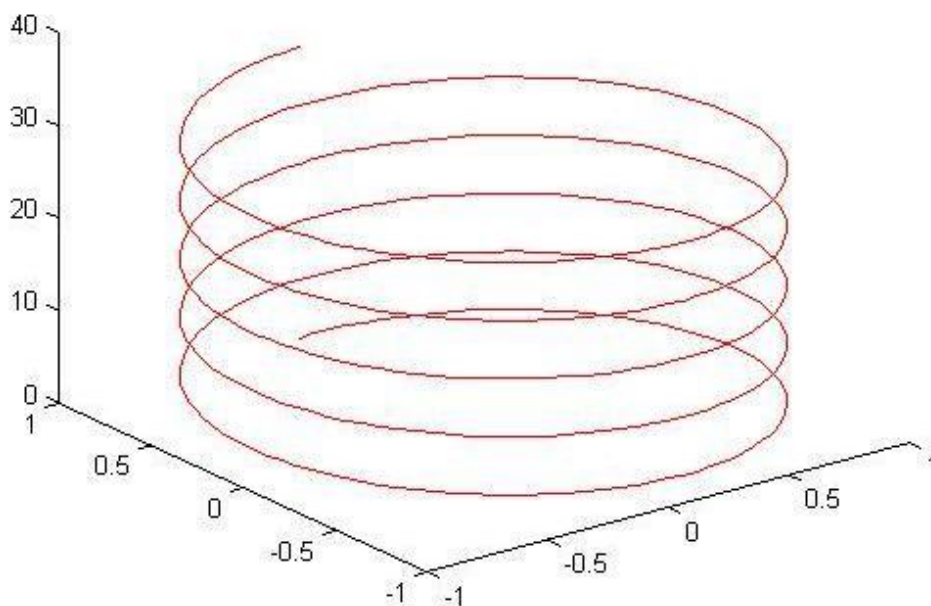


Abbildung 5 Matlab, Plot 3D

Das Matlab ist in der Lage diesen Code zu kompilieren und ihn in andere Sprachen, wie z.B. am meisten verbreitete Programmiersprache C, zu konvertieren. Dieser Schritt hat dazu beigetragen, dass wesentlich komplexere und umfangreiche Berechnungen nicht mehr im C Code programmiert werden müssen.

Man kann nur raten, wie viel Aufwand und Zeit ein Programmierer braucht, um das gleiche Ergebnis in einer anderen Sprache erzielen zu können. Matlab ist auch in der Lage, große Datenmengen aus unterschiedlichen Dateien, wie z.B. *.csv ein- und auszu-lesen und zu bearbeiten.

Ein wichtiger Teil von Matlab ist das Programm Simulink. Mit dem Programm können beliebige Regler entworfen und Prozesse simuliert werden. Für die numerische Simulation stehen ebenfalls verschiedene Lösungsverfahren (engl. "Solver") zur Verfügung.

In der Bibliothek Simulink Library Browser (Abb. 6) befinden sich alle Blöcke, mit deren Hilfe Regler ausgelegt und Prozesse simuliert werden können. Mithilfe geeigneter Toolboxen ist es möglich, aus Matlab/Simulink heraus fertigen Code (C und VHDL) für Mikroprozessoren, Computer und FPGAs zu erzeugen. Simulink unterstützt ebenfalls alle Integer-, Gleit- und Festkommatypen (engl. float and fixed point) in der Simulation und Codegenerierung.

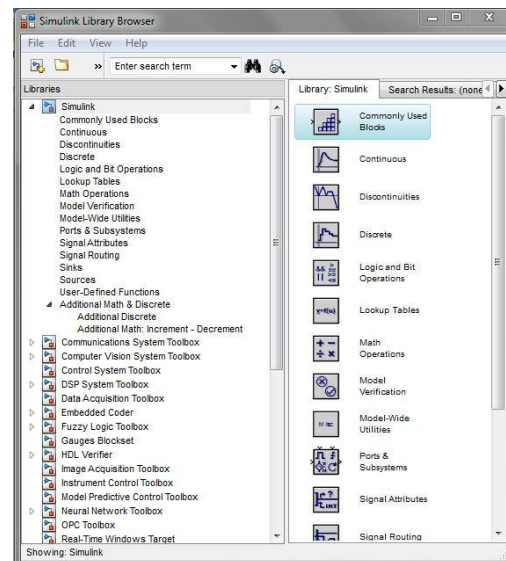


Abbildung 6 Simulink, Library Browser

Bei der Entwicklungsumgebung hat man sich für Microsoft Visual Studio 2010 Professional und Visual Studio 2005 Professional entschieden. Laut der Information auf der Homepage von Mathworks (Abb. 7) für die Erstellung des C++ Codes vom Matlab Coder wird Microsoft Visual Studio 2010 Professional benötigt.

Simulink Product Family – Release 2014a						
	Simulink	Simulink	Stateflow	Simulink Coder	Embedded Coder	Simulink Real-Time
Compiler	For S-Function compilation	For Model Referencing, Accelerator mode, Rapid Accelerator mode, and MATLAB Function blocks	For all features	For all features	When targeting the host OS	For all features
Microsoft Windows SDK 7.1 Available at no charge; requires .NET Framework 4.0	✓	✓	✓	✓ 7	✓ 7	✓
Microsoft Visual C++ 2013 Professional	✓	✓	✓	✓	✓	
Microsoft Visual C++ 2012 Professional	✓	✓	✓	✓	✓	✓
Microsoft Visual C++ 2010 Professional SP1	✓	✓	✓	✓	✓	✓

Abbildung 7 Simulink, unterstützte Compiler

Das Visual Studio 2005, das gleichzeitig die Entwicklungsumgebung für die PhyCARD Plattform ist, wird vom Mathworks nicht unterstützt.

Um den C++ Code erfolgreich generieren zu können, wird der Matlab Compiler und der Matlab Coder benötigt. Das andere Programm Visual Studio 2005 wird mit der PhyCARD Plattform mitgeliefert und nur in dieser Entwicklungsumgebung können Projekte für die Plattform erstellt und kompiliert werden, da das Plattform Builder der Plattform speziell für das Visual Studio 2005 entwickelt wurde. Die Programmierung und die Navigation sind in beiden Programmen ähnlich (Abb. 8).

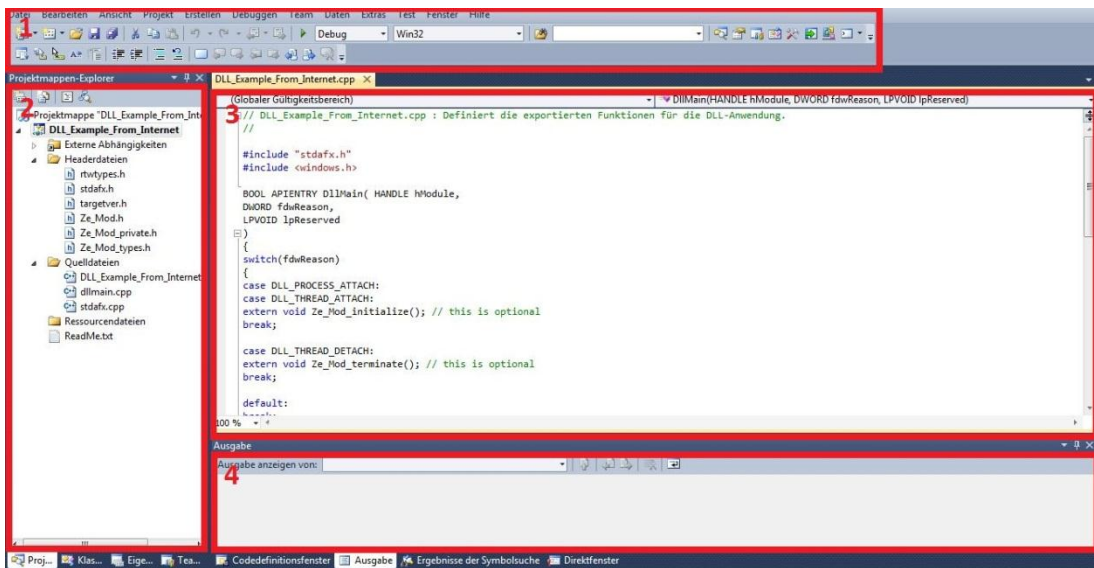


Abbildung 8 Visual Studio 2010, Hauptfenster

Im rot markierten Bereich 1 sind alle relevanten Befehle und Menüs zusammengefasst. Im Bereich zwei kann man zu einem Projekt unterschiedliche Dateien löschen oder hinzufügen. Im dritten Fensterbereich werden die Inhalte gerade geöffneter Datei dargestellt. Unten rechts werden die Informationen zu der Ausgabe sowie Fehlermeldungen angezeigt.

Hier ist eine Übersicht dazu, in welcher Reihenfolge man die Software installieren sollte:

1. Installation vom Windows x86 Professional Betriebssystem
2. Installation vom Microsoft Visual Studio 2005 Professional
3. Installation von zusätzlichen Modulen für die PhyCARD Plattform laut dem Benutzerhandbuch.
4. Installation von Microsoft Visual Studio 2010 Professional
5. Installation von Matlab/ Simulink

6 Konstruktionsphase

Da alle Teile der mechanischen Konstruktion generell aus Aluminium gefräst wurden, hatte man ziemlich viel Freiheit beim Konstruieren. Aufgrund von Erfahrung mit dem Programm Solid Edge, hat man sich für dieses Programm entschieden. Die Konstruktion sollte eine stabile Lage haben – das heißt die ganze Konstruktion sollte sich auf einer stabilen und trotzdem nicht zu schweren Bodenplatte befinden. Als Nächstes wurde ein Gestell entworfen, das aus zwei Laschen bestehen sollte. Für eine leichte Montage sollten sie zukünftig auf die Bodenplatte in spezielle Aussparungen gesteckt werden und dann von unten mit jeweils zwei Zylinderschrauben M6 mit der Bodenplatte verschraubt werden. Einer der Servomotoren befindet sich in einer Lasche, um einerseits Platz zu sparen und andererseits, dass der Drehmittelpunkt des Motors mit der Drehachse der PhyCARD Plattform kollinear sind. In der anderen Lasche befindet sich eine Bohrung für ein Kugellager, die mit der Drehachse des gegenüber liegenden Servomotors kollinear ist. Die Drehachsen der PhyCARD Plattform befinden sich auf der Oberfläche des Touchdisplays und gehen durch dessen Mitte. Unter der Mitte ist nicht die physikalische, sondern die resistive Mitte des Displays zu verstehen (Abb. 9).

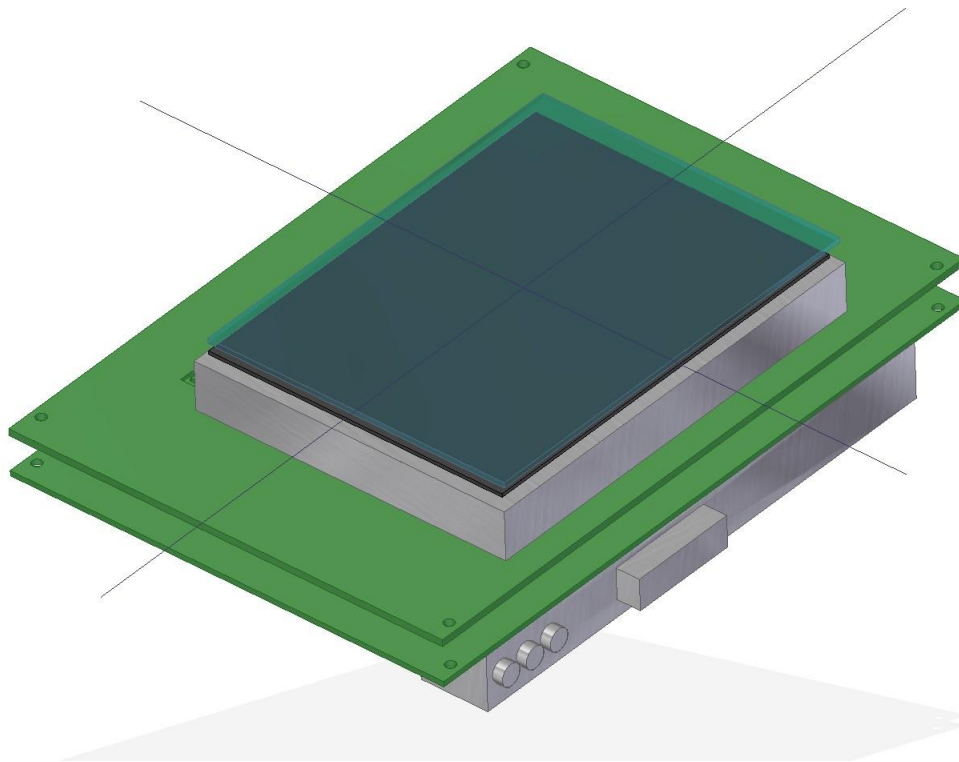


Abbildung 9 PhyCARD Plattform

In Abbildung 9 ist gut zu sehen, dass die PhyCARD aus zwei Platinen besteht, die miteinander fest verschraubt werden sollten. Die Drehachsen sind ebenfalls in der Abbildung gut erkennbar.

Es wurde ein spezieller Zwischenrahmen konstruiert, der kompakt und stabil ist und ein sehr geringes Gewicht hat (Abb. 10). Das geringe Gewicht des Rahmens ist darum wichtig, weil es auf die Trägheitsmasse der drehenden Teile einen direkten Einfluss hat.

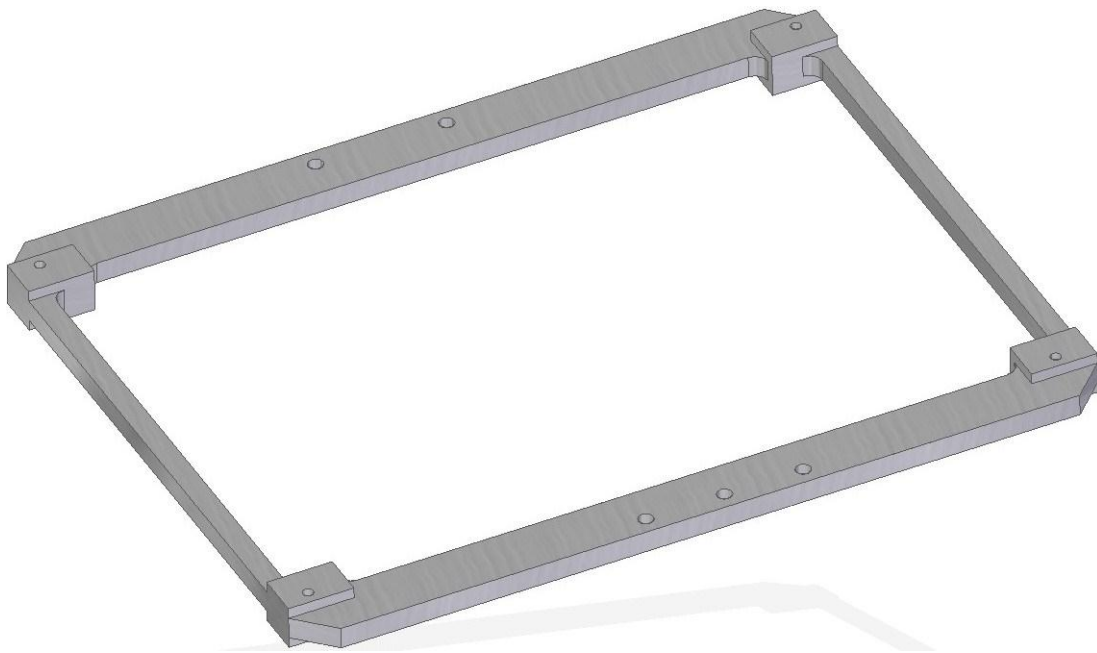


Abbildung 10 Zwischenrahmen für die PhyCARD Platinen

Die beiden Platinen werden oberhalb und unterhalb des Zwischenrahmens mit Schrauben befestigt, um die Elektronik zwischen den Platinen nicht zu gefährden. (Abb. 11). Ein wichtiger Hinweis an der Stelle ist, dass die beiden Drehachsen wegen der Asymmetrie der PhyCARD Plattform nicht auf den Mittellinien der Bodenplatte liegen und die richtige Positionierung der Achsen nur gemessen und berechnet werden kann.

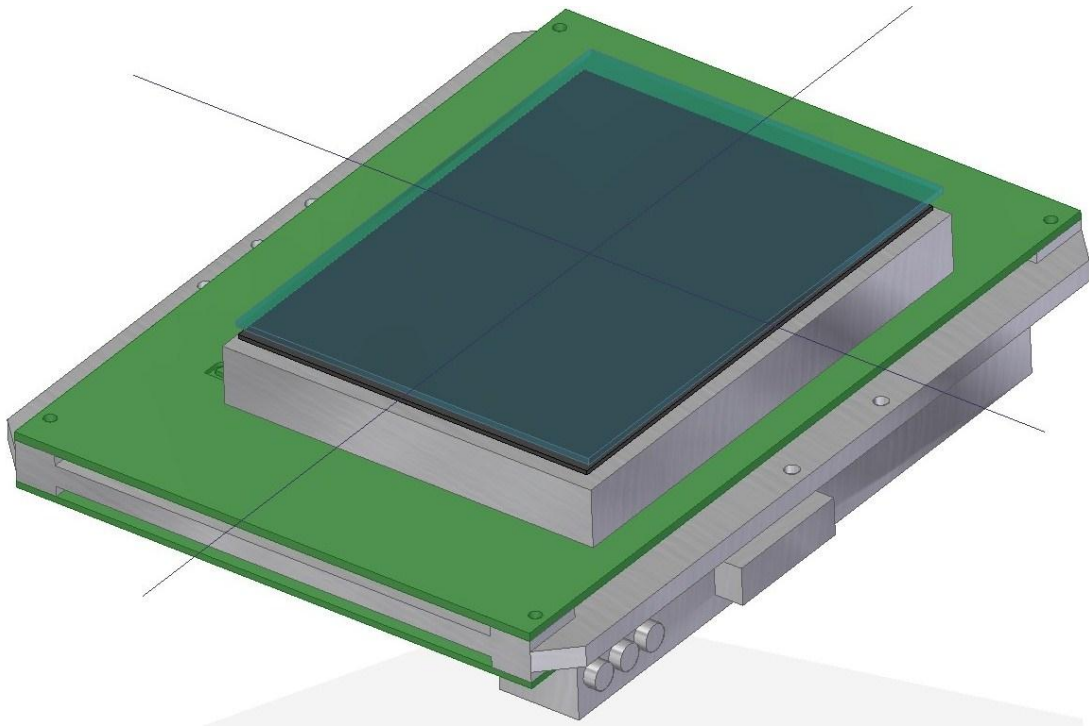


Abbildung 11 PhyCARD Plattform

Zwischen zwei Laschen befindet sich dann ein Rahmen, in dem sich, um 90 Grad gedreht, die andere Drehachse befindet. Auf einer Seite dieses Rahmens ist eine Passbohrung für einen Zylinderstift vorgesehen, dessen anderes Ende sich in dem Kugellager in einer Lasche befindet. Auf der anderen Seite sind spezielle kleine Gewindebohrungen vorhanden, um den Hebel des ersten Servomotors zu verschrauben. Um dann diesen Rahmen mit dem Zwischenrahmen der PhyCARD Plattform zu verbinden, waren zwei Blöcke notwendig. In einem Block wird der zweite Servomotor befestigt und in dem anderen Block das zweite Kugellager. Die beiden Blöcke werden auf dem Zwischenrahmen befestigt und von unten verschraubt. Das Gehäuse für die Konstruktion sollte nach Möglichkeit kompakt, leicht und stabil sein. Auf Wunsch hat man als Material Acrylglas ausgewählt, damit man ein durchsichtiges Gehäuse hat. Das Gehäuse besteht aus 6 Acrylglasplatten: Boden, Deckel und 4 Wände. Es wurde auf die Schraubverbindung verzichtet, um unnötiges Bohren und Beschädigungen an Platten zu vermeiden. Stattdessen sollten alle Platten an den Kanten mit dem speziellen Acrylglas zusammengeklebt werden. An einer Platte wurde aufgrund von Platzmangel ein Ausschnitt für einen der Servomotoren vorgesehen (Abb. 12).

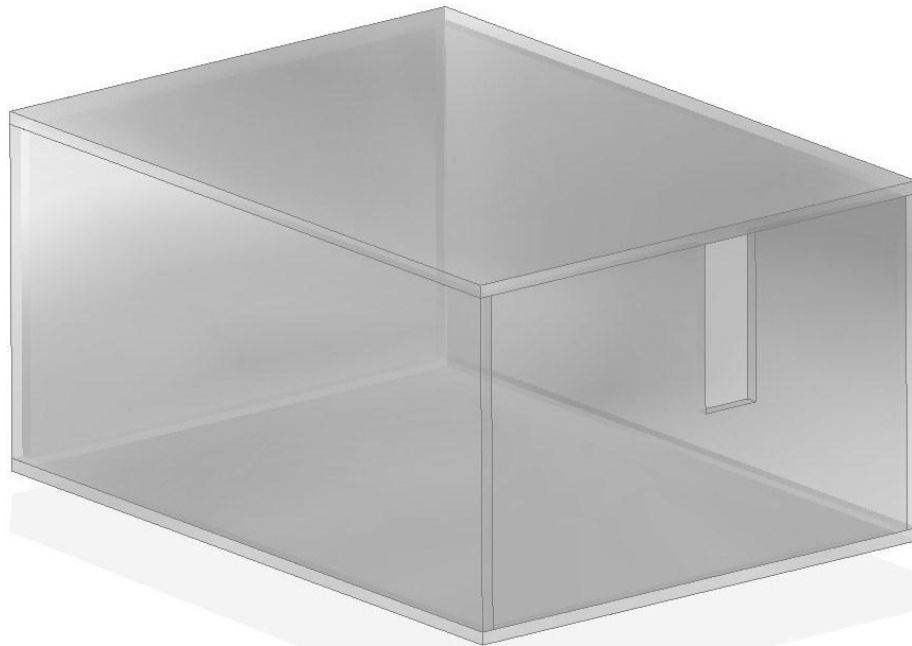


Abbildung 12 Gehäuse aus Acrylglas

Als auch das Gehäuse fertig war, konnte man alle Teile in dem Programm Solid Edge in einer Baugruppe zusammen montieren und prüfen, ob alle Teile passen und Abstände stimmen (Abb. 13).

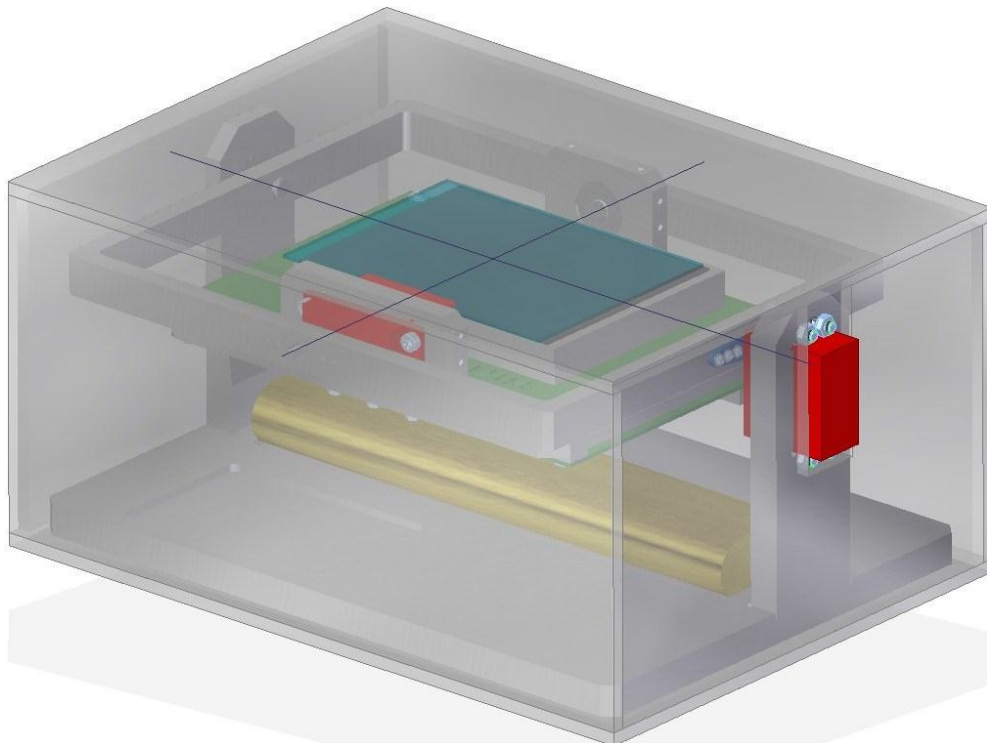


Abbildung 13 Komplette Konstruktion

7 Montage und Test der Konstruktion

In diesem Kapitel werden allgemeine Hinweise für die Nutzung des Textsystems Microsoft Word für das Schreiben von wissenschaftlichen Arbeiten gegeben.

7.1 Montage

Bei der Montage hat es keine Probleme gegeben, da alle Teile passgenau gefertigt wurden. Leichte Zweifel hatte man bei der Fertigung der Stifte, da sie im eingebauten Zustand kein Spiel aufweisen sollten. Bei der Montage zeigte sich, dass die Toleranzen eingehalten wurden. Ein kleines Problem hat es bei dem Zwischenrahmen für die PhyCARD Plattform gegeben. Die Abstände zwischen Bohrungen in den Platinen waren um 0,5mm größer als es gemessen wurde. Da man den gefrästen Rahmen nicht mehr nachbearbeiten konnte, hat man einfach die Bohrungen in den Platinen der PhyCARD Plattform selbst in die nötige Richtung um 0,5mm leicht aufgebohrt, was weder die Funktion beeinträchtigt, noch die Elektronik auf der Platine gefährdet.

7.2 Test

Nachdem die Servomotoren an die PhyCARD angeschlossen wurden, um die Dynamik des Gesamtsystems zu testen, hat sich herausgestellt, dass die digitalen Regler ein ungenügendes Verhalten aufweisen.

Das Spiel im Getriebe der Servomotoren betrug ca. 3 Grad, was eine wesentliche Störung bei den Regeln darstellte. Nachdem die Servomotoren durch das PWM-Signal angeregt wurden und dann in der Endposition stehen blieben, gerieten sie in Schwingungen, die nur per Hand gebremst werden konnten. Dieses Aufschaukeln war durch ein zu großes Spiel der Servomotoren, schnelle Bewegungen und die fehlende Dämpfung der bewegenden Massen bedingt.

8 Verbesserung der Konstruktion

Wie man während der Inbetriebnahme festgestellt hat, waren weitere Optimierungen bei den Servomotoren/ Dämpfer notwendig.

Die erste Lösung war, ein paar Dämpfer für beide Drehachsen zu verbauen, um die Frequenz der ungewollten Schwingungen zu reduzieren. Nach kurzer Analyse der Konstruktion und der Suche möglicher Varianten hat man sich auf der Verwendung von Federn geeinigt. Die Federn sollten sich zwischen der Bodenplatte und dem Rahmen und dem Zwischenrahmen befinden. Da die PhyCARD Plattform sich in vier Richtungen drehen lässt, waren vier entsprechende Federn notwendig.

Die zweite Lösung des Problems war, das Verzichten auf die vorhandenen Servomotoren, eine weitere Suche nach den Servomotoren, die weniger Spiel aufweisen, damit das Aufschaukeln nicht stattfinden kann.

Nach Evaluierung aller Vor- und Nachteile, hat man sich für die zweite Lösung entschieden, nämlich die Suche nach besseren Servomotoren. Es war nicht einfach im Internet Servomotoren zu finden, die besser vom Spiel her sein sollten, da kein Hersteller es direkt in den technischen Daten seiner Produkte angibt. Die Aushilfe war das Durchsuchen unzähliger Internetseiten, in denen über die Servomotoren diskutiert wird. Am Ende gab es zwei Favoriten, nämlich ein analoger Servomotor S9206 der Firma Futaba und ein digitaler Servomotor BLS452 ebenfalls der Firma Futaba. Bei dem Test beider Servomotoren hat man festgestellt, dass der digitale Servomotor fast absolut spielfrei war, jedoch ebenfalls in Schwingung geraten konnte und somit nicht geeignet war. Der Analoge hatte etwas Spiel, jedoch war dieses Spiel wesentlich geringer, als bei den ersten Servomotoren. Wichtiger jedoch war, dass dieser analoge Servomotor keine Schwingung aufwies und dieses Problem somit beseitigt war. Die digitalen Servomotoren hatten einen digitalen Regler verbaut, der selbst bei den kleinen Auslenkungen mit voller Kraft nachregelte. Die analogen Servomotoren hatten einen analogen Regler drin und konnten somit nicht die maximale Leistung bei den kurzen Bewegungen erreichen und hatten wesentlich sanfteres Verhalten als die digitalen Servomotoren. An diesem Beispiel konnte man deutlich sehen, wie wichtig das Auswählen der richtigen Komponente in Bezug auf die Stabilität und das Verhalten des Gesamtsystems ist.

9 Entwicklungsphase

Hier muss erwähnt werden, dass die DLL-Dateien, die im Matlab und Simulink direkt erstellt werden, können nur auf einem Windows Rechner und nicht auf einem eingebetteten System laufen. Damit man die DLL auf einem eingebetteten System verwenden kann, muss zuerst der C++ Code im Matlab oder Simulink generiert werden und erst dann kann die DLL aus dem C++ Code im Visual Studio 2005 mit dem passenden Plattform Builder für die PhyCARD erstellt werden.

9.1 Erstellung und Einbindung der DLL für Desktop Systeme

9.1.1 Erstellung der DLL im Matlab

Als Erstes muss das Programm Matlab eingestellt werden. Wenn Matlab gestartet ist, gibt man in der Kommandozeile das Befehl „mex -setup“ ein, um einen passenden Compiler auszuwählen. An der Stelle sollte erwähnt werden, dass der originale Compiler vom Matlab nur den C Code unterstützt und da man nicht den C Code, sondern C++ Code zur Codegenerierung braucht, ist man auf die Installation von Microsoft Visual Studio 2010 und seinen Compiler im Matlab angewiesen. Nachdem der Befehl eingegeben wurde, wird Matlab zuerst fragen, ob es nach allen bekannten und installierten Compilern auf dem Rechner gesucht werden sollte. Wenn Sie „Y“ drücken, wird eine Liste mit allen verfügbaren Compilern angezeigt. An der Stelle sollte der Compiler vom Microsoft Visual Studio 2010 ausgewählt werden, dann drückt man anschließend „Enter“ und bestätigt die Auswahl mit „Y“. Nun wird der Compiler bei dem Erstellen von Simulink Modellen immer den richtigen Compiler hernehmen, um später aus Matlab Code und Simulink-Modellen nicht den C, sondern C++ Code erstellen zu können. Die Generierung der DLL wird in dem folgenden Beispiel gezeigt. Man erstellt eine Script-Datei namens AddNumbers.m mit folgendem Code.

```
function c = AddNumbers(a,b) %#codegen
c=a+b;
end
```

Der Name der Funktion in der Script-Datei ist wichtig, da später über diesen Namen in der DLL die Funktion aufgerufen wird. Um dann den Matlab Coder zu starten, gibt man in der Kommandozeile den Befehl „coder“ ein. Der Name des Projekts kann frei gewählt werden. In dem Reiter „Overview“ unter „Entry Point“ müssen die Eingangsvari-

ablen definiert werden. In diesem Fall wurden sie mit dem Typ „double, skalar“ deklariert (Abb. 14).

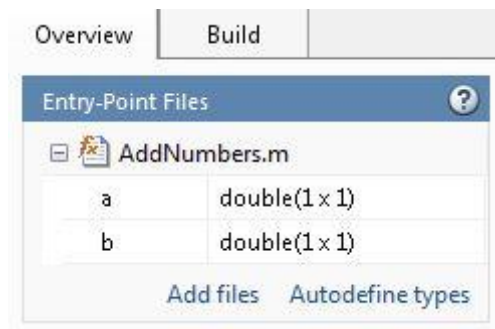


Abbildung 14 Matlab, Definition der Eingangsvariablen

In dem Reiter „Build“ gibt man den Namen der DLL und wählt den Type „C/C++ Dynamic Library“ aus. Der Haken gegenüber der Option „Generate code only“ muss ab sein. Nun klickt man auf den Button „Build“ um die DLL zu erstellen.

Nach der Generierung des Codes ist die DLL im Matlab-Ordner unter „\codegen\dll\Name des Projekts“ zu finden.

Mit dem Programm Dependency Walker kann die DLL geöffnet werden, um die DLL auf die Existenz der externen Funktionen zu prüfen. Wie man in der Abbildung 15 sieht, haben die externen Funktionen AddNumbers, AddNumbers_initialize und AddNumbers_Terminate ein Zeichen „_“ jeweils vor dem Namen bekommen (Abb. 15). Bei der Einbindung der DLL muss auf dieses Zeichen geachtet werden.

E	Ordinal ^	Hint	Function	Entry Point
<input checked="" type="checkbox"/>	1 (0x0001)	0 (0x0000)	_AddNumbers	0x00001354
<input checked="" type="checkbox"/>	2 (0x0002)	1 (0x0001)	_AddNumbers_initialize	0x00001367
<input checked="" type="checkbox"/>	3 (0x0003)	2 (0x0002)	_AddNumbers_terminate	0x00001372

Abbildung 15 Dependency Walker, Auflistung der externen Funktionen

9.1.2 Erstellung der DLL im Simulink

Erstellen eines Reglers im Simulink (Output: DLL)

Um in das Programm Simulink zu gelangen, gibt man in der Kommandozeile vom Matlab den Befehl „simulink“ und drückt „Enter“. Nach dem die Bibliothek mit unterschiedlichen Blöcken geladen ist, erstellt man ein neues Modell durch die Tastenkombination „Strg+N“. Als Beispiel wird im Simulink ein Modell mit dem Namen „Example“ erstellt. Das Simulink-Modell (Abb. 16) soll zwei Eingangsvariable zusammen addieren und multiplizieren können.

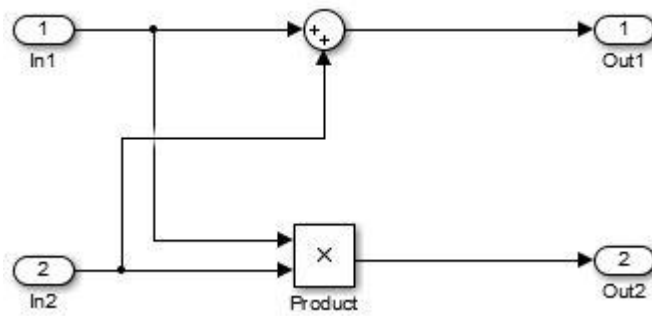


Abbildung 16 Simulinkmodell mit der Addition und Multiplikation

Alle Typen bei den Variablen und Blöcken müssen auf „double“ umgestellt werden. Nach dem Entwurf des Reglers sollten noch dessen Einstellungen so geändert werden, dass man daraus eine DLL erstellen könnte. Dies erreicht man, indem man die Tastenkombination „Strg+E“ betätigt. Im neuen Fenster „Configuration Parameters“ geht man auf Tab „Solver“ und ändert die Option „Type“ von „Variable-step“ auf „Fixed-step“ und die Option „Solver“ auf „discrete (no continuous states)“. Danach sollte man im Tab „Code Generation“ als „System target file“ das „ert_shrlib.tlc“ auswählen. Wenn man gleiche Einstellungen wie in der Abbildung 20 hat, kann die DLL generiert werden, indem man den Button „Build“ betätigt (Abb. 17).

Target selection

System target file: ert_shrlib.tlc Browse...

Language: C

Description: Embedded Coder (host-based shared library target)

Build process

Toolchain settings

Toolchain: Automatically locate an installed toolchain Validate...

Microsoft Visual C++ 2010 v10.0 | nmake (32-bit Windows)

Build configuration: Faster Builds Show settings

Minimize compilation and linking time

Data specification override

Ignore custom storage classes Ignore test point signals

Code Generation Advisor

Prioritized objectives: Unspecified Set objectives ...

Check model before generating code: Off Check model ...

Generate code only Build

Package code and artifacts Zip file name:

Abbildung 17 Simulink, empfohlene Einstellungen

Die DLL ist nach der Codegenerierung im Matlab-Ordner unter dem Namen „Name des Simulink-Modells_win32.dll“ zu finden. Nun wird wieder das Programm „Dependency Walker“ zum Öffnen der DLL benötigt.

E	Ordinal ^	Hint	Function	Entry Point
<input checked="" type="checkbox"/>	1 (0x0001)	0 (0x0000)	Example_M	0x00009110
<input checked="" type="checkbox"/>	2 (0x0002)	1 (0x0001)	Example_M_	0x0000DAE0
<input checked="" type="checkbox"/>	3 (0x0003)	2 (0x0002)	Example_U	0x0000DAC0
<input checked="" type="checkbox"/>	4 (0x0004)	3 (0x0003)	Example_Y	0x0000DAD0
<input checked="" type="checkbox"/>	5 (0x0005)	4 (0x0004)	Example_initialize	0x00001030
<input checked="" type="checkbox"/>	6 (0x0006)	5 (0x0005)	Example_step	0x00001000
<input checked="" type="checkbox"/>	7 (0x0007)	6 (0x0006)	Example_terminate	0x00001070

Abbildung 18 Dependency Walker, Externe Funktionen der DLL

Die externen Funktionen Example_initialize, Example_step und Example_terminate sind für die Initialisierung, den Aufruf und die Beendung des Reglers zuständig. Über die Funktion Example_U und Example_Y erfolgt der Datenaustausch zwischen dem Hauptprogramm und dem Regler (Abb. 18).

9.1.3 Einbindung der im Matlab erstellten DLL im Visual Studio 2005

Man legt ein neues Projekt, z.B. mit dem Namen DLL_TEST im Visual Studio an. Als „Projekt types“ und „Templates“ wählt man entsprechend „Win32“ und „Win32 Projekt“ und anschließend klickt man auf den Button „Finish“. Nachdem das Projekt erstellt wurde, sollte die DLL in den Ordner mit dem Projekt kopiert werden. Damit man nun die DLL einbinden kann, sollte der untenstehende Text an passender Stelle (Abb. 19) im DLL_TEST.cpp eingefügt werden.

```
//+++++
typedef double (*BinaryFunction_t) (double, double) ;
BinaryFunction_t AddNumbers ;
double result ;
BOOL fFreeResult ;

// DLL Datei laden
HINSTANCE hinstLib = LoadLibrary (TEXT("AddNumbers.dll")) ;

if (hinstLib != NULL)
{
// Die Einsprungsadresse abfragen
AddNumbers = (BinaryFunction_t) GetProcAddress (hinstLib,
```

```

        "_AddNumbers") ;

// Die Funktion aufrufen
    if ( AddNumbers != NULL )
        result = (*AddNumbers) (1, 2) ;

// Die DLL-Datei wieder entladen
    FreeResult = FreeLibrary (hinstLib) ;
}

//+++++

```

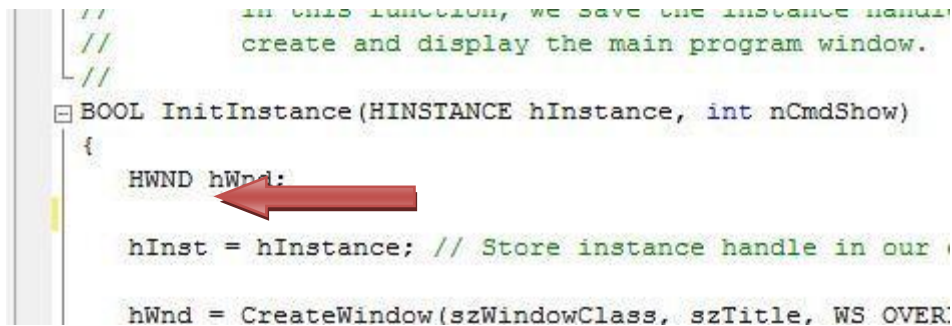


Abbildung 19 VS 2005, Platz zum Einfügen des Codes

An die Funktion „AddNumbers“ werden zwei Werte 1 und 2 übergeben, die dann zusammen addiert und in der Variable „result“ gespeichert werden.

9.1.4 Einbindung der im Simulink erstellten DLL im Visual Studio 2005

Die Einbindung der DLL, die mit dem System target file „ert_shrllib.tlc“ im Simulink erstellt wurde, ist identisch mit der Einbindung der DLL im Kapitel 9.2.3.

Die Einbindung der DLL, die im Kapitel 9.1.2 beschrieben wurde, ist mit der Einbindung der DLL im Kapitel 9.2.3 identisch.

9.2 Erstellung und Einbindung der DLL für eingebettete Systeme

9.2.1 Erstellung von C++ Code im Simulink

Erstellen eines Reglers im Simulink (Output: C++ und Headerdatei)

Das Erstellen des C++ Codes ist dem Beispiel aus dem Kapitel 9.1.2 sehr ähnlich. Man wählt nun aber im Tab „Code Generation“ als „System target file“ das „ert.tlc“. Nachdem die Option „ert.tlc“ gewählt wurde, erscheint unten die Möglichkeit, die Ausgangssprache zu wählen. Um später die DLL-Datei fehlerfrei kompilieren zu können, wird es empfohlen, die Sprache „C++“ zu wählen. Im Weiteren wird das Simulink-Modell mit dem Namen EC.slx betrachtet (Abb. 20).

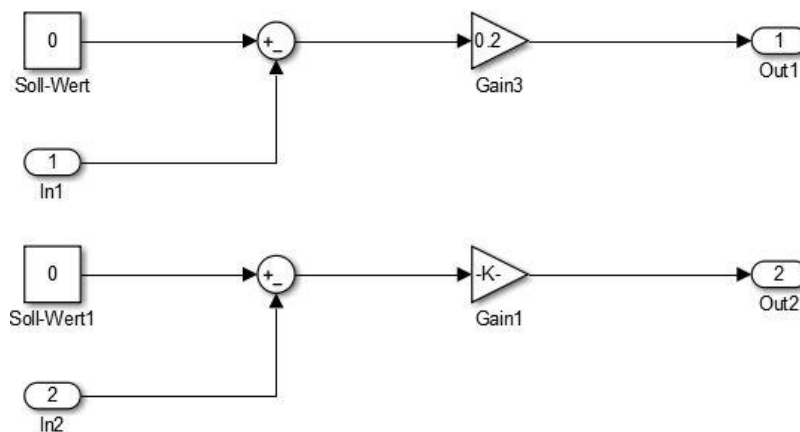


Abbildung 20 Simulink-Modell, P-Regler

Wenn man soweit ist, wird der C++ Code mit sämtlichen Dateien im „Matlab-Ordner\EC_ert_rtw\“ generiert, indem man auf den Button „Build“ klickt. Für die Einbindung der DLL im Kapitel 9.2.2 werden folgende Dateien benötigt:

- EC.cpp
- EC.h
- EC_data.cpp
- EC_private.h
- EC_types.h
- rtwtypes.h

9.2.2 Erstellung der DLL aus dem C++ Code im Visual Studio 2005

Man erstellt ein neues Projekt wie folgt:

New Project->Smart Device->Win32 Smart Device Project->Name des Projekts.

Hier wurde der Name für das Projekt „CREATE_DLL_FROM_EC_FOR_ARM“ angenommen.

Bei der Auswahl von SDK wählt man „Phytec i.MX35 SDK“ aus. Im nächsten Fenster aktiviert man die Option „DLL“ und klickt auf „Finish“.

Anschließend kopiert man die Dateien, die im Kapitel 9.2.1 erstellt wurden, in den Projektordner „Visual Studio 2005\Projects\CREATE_DLL_FROM_EC_FOR_ARM\CREATE_DLL_FROM_EC_FOR_ARM“. Danach müssen alle Dateien im Visual Studio mit dem Projekt verbunden werden (Abb. 21).

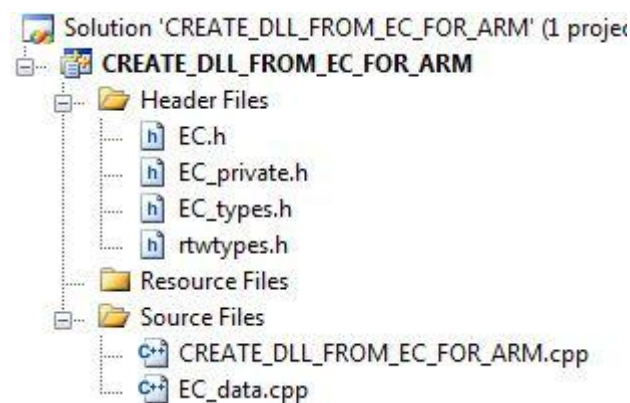


Abbildung 21 VS 2005, Projektordner

Den C++ Code aus der Datei EC.cpp kopiert man komplett in die Datei CREATE_DLL_FROM_EC_FOR_ARM.cpp.

Nun müssen die Externe Funktionen in der CREATE_DLL_FROM_EC_FOR_ARM.cpp Datei umgeschrieben werden.

Nun sind folgende Änderungen des Textes notwendig:

```
extern "C" __declspec(dllexport) void EC_step()
//void EC_step(void)

extern "C" __declspec(dllexport) void EC_initialize()
//void EC_initialize(void)

extern "C" __declspec(dllexport) void EC_terminate()
//void EC_terminate(void)
```

Ebenfalls in der Datei EC.h müssen Änderungen vorgenommen werden:

```
extern "C" __declspec(dllexport) Parameters_EC EC_P;
//extern Parameters_EC EC_P;
```

```
extern "C" __declspec(dllexport) ExternalInputs_EC EC_U;
//extern ExternalInputs_EC EC_U;

extern "C" __declspec(dllexport) ExternalOutputs_EC EC_Y;
//extern ExternalOutputs_EC EC_Y;
//extern void EC_initialize(void);
//extern void EC_step(void);
//extern void EC_terminate(void);

extern "C" __declspec(dllexport) RT_MODEL_EC *const EC_M;
//extern RT_MODEL_EC *const EC_M;
```

Nach der Compilierung des Projekts wird eine DLL namens CREATE_DLL_FROM_EC_FOR_ARM.dll in „\Visual Studio 2005\Projects\CREATE_DLL_FROM_EC_FOR_ARM\CREATE_DLL_FROM_EC_FOR_ARM\Phytec i.MX35 SDK (ARMV4I)\Debug“ erstellt.

9.2.3 Einbindung der DLL im Visual Studio 2005

Für die Einbindung wird die DLL aus dem Kapitel 9.2.3 verwendet.

Man erstellt wieder ein neues Projekt wie folgt:

New Project->Smart Device->Win32 Smart Device Project->Name des Projekts.

Nun wird das Projekt „USE_DLL_FROM_EC_FOR_ARM“ genannt.

Bei der Auswahl von SDK wählt man wieder „Phytec i.MX35 SDK“ aus und das Projekt kann erstellt werden.

Die DLL wird in den Ordner „Visual Studio 2005\Projects\USE_DLL_FROM_EC_FOR_ARM\USE_DLL_FROM_EC_FOR_ARM“ kopiert.

Man geht auf „Project->Properties->Deployment->General->Additional Files“ und gibt Folgendes ein: CREATE_DLL_FROM_EC_FOR_ARM.dll|C:\Users\Z8G1\Documents\Visual Studio 2005\Projects\USE_DLL_FROM_EC_FOR_ARM\USE_DLL_FROM_EC_FOR_ARM|%CSIDL_PROGRAM_FILES%\USE_DLL_FROM_EC_FOR_ARM|0“

Danach geht man auf „Project->Properties->C/C++/General->Additional Include Directories“ und gibt Folgendes ein: C:\Users\Z8G1\Documents\Visual Studio 2005\Projects\USE_DLL_FROM_EC_FOR_ARM\USE_DLL_FROM_EC_FOR_ARM

Auch beim Linker/General muss im „Additional Library Directories“ folgender Text eingefügt werden: C:\Users\Z8G1\Documents\Visual Studio 2005\Projects\USE_DLL_FROM_EC_FOR_ARM\USE_DLL_FROM_EC_FOR_ARM

Zum Einbinden der DLL fügt man nach dem Programmcode

```
CommandBar_Show(g_hWndCommandBar, TRUE);
```

```
}

```

im USE_DLL_FROM_EC_FOR_ARM.cpp folgenden Text ein:

```
DWORD WINAPI GetLastError(void);
HINSTANCE handleLib;
handleLib = LoadLibrary(L"CREATE_DLL_FROM_EC_FOR_ARM.dll");
void (*mdl_initialize)(BOOL);
void (*mdl_step)(void);
void (*mdl_terminate)(void);
ExternalInputs_Example (*mdl_Uptr);
ExternalOutputs_Example (*mdl_Yptr);
mdl_initialize = (void(*) (BOOL)) GetProcAddress(handleLib
,L"EC_initialize");
mdl_step = (void(*) (void)) GetProcAddress(handleLib
,L"EC_step");
mdl_terminate = (void(*) (void)) GetProcAddress(handleLib
,L"EC_terminate");
mdl_Uptr = (ExternalInputs_Example*) GetProcAddress(handleLib
,L"EC_U");
mdl_Yptr = (ExternalOutputs_Example*) GetProcAddress(handleLib
,L"EC_Y");
if ((mdl_initialize && mdl_step && mdl_terminate && mdl_Uptr &&
mdl_Yptr)) {
    /* === user application initialization function === */
    mdl_initialize(1);
    /* insert other user defined application initialization code
here */
    /* === user application step function === */
    mdl_Uptr->In1 = 0.1;
    mdl_Uptr->In2 = -0.25;
    mdl_step();
    double a1=0, a2=0;
    a1=(*mdl_Yptr).Out1;
    a2=mdl_Yptr->Out2;
    /* === user application terminate function === */
    mdl_terminate();
    /* insert other user defined application termination code here */
}
else {
    return(-1);
}
return(FreeLibrary(handleLib));

```

Die In1 und In2 sind Eingangsvariablen des Reglers und die a1 und a2 Ausgangsvariablen.

Nach diesem Schritt wird auch die USE_DLL_FROM_EC_FOR_ARM.h Datei wie folgt ergänzt:

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

#ifndef _APP_MAIN_HEADER_
#define _APP_MAIN_HEADER_

typedef struct {
    double In1;
    double In2;
} ExternalInputs_Example;

typedef struct {
    double Out1;
    double Out2;
} ExternalOutputs_Example;

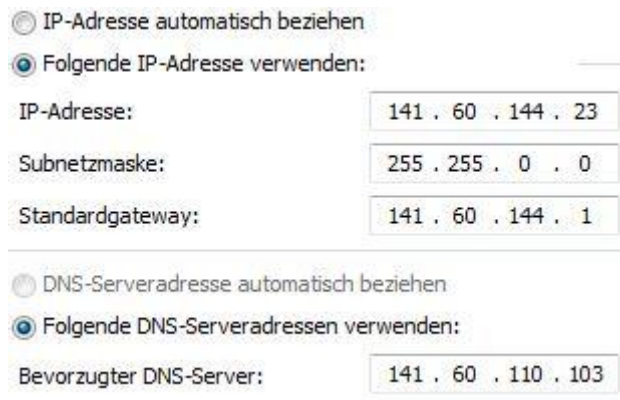
#endif
```

Wenn alle Schritte durchgeführt wurden, wird das Projekt auf der PhyCARD Plattform fehlerfrei kompiliert.

10 Test der Schnittstelle

10.1 Benutzung der SQL Datenbank

Ein wichtiger Bestandteil dieser Arbeit ist das Vergleichen der Messdaten, die auf der PhyCARD Plattform aufgenommen wurden, mit den Ergebnissen vom ausgelegten Regler im Simulink. Um die Messwerte von der PhyCARD bearbeiten zu können, sollten sie vorher von dem SQL Server heruntergeladen werden. Um mit dem Server Verbindung aufbauen zu können, sollen dem Client Rechner im Windows folgende IP-Einstellungen zugewiesen werden (Abb. 22).



The image shows a screenshot of the Windows network configuration interface. It is divided into two main sections: IP settings and DNS settings. In the IP settings section, the radio button 'Folgende IP-Adresse verwenden:' is selected. Below it, three text boxes are filled with the IP address '141 . 60 . 144 . 23', the subnet mask '255 . 255 . 0 . 0', and the standard gateway '141 . 60 . 144 . 1'. In the DNS settings section, the radio button 'Folgende DNS-Serveradressen verwenden:' is selected, and the text box for 'Bevorzugter DNS-Server:' is filled with '141 . 60 . 110 . 103'.

Abbildung 22 IP-Einstellungen

Für die erfolgreiche Benutzung der SQL Datenbank von der Client Maschine sollte die DLL „libmysql.dll“ (befindet sich auf der CD unter \TABLEDANCEagent\) im Windows unter „C:\Windows\System32“ abgelegt werden. Wenn das x64 Betriebssystem benutzt wird, ändert sich der Speicherort der DLL zu „C:\Windows\System64“.

Nachdem diese Schritte erledigt wurden, gibt man die IP-Adresse des Servers im Internetbrowser wie folgt ein:

IP-Adresse des Servers: 141.60.140.43

Nachdem die Seite des Servers erschienen ist, klickt man in der linken Leiste auf den Button „phpMyAdmin“. Auf der nächsten Seite klickt man wieder in der linken Leiste auf den Button „tabledance“. Dadurch gelangt man zu dem Hauptfenster der Datenbank der PhyCARD Plattform (Abb. 23).

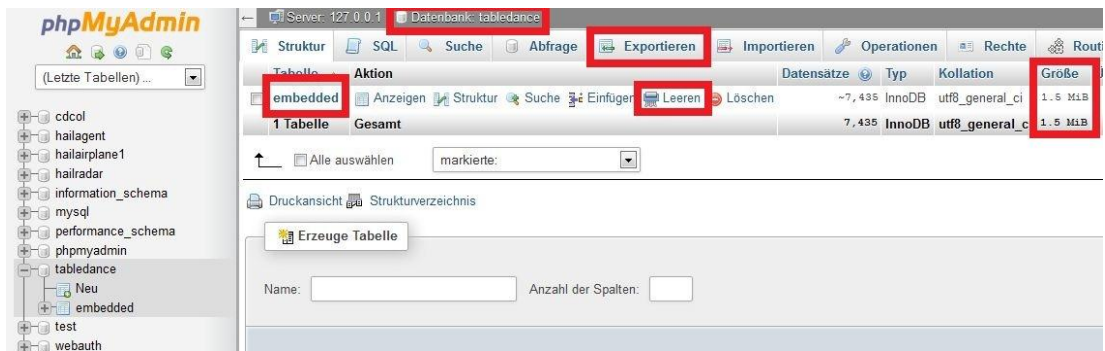


Abbildung 23 PhyCARD Datenbank

Wenn die Messdaten vorher schon aufgenommen wurden, klickt man oben auf den Button „EXPORTIEREN“ und wählt anschließend, vor dem Speichern, das Format der Datei „CSV for MS Excel“. In der .CSV bzw. .XLS Datei müssen alle Punkte durch Kommas ersetzt werden, ansonsten ist die Auswertung der Messdaten nicht möglich.

Um die Messung neu zu starten, muss die „embedded“ Tabelle vorher geleert werden. Zum Leeren der Tabelle klickt man auf den Button „leeren“. Das versehentliche Löschen der Tabelle sollte vermieden werden, da die Tabelle dann wieder neu erstellt werden muss.

Eine Messung startet erst dann, wenn das Client Programm „TABLEDANCEagent“ (befindet sich auf der CD im Ordner „\TABLEDANCEagent“) auf dem Client Rechner mit der IP-Adresse 141.60.144.23 gestartet wird. Nachdem der Messvorgang beendet ist, wird die Aufnahme der Messdaten mit dem Schließen des Programms „TABLEDANCEagent“ gestoppt. Da die Namen der Spalten beim Speichern immer gelöscht werden, sind sie der „embedded“ Tabelle auf dem Server zu entnehmen.

10.2 Vergleich von PhyCARD und Simulink Ergebnissen

Es wurde eine Messung durchgeführt, bei der mit dem Finger ein provisorischer Kreis auf dem Touchdisplay gezeichnet wurde. Die Tabelle „embedded“ wurde als tabledance.csv Datei gespeichert und in eine Microsoft Excel-Datei konvertiert. Da nur fünf Spalten von Interesse sind, nämlich Zeit, Eingänge und Ausgänge für die X- und Y-Achse, wurden die restlichen Spalten gelöscht. Die Abtastzeit beträgt 10 ms, d.h. 100 Einträge in der Tabelle wurden innerhalb von 1 Sekunde aufgenommen. Um die Ergebnisse maximal überschaubarer zu machen, wurden die Einträge vor und nach dem Messvorgang, die keine nützliche Information enthalten, gelöscht. Die Werte auf den unten folgenden Diagrammen entsprechen den Achsen wie folgt:

S02_PointX - Position der Kugel (in dem Fall des Fingers) auf der X-Achse der PhyCARD Plattform. Da sind aber auch gleichzeitig die Eingangswerte der X-Achse für den Regler in der DLL.

S02_PointY - Position der Kugel (in dem Fall des Fingers) auf der Y-Achse der PhyCARD Plattform. Da sind aber auch gleichzeitig die Eingangswerte der Y-Achse für den Regler in der DLL.

S03_AvarX – die Ausgangswerte vom Regler in der DLL für die X-Achse, die gleichzeitig Eingangswerte für den Servomotor auf der X-Achse sind.

S03_AvarY - die Ausgangswerte vom Regler in der DLL für die Y-Achse, die gleichzeitig Eingangswerte für den Servomotor auf der Y-Achse sind.

TIME Anzahl der Schritte, pro Messvorgang. Jeder Messvorgang dauert 10 ms. Um auf die Zeit zu kommen, multipliziert man die Werte in dieser Spalte mit dem Faktor 10 und bekommt die Zeit in Millisekunden.

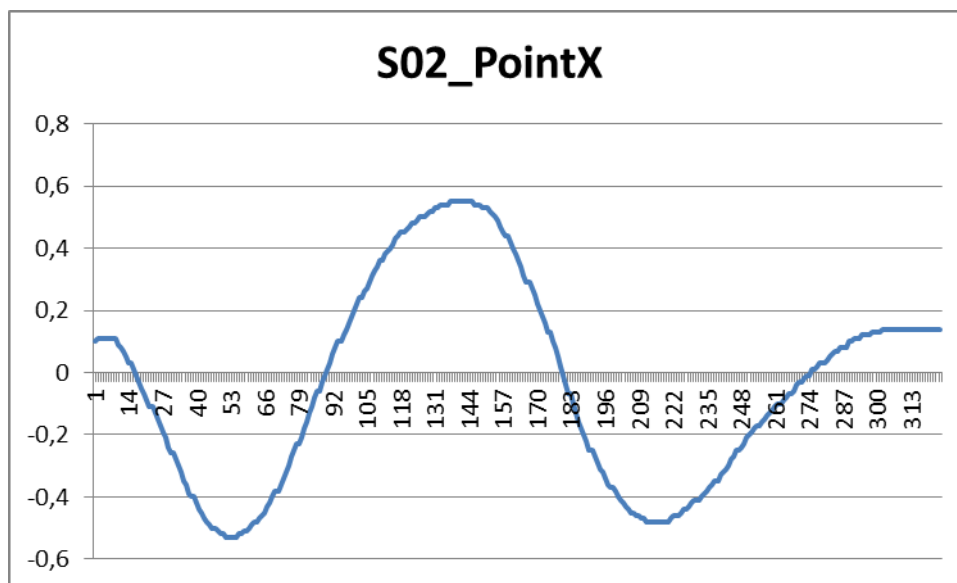


Abbildung 24 Eingangswerte für die X-Achse des Reglers

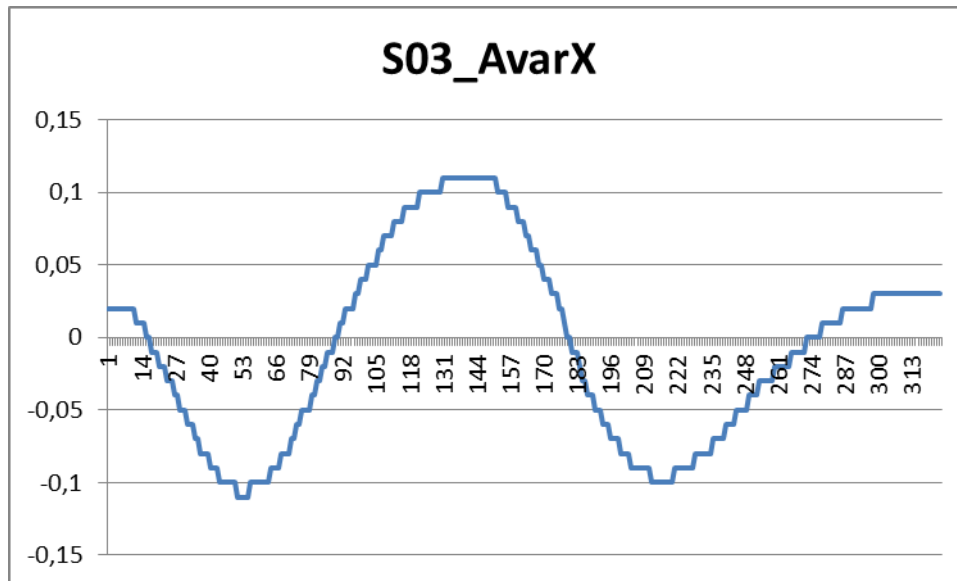


Abbildung 25 Ausgangswerte des Reglers für die X-Achse

Wie man in den Abbildungen 24 und 25 sieht, ist die Amplitude der Ausgangswerte des Reglers um den Faktor 5 geringer als die Amplitude der Eingangswerte. Laut der Zeitskala dauerte der Messvorgang ca. 3 Sekunden, was der Tatsache entspricht. Sowohl die Eingangswerte als auch die Ausgangswerte des Reglers liegen im Bereich zwischen -1 bis 1 mit der Auflösung 0,01. Dies entspricht 200 Werten bei dem Bildschirm und dem Servomotor. Da es sich bei diesem Regler um einen einfachen P-Regler mit dem Verstärkungsfaktor 0,2 handelte, litt darunter die Auflösung der Ausgangsvariablen. In dem Fall betrug die Auflösung der Ausgangswerte des Reglers bei 200 Werten * 0,2 Verstärkungsfaktor = 40 Werte. Andererseits konnten, durch diese geringere Skalierung, die Servomotoren im kleineren Bereich geregelt werden. Die Skalierung der Ausgangsgrößen ist an den unterschiedlichen Ausgangs- und Eingangswerten des Reglers erkennbar.

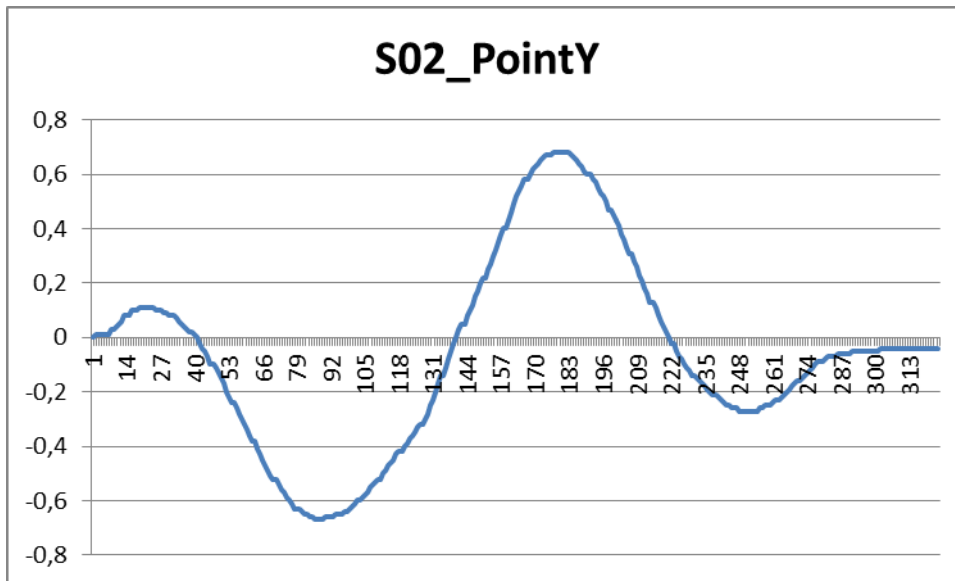


Abbildung 26 Eingangswerte für die Y-Achse des Reglers

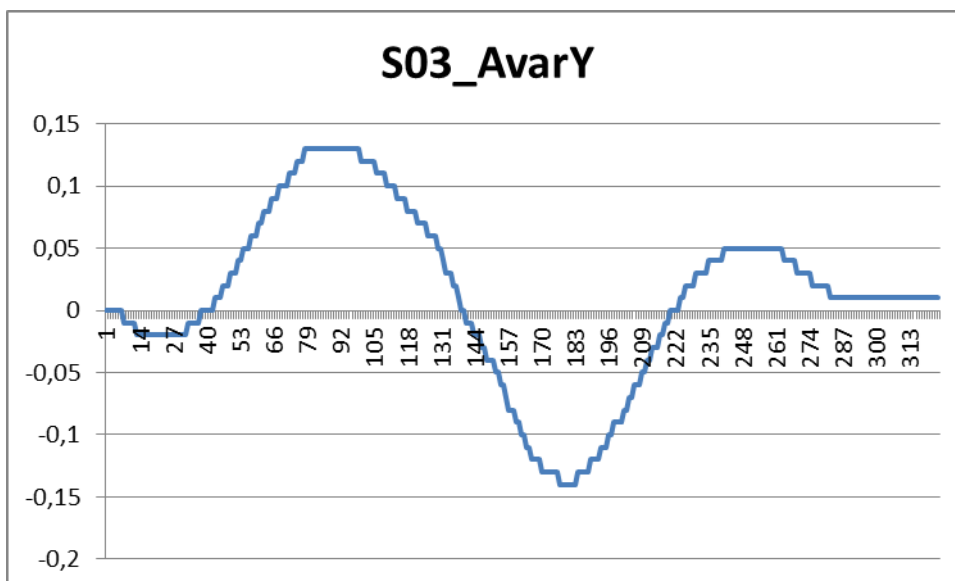


Abbildung 27 Ausgangswerte des Reglers für die Y-Achse

Bei der Y-Achse sind die Ausgangsgrößen umgekehrt proportional zu den Eingangsgrößen (Abb. 26 und Abb. 27). Dies wird durch die spezifische Bauweise der Konstruktion bedingt. Wenn man einen der beiden Servomotoren andersrum montiert hätte, dann hätten die Ein- und Ausgangsgrößen für die beiden Achsen das gleiche Vorzeichen. Ansonsten weisen die Ein- und Ausgangsgrößen der Y-Achse das gleiche Verhalten auf wie die Ein- und Ausgangsgrößen der X-Achse.

Beim nächsten Schritt wurden die Ergebnisse von dem Regler mit den Ergebnissen vom Simulink Modell verglichen. Dafür wurden im Simulink als Eingangsgrößen dieselben Eingangswerte von der X- und Y-Achse genommen.

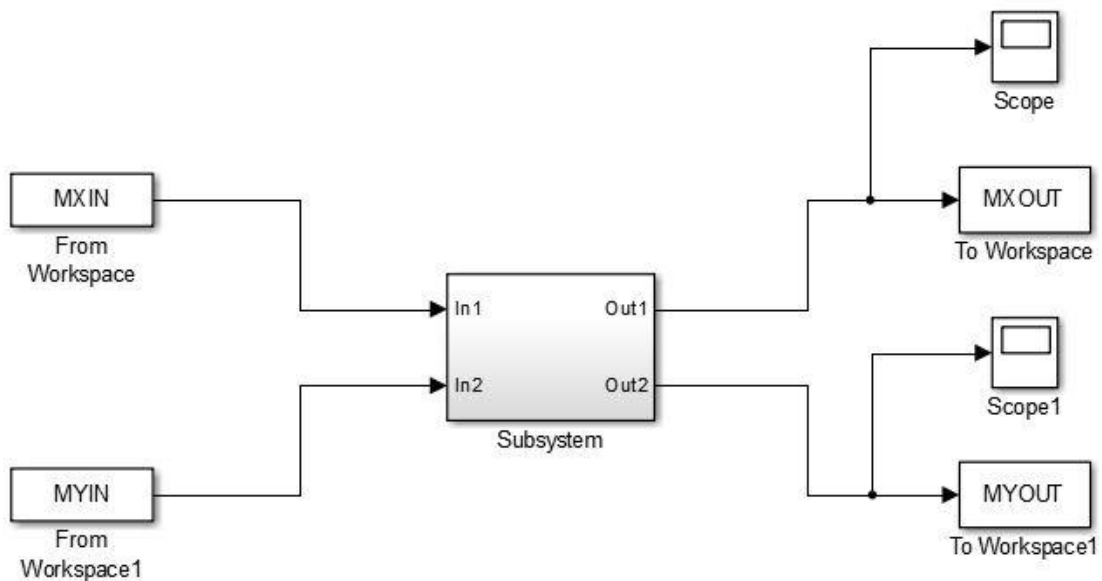


Abbildung 28 Simulink-Modell zum Vergleich von Ergebnissen

Das Simulinkmodell des Reglers wurde entsprechend erweitert, damit man die Ergebnisse im Workspace speichern und anschließend im Scope sehen konnte (Abb. 28).

Im Subsystem des erweiterten Reglers befindet sich der Regler, der bereits in der DLL getestet wurde (Abb. 29).

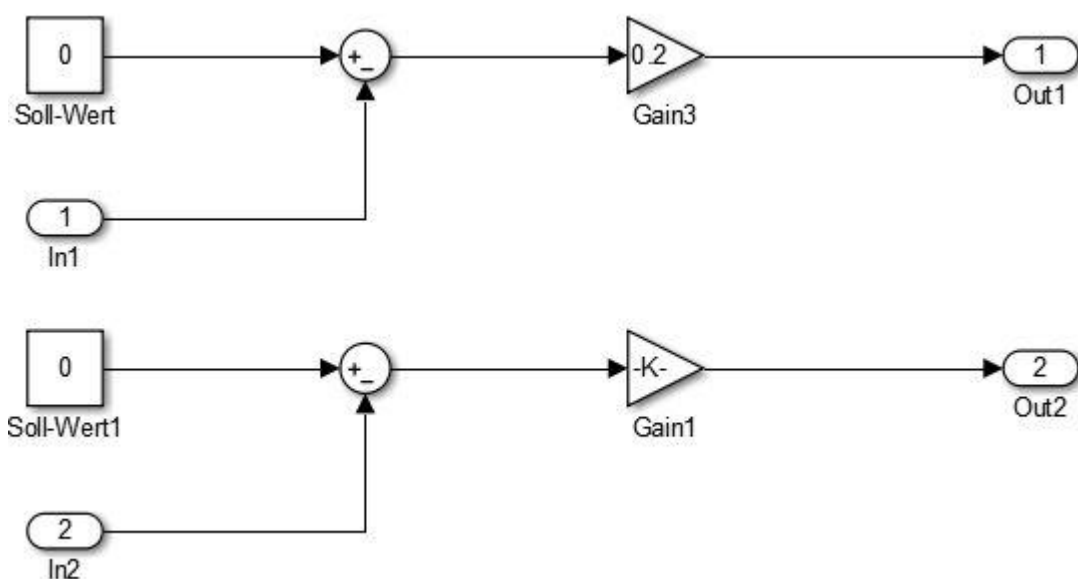


Abbildung 29 Simulink-Modell, das in der DLL getestet wurde

Die Eingangsgrößen für die X- und Y-Achse wurden entsprechend in das Workspace geladen und in den Matrizen namens MXIN und MYIN gespeichert. Die Matrizen wurden folgendermaßen definiert:

```
MXIN=[Time*10,XIN]
```

```
MYIN=[Time*10,YIN]
```

Die Variablen Time, XIN, YIN haben die Werte der ersten 3 Spalten der Kreis.xlsx-Datei.

Nach der Simulation konnten die Ergebnisse im Scope betrachtet werden.

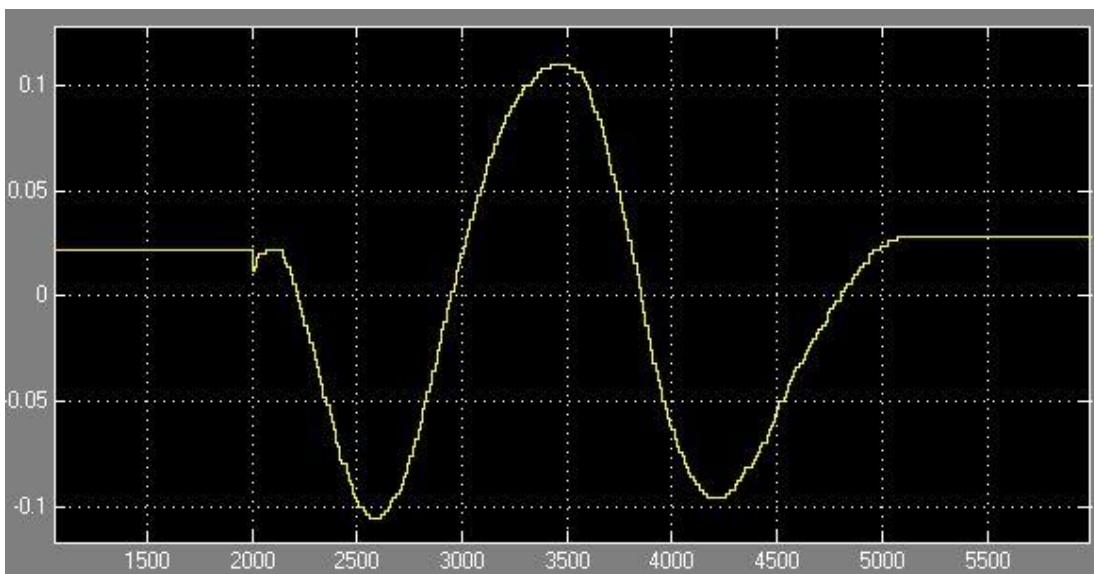


Abbildung 30 Ausgangswerte des Simulinkmodells für die X-Achse

Das Verhalten am Ausgang der X-Achse ist gleich dem Verhalten des Reglers in der DLL (Abb. 30 und Abb. 25).

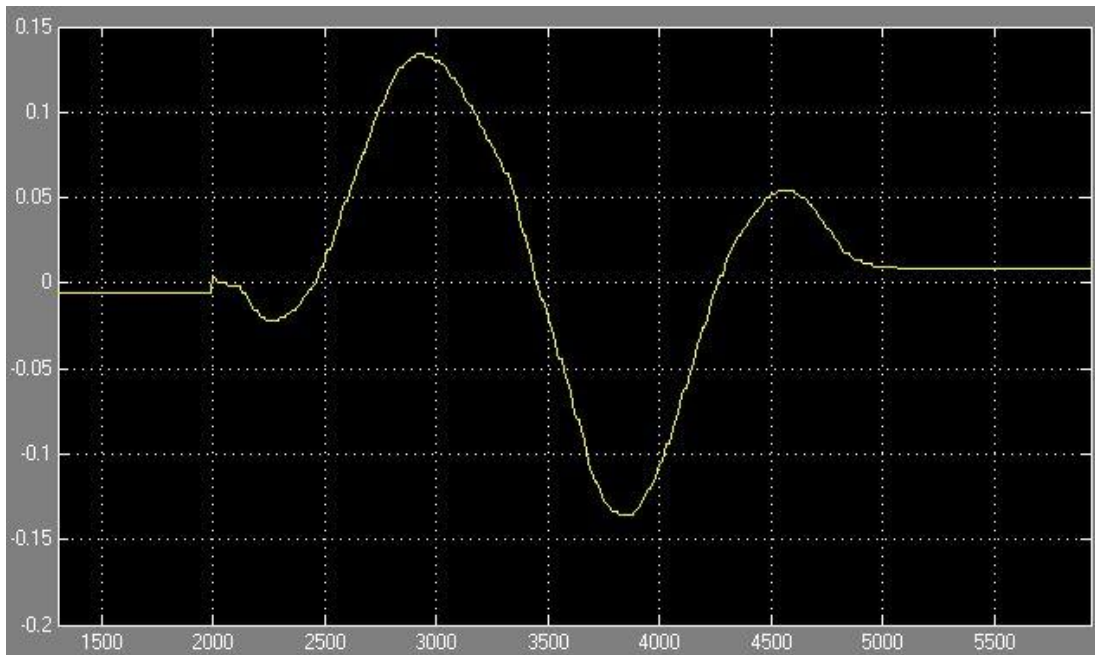


Abbildung 31 Ausgangswerte des Simulinkmodells für die Y-Achse

Auch das dynamische Verhalten des Simulinkmodells für die Y-Achse stimmt mit dem Verhalten des Reglers für die Y-Achse überein (Abb. 31 und Abb. 27) und der Test ist somit beendet.

10.3 Sprungantwort des Systems

Um einen Regler künftig auslegen zu können, der dem Verhalten eines Systems entspricht, ist es wichtig die Sprungantwort des Systems zu ermitteln. Als Sprung wurde die Auslenkung eines der Servomotoren um 5% des maximalen Ausschlags definiert. 100% der Sprungweite entsprechen der Drehung der PhyCARD Plattform um 45 Grad an einer der Achsen. Die Werte wurden aufgenommen, während die Kugel von einer Kante des Displays bis zu der anderen gerollt ist. Der Versuch wurde einzeln für beide Achsen durchgeführt. Über die Ordinate sind die Werte, die der Regler normalerweise von der PhyCARD Plattform bekommt, eingetragen. Über die Abszisse ist die Zeit in Millisekunden aufgetragen. XIN bedeutet in dem Fall die Eingangswerte für den Regler, für die X-Achse und YIN dementsprechend für die Y-Achse.

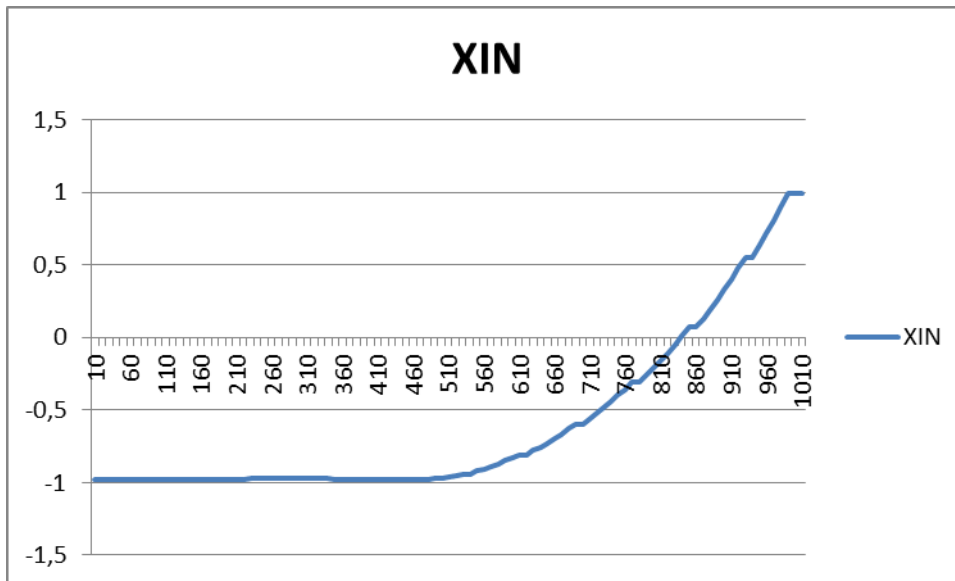


Abbildung 32 Sprungantwort des Systems für die X-Achse

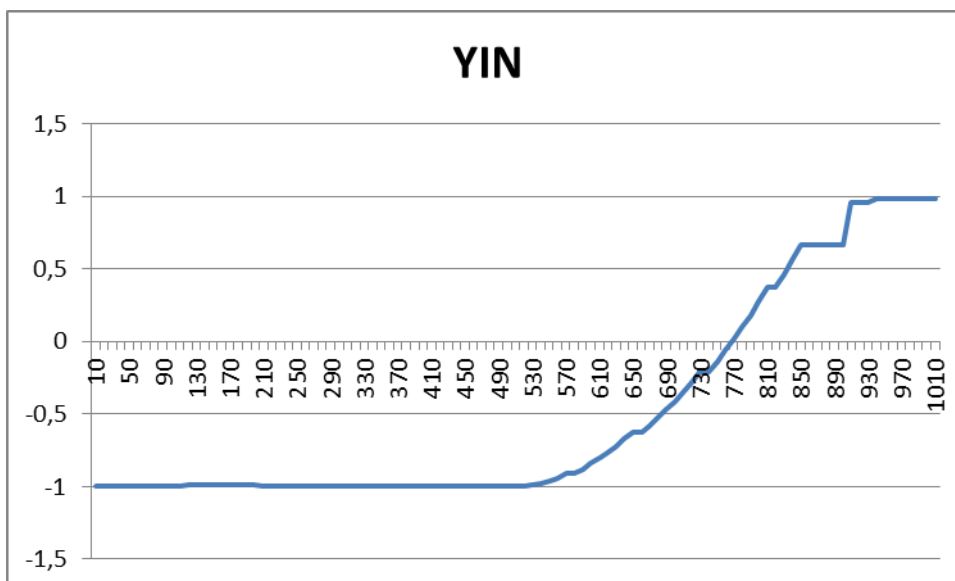


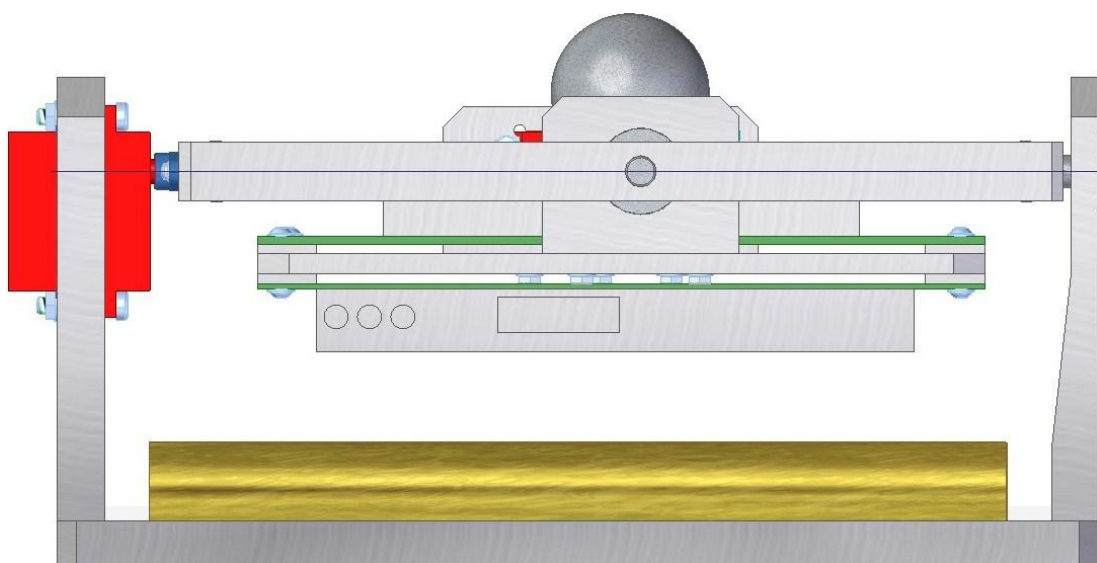
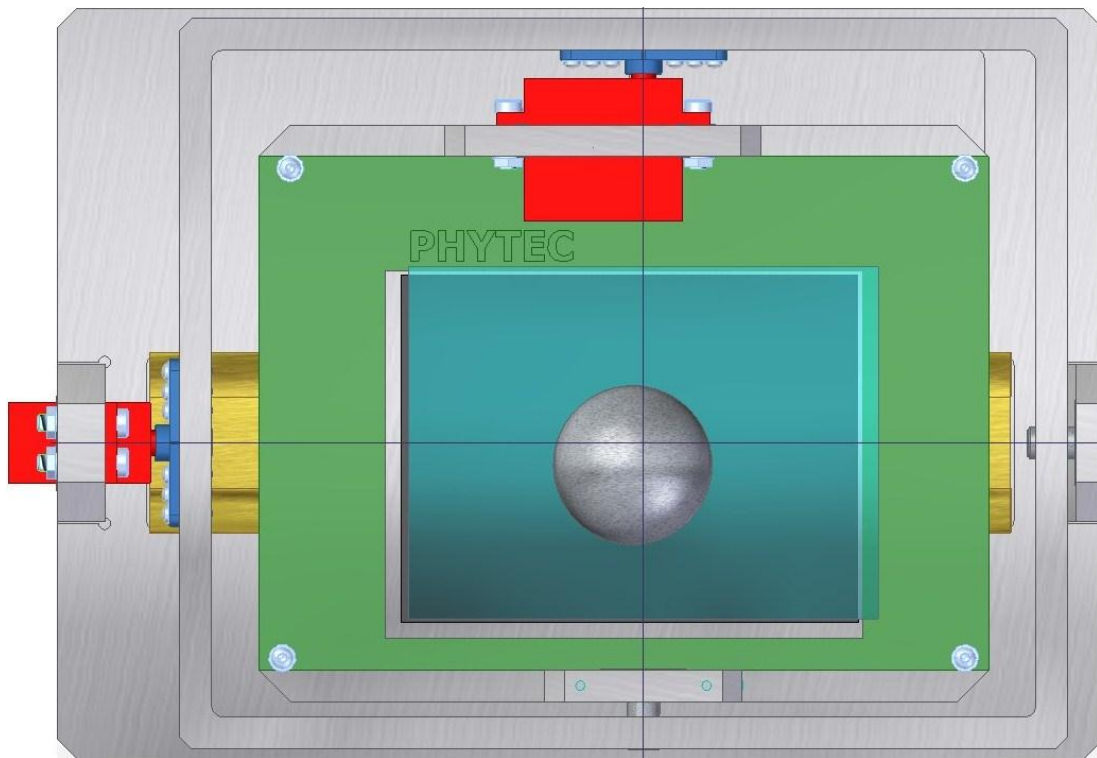
Abbildung 33 Sprungantwort des Systems für die Y-Achse

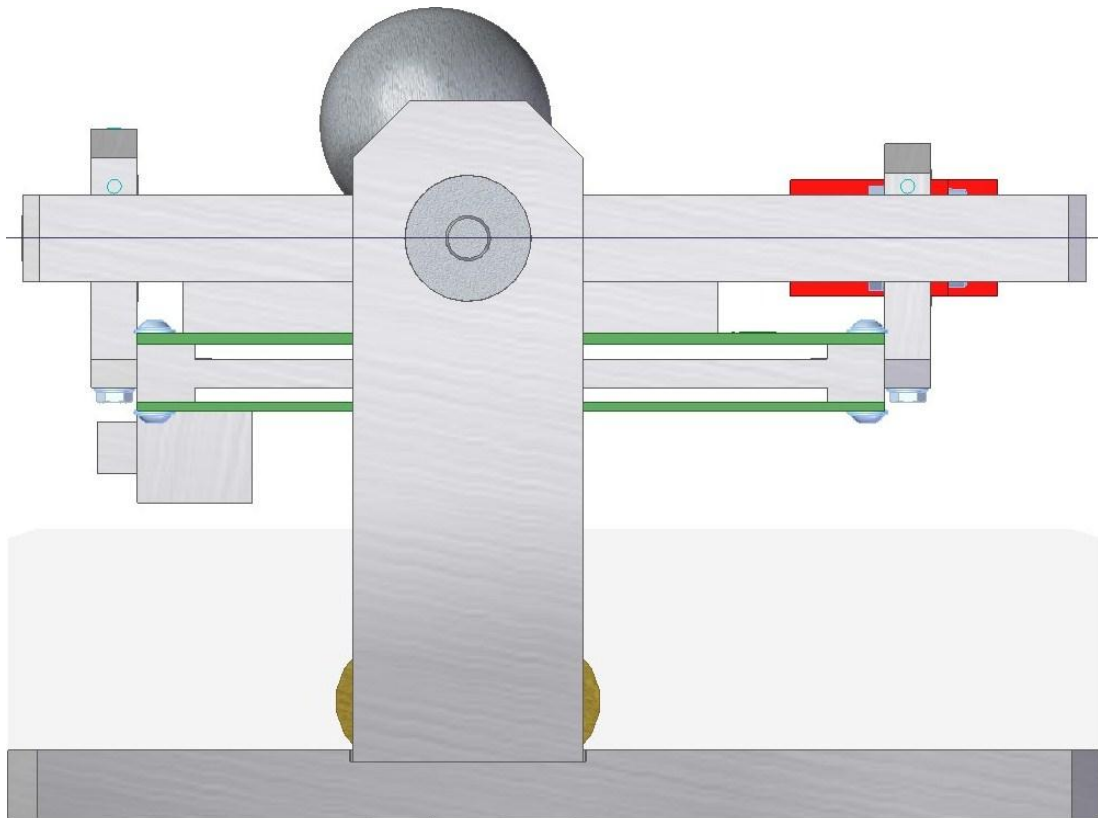
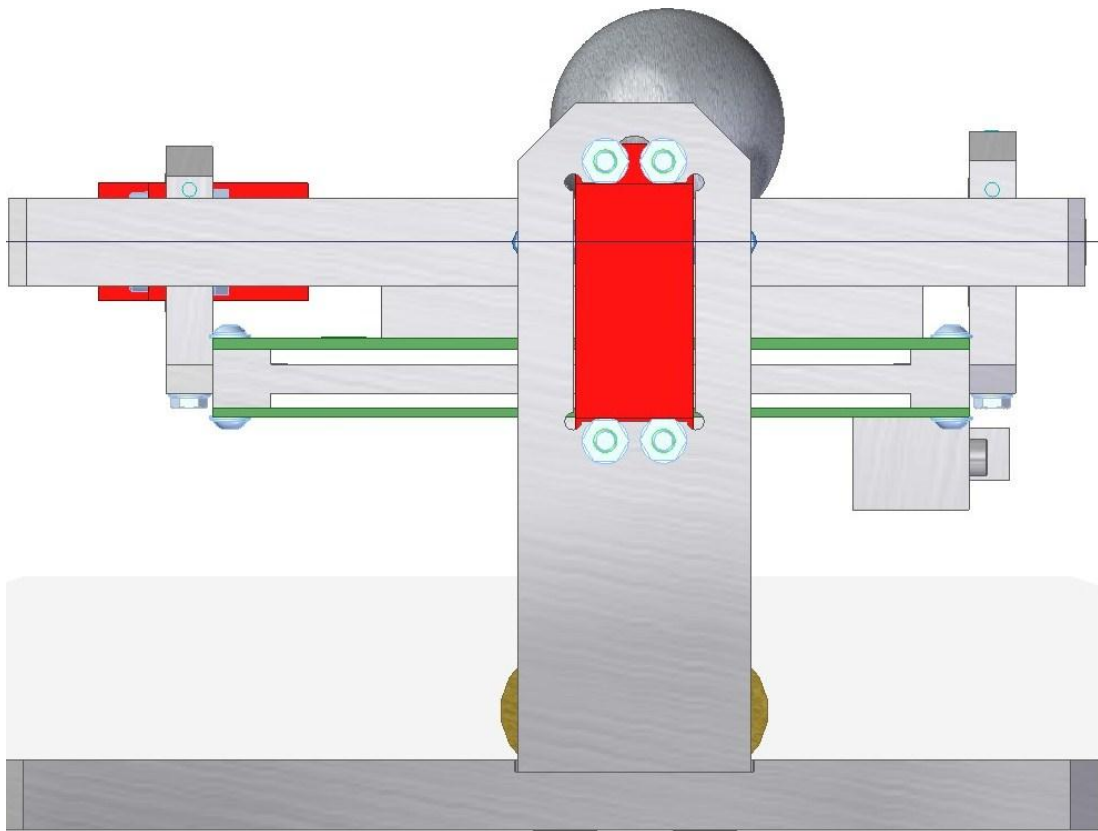
Wie man in der Abbildung 33 bei der Y-Achse sieht, gibt es nach 870ms einen Knick in der Kurve. Dieser Effekt tritt nur bei der Y-Achse auf und muss in Zukunft analysiert werden.

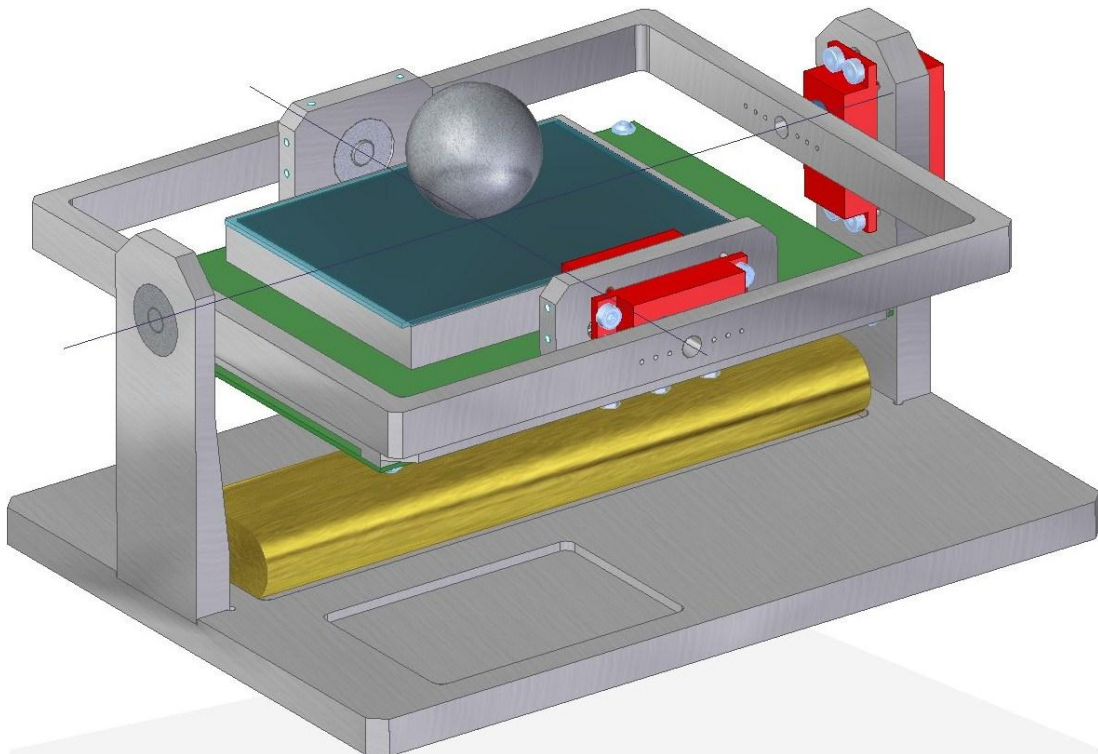
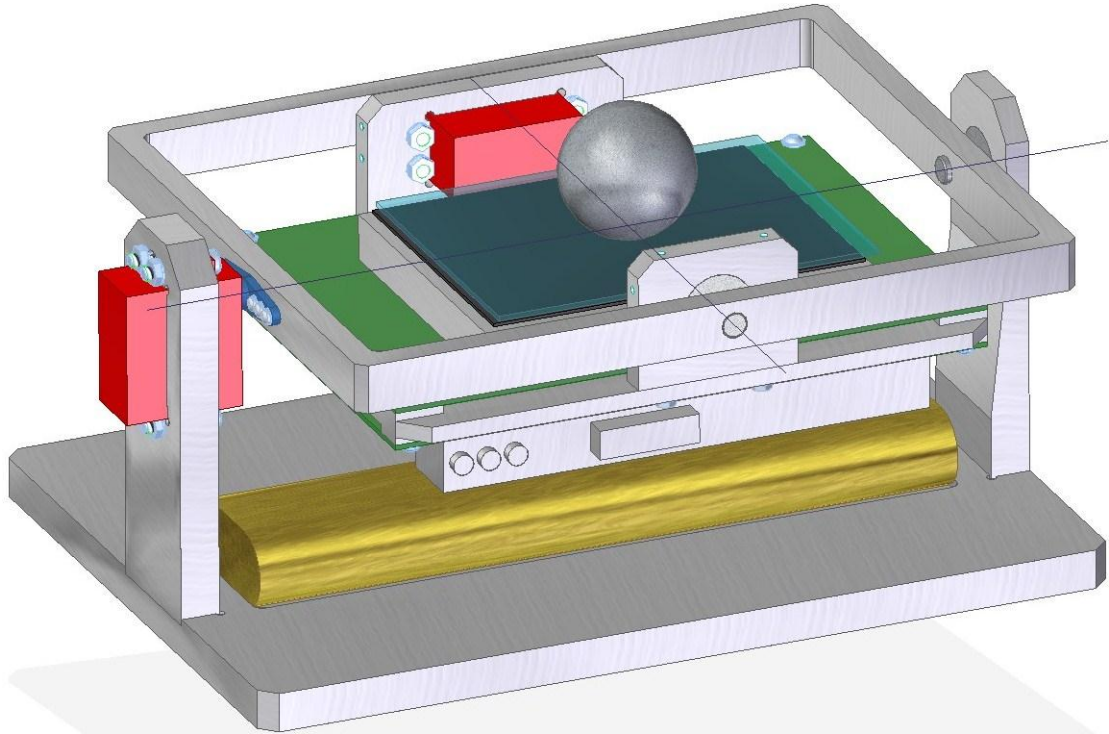
11 Resümee der Bachelorarbeit

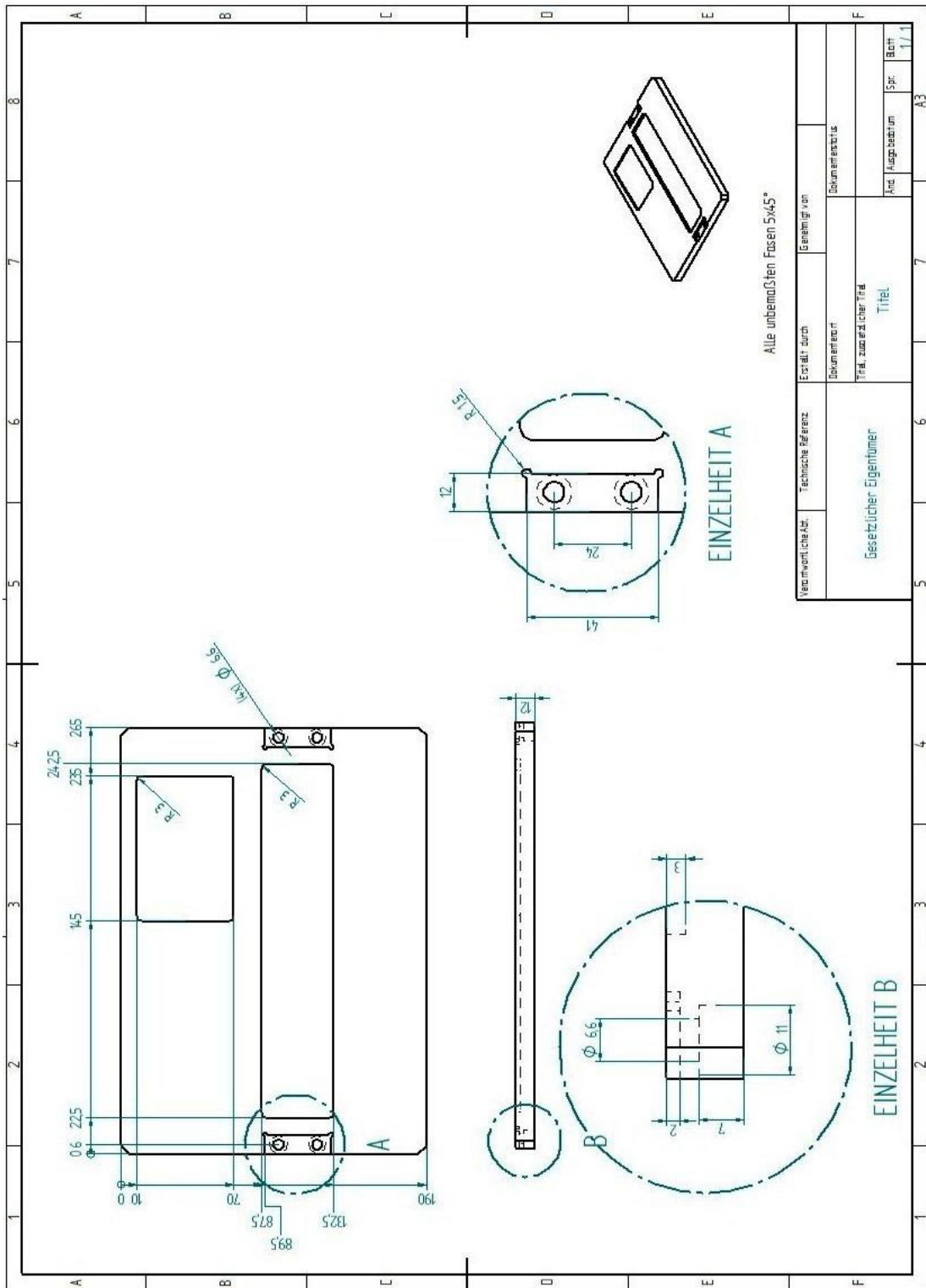
Ziel dieser wissenschaftlichen Arbeit war es, die Simulationstechniken der Echtzeitregelung zu verbessern. Zu diesem Zweck wurde die Schnittstelle zwischen dem Simulationsprogramm Simulink und dem Echtzeitbetriebssystem Windows CE 6.0 aufgebaut und getestet. Damit war es möglich, Regler unterschiedlicher Komplexität wesentlich schneller auf eingebetteten Systemen zu testen, denn die Programmierung des Reglers wird nun komplett vom Simulink übernommen. Was die Simulation/Genauigkeit betrifft, so konnte anhand der Ergebnissen der PhyCARD Plattform gezeigt werden, dass der Regler, der im Simulink entworfen und in einer DLL gespeichert wurde, fehlerfrei funktioniert hat. Andererseits musste festgestellt werden, dass ein kleines Spiel in den Servomotoren die Stabilität des Systems gefährden kann. Abschließend kann festgestellt werden, dass die Frage, ob Simulink tatsächlich eine echte Alternative zum gewöhnlichen Programmierungsweg ist, differenziert beantwortet werden muss. Einerseits kann Simulink aus einem beliebigen Regler den C++ Code generieren, andererseits sollte noch geprüft werden, in wie weit diese Behauptung stimmt und es keine Beschränkungen bei der Code-Generierung gibt. Somit kann diese Arbeit als Grundlage für weitere Forschungsprojekte dienen und wird hoffentlich zur weiteren Entwicklung des Themas „Echtzeitregelung“ noch etwas beitragen. Insgesamt lässt sich hieraus der Schluss ziehen, dass das Thema „Echtzeitregelung“ in Zukunft eine wichtige Rolle spielen wird.

Anhang



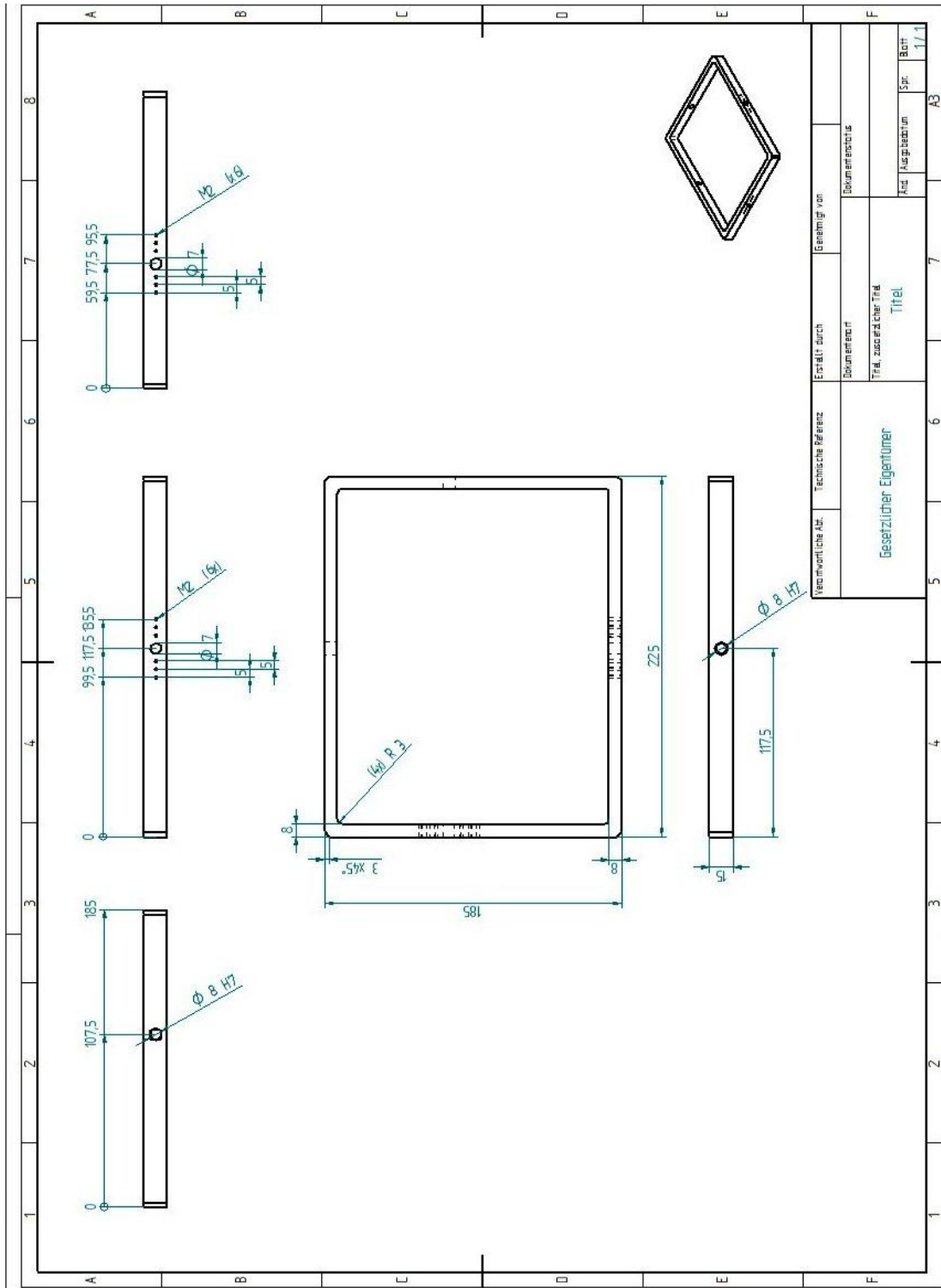


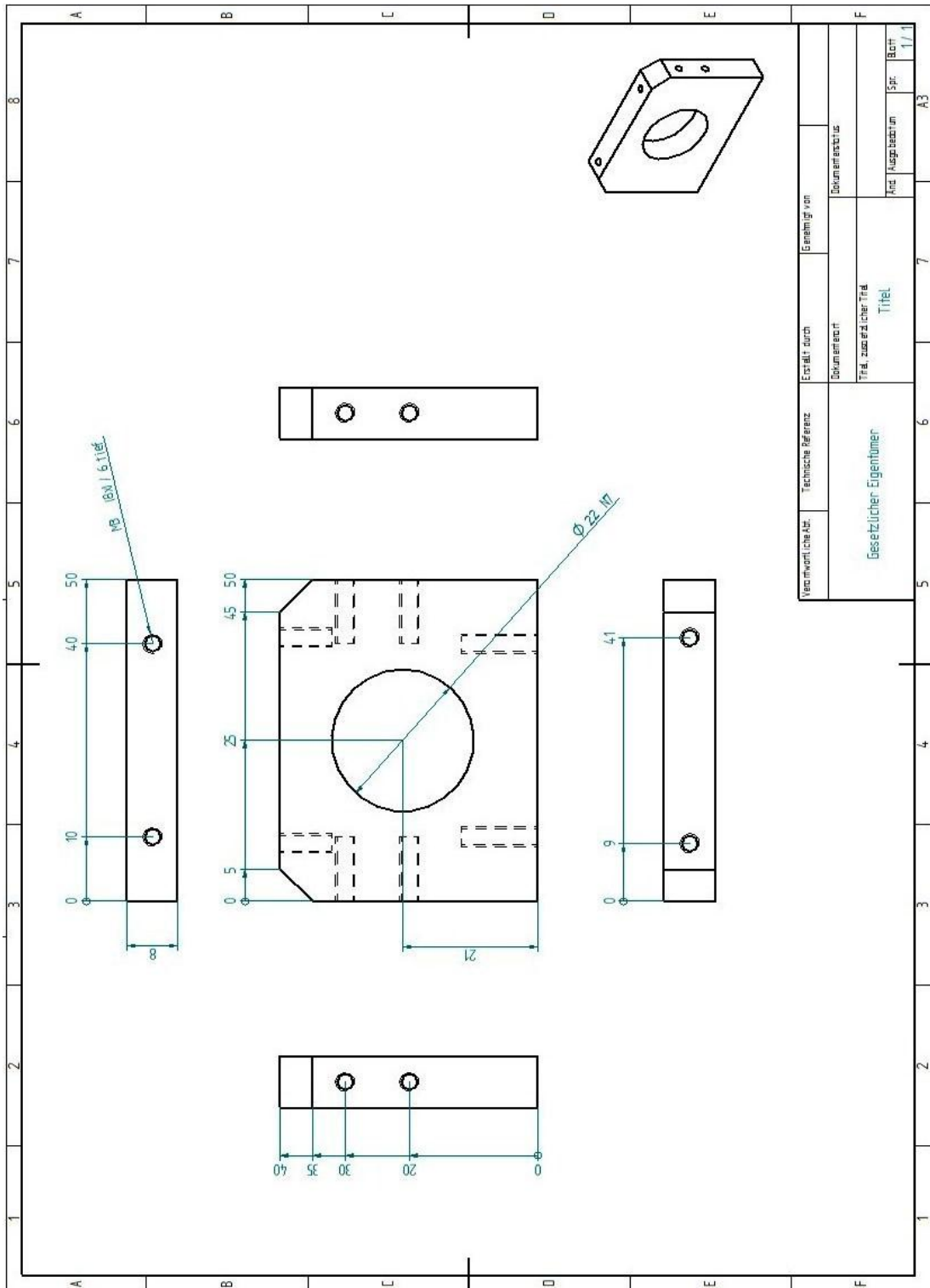




Verantwortliche:	Technische Referenz	Erstellt durch	Genehmigt von
Gesetzlicher Eigentümer		Dokumentiert	Dokumentstatus
		Titel, zusammenf. über Titel	
		Titel	
	Ans.	Ausgabedatum	Spic.
			1 / 1

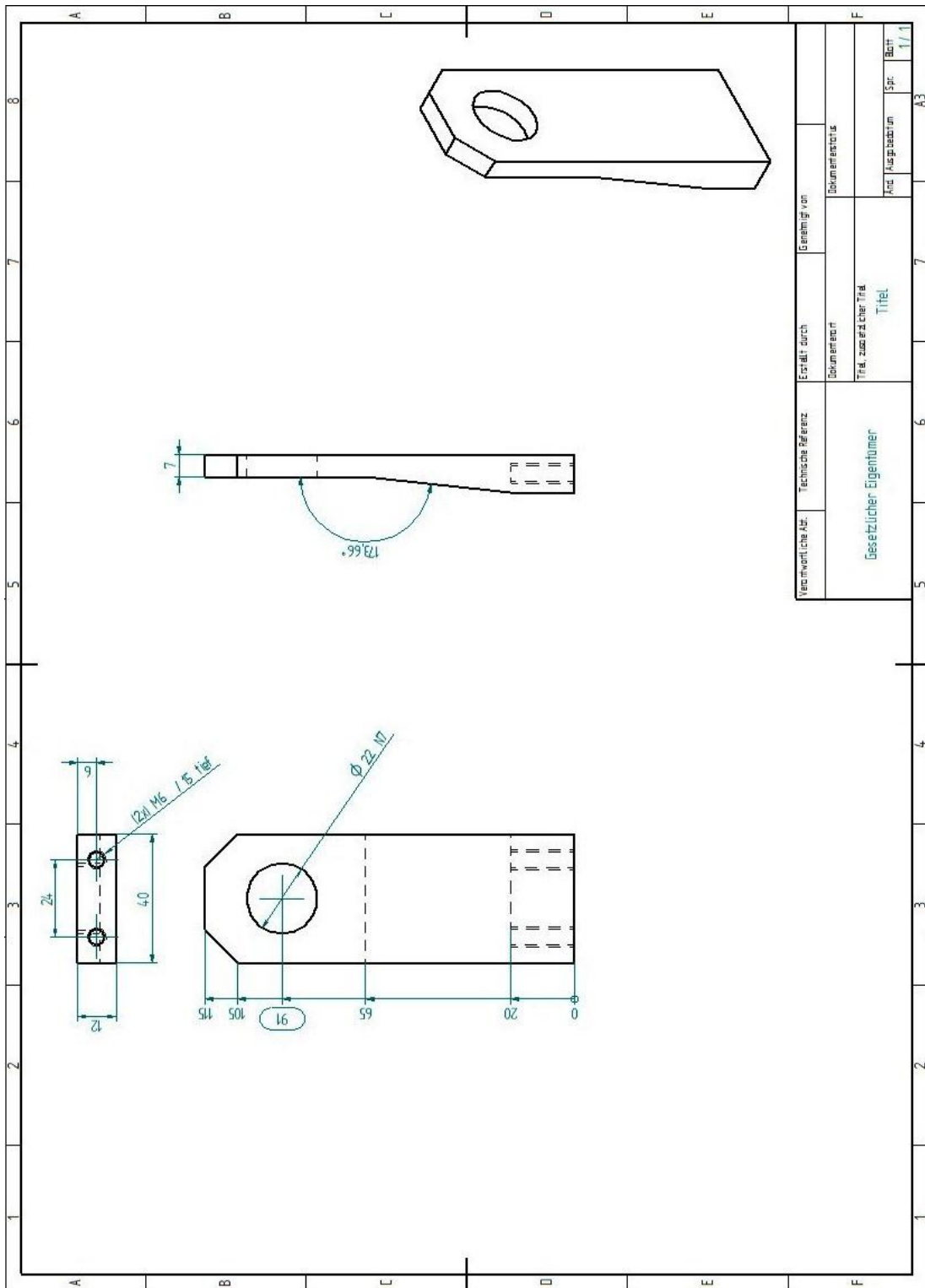
A3

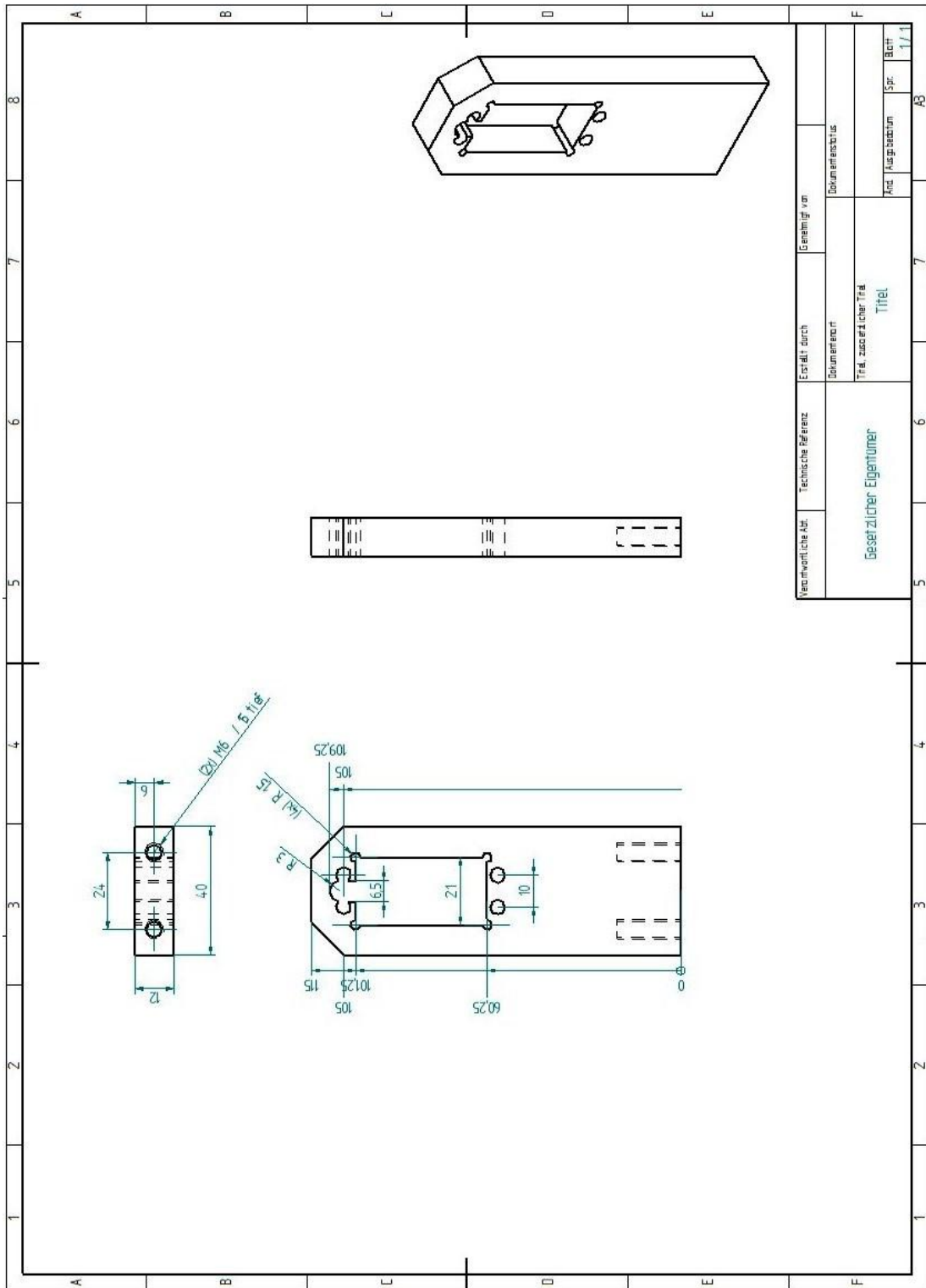


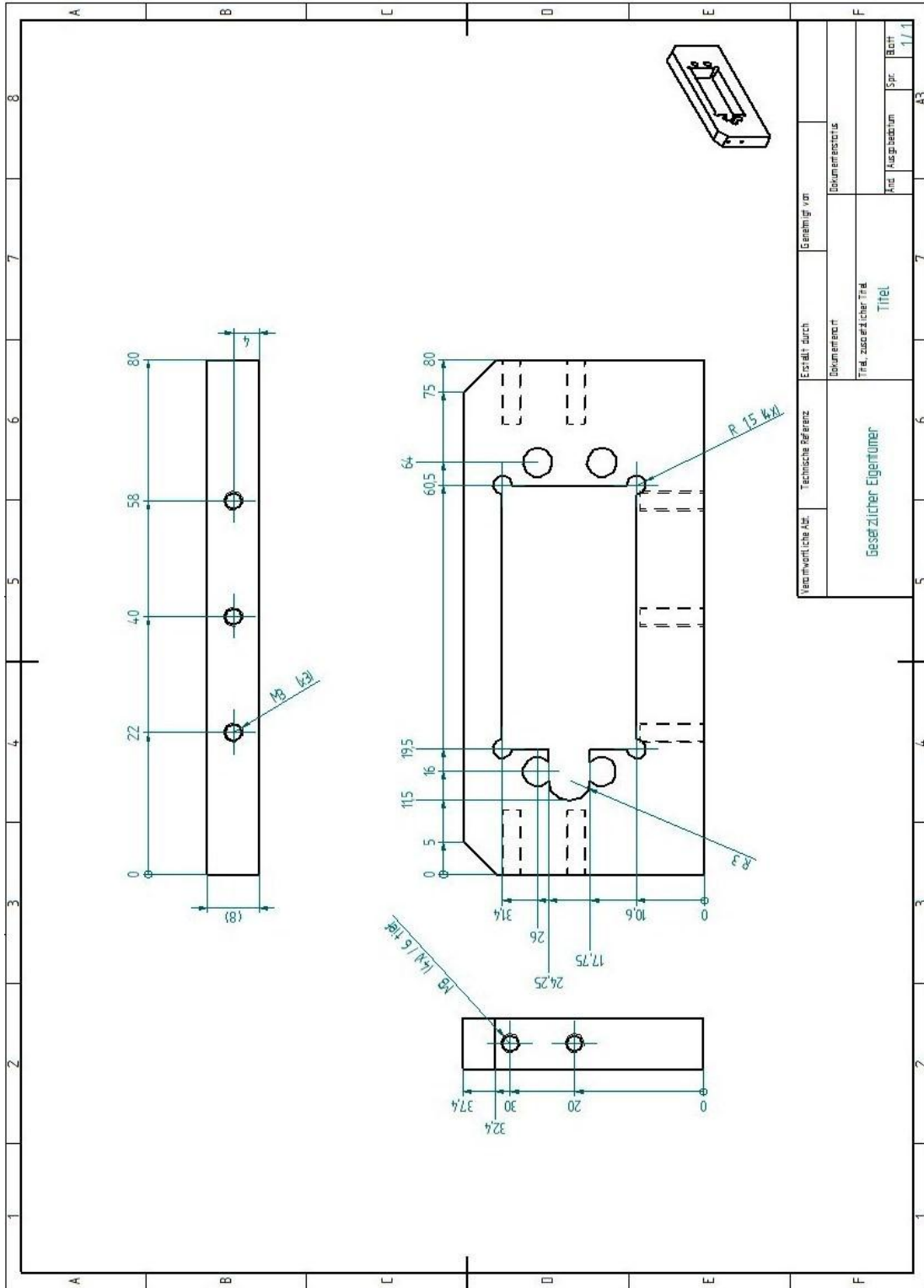


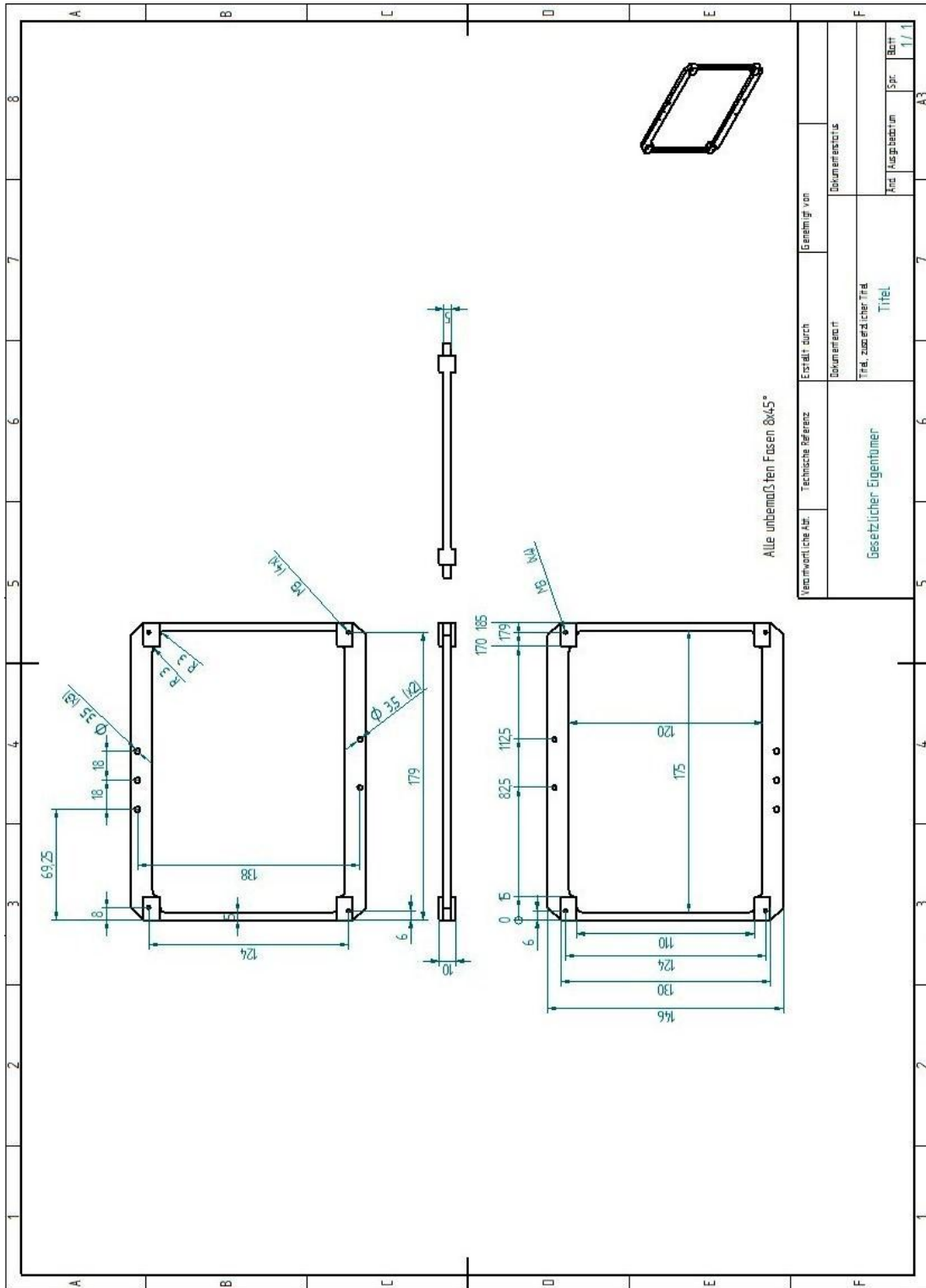
Verantwortliche/r:	Technische Referenz	Erstellt durch	Genehmigt von
Gesetzlicher Eigentümer		Dokumentiert	Dokumentaricus
		Titel, zusammenlicher Titel	
		Titel	
		Art	Ausgabeart
		Spz.	Blatt
			1 / 1

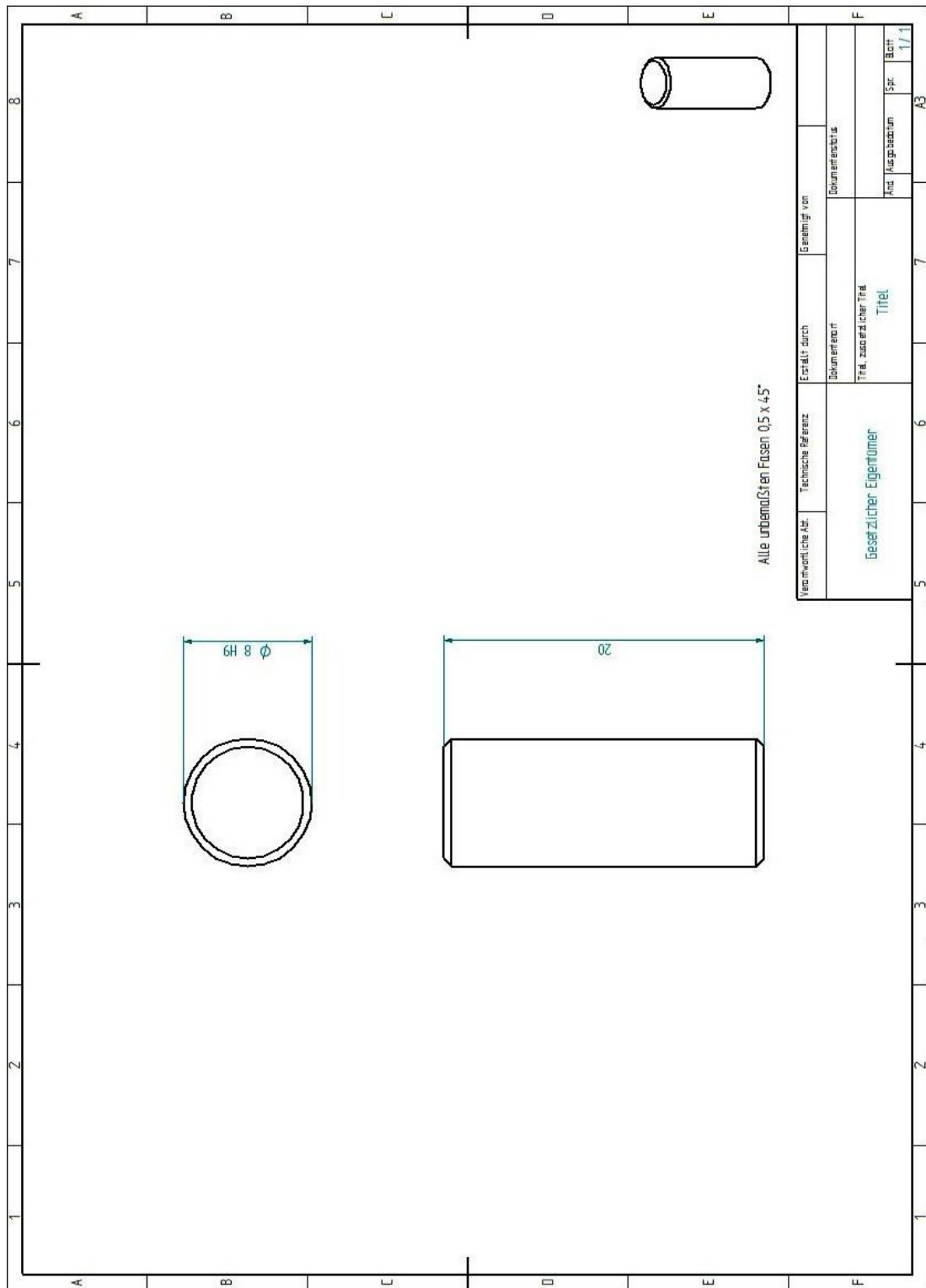
A3

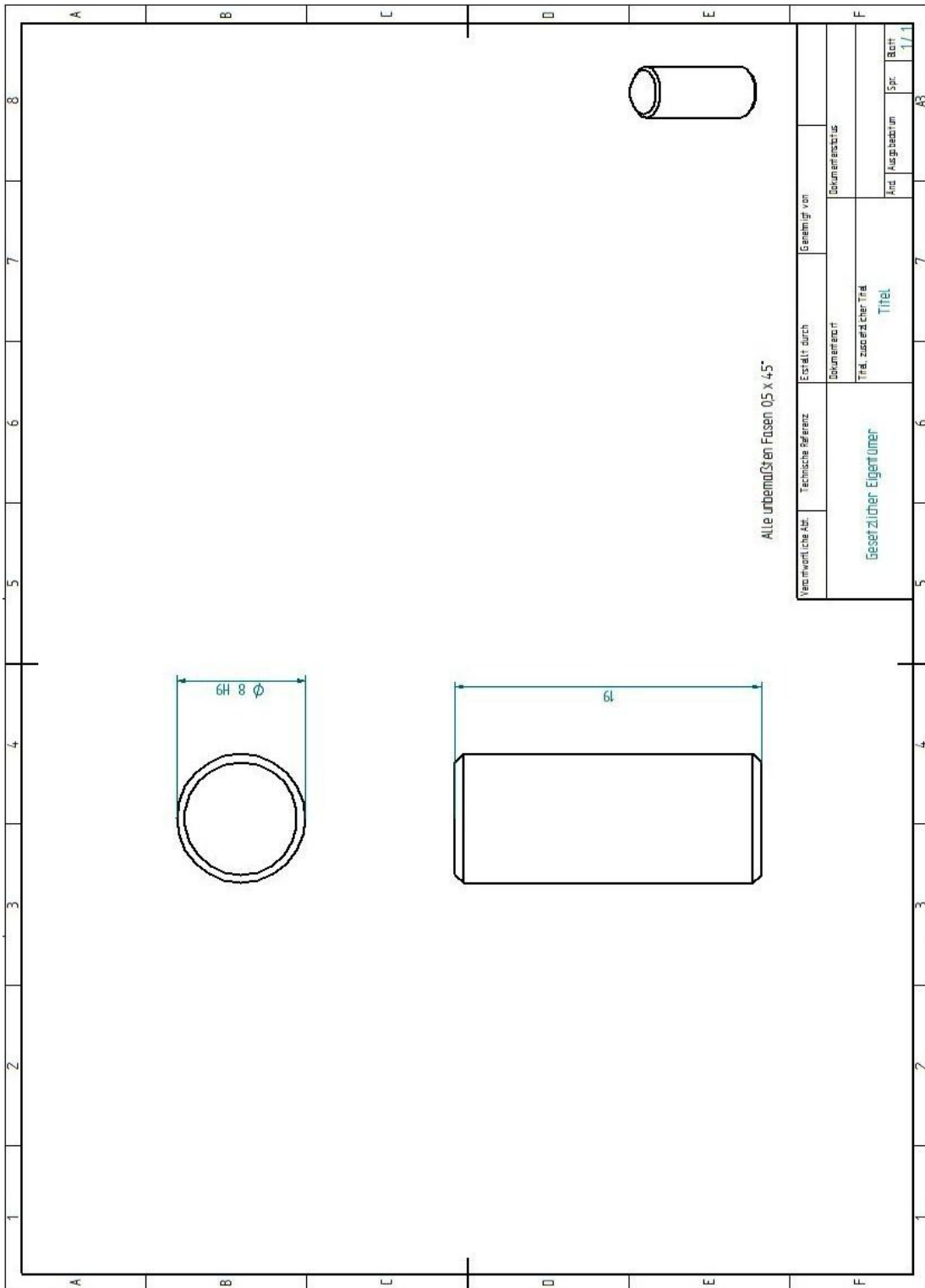












Literaturverzeichnis

Hochschule Rosenheim: Leitfaden für das Erstellen von Abschlussarbeiten in der Fakultät für Ingenieurwissenschaften.

Douglas Boling: Programming for Windows CE 6.0, Developer Reference.

David Kruglinski, George Shepherd, Scott Wingo: Inside of Visual C++ 6.0.

Charles Petzold: Windows-Programmierung, 5 Auflage.

Mathworks GmbH: Support.

Wikipedia: Einbindung von DLL. http://de.wikipedia.org/wiki/Dynamic_Link_Library.

Microsoft AG: Was ist eine DLL. <http://support.microsoft.com/kb/815065/de>.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift