Hochschule **Rosenheim**
University of Applied Sciences

MASTERARBEIT IM MASTERSTUDIENGANG
ANGEWANDTE FORSCHUNG UND ENTWICKLUNG

Konzeptionierung und Bau eines regelungstechnischen Demonstrationsversuchs zum Abfahren einer Kugel auf vorgebbaren Bahnen

Autor: Joseph Lechner
Matrikelnummer: 700914

Prüfer: Prof. Dr. Peter Zentgraf
Datum der Abgabe: 23.02.2017

Inhaltsverzeichnis

1. Prolog

1.1 Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Ausarbeitung der Masterarbeit unterstützt haben.
Im Besonderen danke ich Herrn Prof. Dr. Peter Zentgraf für seine engagierte Unterstützung.

1.2 Erklärung

Hiermit erkläre ich, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Unterschrift:_____

1.3 Ziele der Masterarbeit

Das Ziel dieser Masterarbeit ist das Bauen eines verbesserten Nachfolgesystems eines bereits bestehenden Systems zur Regelung einer Kugel auf vorgebbaren Bahnen.

2. Einleitung

Im Zuge einer Bachelorarbeit wurde bereits ein System zur Regelung der Kurvenbahn einer Kugel auf einer Platte gebaut. Mit der vorangegangenen Projektarbeit wurde das System analysiert und fertiggestellt. Dabei wurde ein regelungstechnisches Modell der Strecke erstellt und anhand des Geräts verifiziert. Ebenso wurden verschiedene Regelungsprogramme mit unterschiedlichen Sollwertvorgaben entwickelt und getestet.
Technisch gesehen handelt es sich bei dem System, „Tabledance“ genannt, um ein Einplatinencomputer mit Touchdisplay, auf dem eine Kugel abrollen kann. Dieses System ist auf einer Kardanlagerung frei beweglich gelagert, wobei jeweils ein Modellbauservomotor den Kippwinkel des Touchdisplays auf der X-Achse und der Y-Achse einstellt. Die Kugelposition wird dabei auf dem Touchdisplay gemäß der Sollwertvorgabe geregelt, indem die Position über das Display gemessen und nach digitaler Verarbeitung mit einem Regelungsalgorithmus die Servomotoren entsprechend angesteuert werden. Der Code für die Regelungsalgorithmen wurde dabei mit Simulink erstellt und nach Umwandlung in eine DLL (dynamische Bibliothek) konnte diese am Gerät verwendet werden. In Abbildung 1 ist vom System ein Bild dargestellt, bei dem der Aufbau gut zu erkennen ist. Für eine genauere Beschreibung des Systems und dem Stand der Technik sei hier auf die vorangegangene Projektarbeit „Modellierung, Reglerauslegung und Bedienung eines drehbar gelagerten Touchpanels zum Abfahren einer Kugel auf vorgebbaren Bahnen“ verwiesen. Nachteilig an diesem Aufbau ist die mit ca. 7cm x 7cm relativ kleine nutzbare Fläche für das Abfahren von Bahnen mit der Kugel. Auch das Ändern von Regelungsalgorithmen geht nur durch Auswechslung der DLL-Datei, wobei jede DLL mit anderen Regelungsparametern zuvor einzeln erstellt werden muss. Zudem ist, aufgrund der genutzten Hardware, keine hochgenaue Regelung realisierbar, weshalb letztendlich entschieden wurde, dass mit den hierbei gesammelten Erfahrungen ein neues, verbessertes Tabledance gebaut werden soll. Dieser soll als regelungstechnischer Demonstrationsversuch eingesetzt werden.

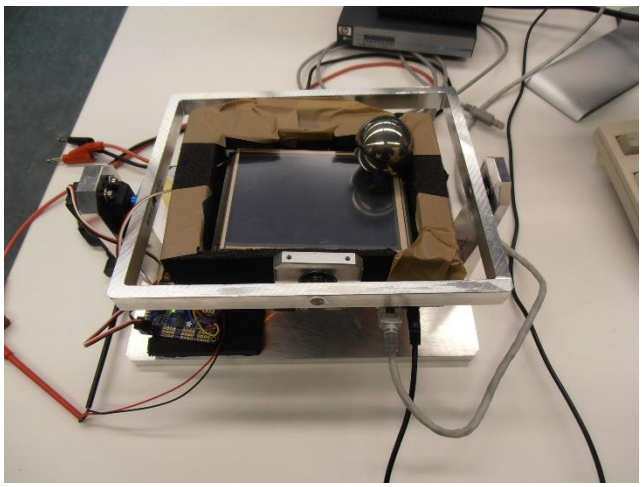


Abbildung 1: Vorgängersystem

3. Geräteplanung

3.1 Anforderungen an das Gerät

Zunächst müssen für die Planung des neuen Tabledance die Anforderungen, die es erfüllen soll, definiert werden.
Folgende Anforderungen wurden definiert:

- Die nutzbare Fläche für das Abfahren von Bahnen mit einer Kugel soll wesentlich größer sein
- Das Gerät soll ein deutlich besseres Regelungsverhalten aufweisen
- Das Gerät soll per Laptop mit Simulink geregelt und gesteuert werden können, wobei es möglich sein soll, Parameter während der Laufzeit zu ändern
- Die Regelungswerte sollen während der Laufzeit angezeigt werden können
- Als Sollwerte sollen ein Festwert, Kreis und Quadrat vorgegeben werden können
- Zudem soll eine beliebige Kurve über das Panel aufgenommen und als Sollwert vorgegeben werden können
- Der gesamte Aufbau soll kompakt und mobil sein
- Als Budget sind bis zu 2000 € vorgesehen

3.2 Grundkonzept

Bevor das System konstruiert werden konnte, musste zu allererst ein sorgfältiges Grundkonzept mit Berücksichtigung der gestellten Anforderungen entwickelt werden. Dieses ist besonders wichtig um später, wenn das System gebaut wurde, aufwendige Änderungen an Systemkomponenten zu vermeiden. Daher wurden zunächst sorgfältig Möglichkeiten zur Realisierung des neuen Tabledance eruiert. Als Grundlage konnten der Aufbau und die Erkenntnisse vom Vorgängersystem genutzt werden.
Wie schon anhand des Vorgängersystems zu erkennen ist, stellen die Regelungseinheit, das Messglied und das Stellglied die wichtigsten Baugruppen an diesem Versuchsaufbau dar. Diese mussten sorgfältig ausgewählt und gut zueinander abgestimmt werden, da diese den Kern des Regelungsversuchs darstellen und maßgeblich für dessen Leistungsfähigkeit verantwortlich sind.

Nach ausgedehnten Recherchen und Überlegungen konnten verschiedene Konzepte für die Realisierung der drei wichtigsten Baugruppen gefunden werden, die nachfolgen kurz erläutert werden.

3.2.1 Konzept zur Regelungseinheit

Die Möglichkeiten zur Realisierung der Regelungseinheit wurden als erstes analysiert, denn diese muss einerseits die wichtigsten Anforderungen wie z.B. Kompatibilität zu Simulink erfüllen und andererseits die beiden anderen wichtigen Komponenten Stellglied und Messglied verknüpfen.
Die Erfahrungen mit dem Vorgängersystem haben gezeigt, dass für ein gutes Regelungsverhalten die Latenz von der Messung der Kugelposition bis zur Stellwertausführung so klein wie möglich gehalten werden soll. Denn schon eine kleine Latenz in Form einer Totzeit von 125 ms hat das Regelungsverhalten des Vorgängers massiv negativ beeinflusst. Das äußert sich vor allem bei höheren Regelungsparametern, bei dem das System eben durch diese Latenz sehr schnell zu schwingen anfängt und instabil wird. Um die geforderte Verbesserung des Regelungsverhaltens zu erreichen, musste bei der Auslegung des ganzen Systems, jedoch insbesondere bei der Regelungseinheit, massiv darauf geachtet werden, dass diese Latenz gering gehalten wird bzw. das System echtzeitfähig ist.
Im Hinblick auf die Anforderung, dass die Regelung über Simulink laufen soll, können folglich lediglich Systeme verwendet werden, die von Simulink in einer geeigneten Form unterstützt werden. Als Grundlage soll, wegen der Mobilität, ein Laptop genutzt werden, auf dem Simulinkmodelle programmiert und ausgeführt werden können.
Simulink selber ist eine eigenständige Erweiterung von Matlab, mit der hauptsächlich mathematisch beschreibbare Vorgänge simuliert werden. Diesbezüglich wurde Simulink unter anderem in der vorangegangenen Projektarbeit zur Simulation des Regelungsverhaltens verwendet. Mit Simulink kann auch kompatible Hardware angesteuert bzw. programmiert werden, was für die Umsetzung des neuen Tabledance relevant ist. Dabei muss, um die Hardware mit Simulink nutzen zu können, das entsprechende „Hardware Support Package“ für die Hardware installiert werden. Manche Hersteller von Hardware bieten eine eigene Unterstützung zu Simulink an, wobei die benötigte Zusatzsoftware bei den jeweiligen Webseiten der Hersteller zu finden ist. Mit der Installation der „Hardware Support Packages“ werden in der Simulinkbibliothek Blöcke zugefügt, mit denen sich die Funktionen der Hardware nutzen lassen. Dabei unterstützt Simulink verschiedene Hardware auf unterschiedliche Arten. Inwiefern Simulink die Zusatzhardware unterstützt, hängt maßgeblich von den unterstützen Betriebsmodi ab. USB-Messkarten, PCI-Messkarten und PCIe-Messkarten werden vorrangig im

Modus „Normal“ unterstützt. Dieser Modus zeichnet sich dadurch aus, dass das Simulinkmodell direkt auf dem Computer bzw. Laptop ausgeführt wird und dabei zumeist über USB-, PCI-, PCIe-Verbindung Funktionen der Zusatzhardware nutzen kann. Ein Beispiel zur Nutzung des Betriebsmodus „Normal“ ist die Messdatenerfassung mit einer Messkarte. Für schnelle Regelungsaufgaben sind die Messkarten aus mehreren Gründen jedoch nicht gut geeignet. Dazu zählt, dass Messkarten nicht für Regelungsaufgaben ausgelegt sind, da diese vorrangig auf das Messen von Signalen und nicht auf das Ansteuern von Aktuatoren spezialisiert sind. Ein anderer Grund ist auch die fehlende Echtzeitfähigkeit des Betriebssystems vom Laptop, welche unvorhersehbare Verzögerungen in der Ausführung des Regelungsalgorithmus verursachen kann. Auch ist z.B. der USB-Bus kein echtzeitfähiges Bussystem. Dies ist dadurch bedingt, dass die übertragenen Daten zunächst gepuffert werden, bevor sie vom Prozessor weiterverarbeitet werden. Die Latenz ist dabei ganz unterschiedlich, liegt aber gewöhnlich zwischen 5 und 20 ms. Deshalb können speziell bei USB-Messkarten noch zusätzliche Latenzen bei der Datenübertragung auftreten, welche sich für die vorliegende Regelungsaufgabe negativ bemerkbar machen würden. Demgegenüber sind Messkarten mit PCI- und PCIe-Bussystem echtzeitfähig, wenn das Betriebssystem echtzeitfähig ist. Dazu bietet Simulink die Erweiterung „Windows Real Time Target“ an. Dieser installiert einen Echtzeitkernel, wodurch das Simulinkmodell auf einem PC mit nicht echtzeitfähigen Betriebssystem in Echtzeit ausgeführt werden kann. Es wird jedoch nur Messhardware mit echtzeitfähigen Bussystemen unterstützt. An einem Desktopcomputer können PCI/PCIe-Messkarten leicht integriert werden, jedoch ist die Integration von solchen Messkarten für das benötigte Laptopsystem sehr umständlich bzw. an vielen Laptops gar nicht möglich.
Neben Messkarten unterstützt Simulink unter anderem noch einige Einplatinencomputer, Mikrocontroller, FPGA-Boards und die Codegenerierung für spezielle Prozessortypen. Hinsichtlich des neuen Tabledance sind vor allem Mikrocontroller und Einplatinencomputer relevant. Unter Mikrocontrollern versteht man dabei Prozessorchips, die neben den Prozessor auch zusätzliche Peripheriefunktionen auf dem Chip integriert haben.
Am Vorgängersystem wurde ein Einplatinencomputer verwendet, der jedoch nicht zu Simulink kompatibel ist. Einplatinencomputer und Mikrocontroller bieten oftmals zahlreiche digitale Ein- und Ausgänge und verschiedene Bussysteme wie SPI und I^2C für die Kommunikation mit anderen Komponenten und können leicht über USB am Laptop angeschlossen werden. Neben dem Betriebsmodus „Deploy to Hardware“, bei dem der Code auf die externe Hardware geladen und dort ausgeführt wird, unterstützen manche auch den Modus „External“. Bei diesem wird der Code ebenso auf die externe Hardware geladen und dort ausgeführt, jedoch ist während der Laufzeit eine Datenübertragung zwischen der externen Hardware und dem Hostcomputer möglich. Dadurch können am Computer bzw. Laptop während der Laufzeit übertragene Daten visualisiert werden.

Auch können die meisten Parameter im Simulinkmodell am PC-System während der Laufzeit geändert werden. Beide Merkmale sind auch geforderte Eigenschaften für das neue System. Hinzu kommt, dass Mikrocontroller und Einplatinencomputer für diesen Modus in einfacher Weise über USB angeschlossen werden können, ohne dass die Echtzeitfähigkeit dadurch verloren geht, da der Simulinkmodellcode auf dem externen Gerät ausgeführt wird und damit keine zeitkritischen Daten über die USB-Verbindung ausgetauscht werden müssen. Dies erleichtert die Integration an einem Laptop erheblich. Die Eigenschaften des Betriebsmodus „External“ zeigen auf, dass sich dieser am besten für die Umsetzung des Regelungssystems eignet.
Industrielle SPS-Systeme oder IPCs (Industriecomputer), womit sich komplexe Regelungs- und Steuerungsaufgabe deutlich einfacher als mit Mikrocontrollern oder Einplatinencomputern verwirklichen lassen, werden hingegen nicht direkt von Simulink unterstützt und können somit nicht für dieses Projekt verwendet werden. Solche industriellen Systeme sind zudem sehr teuer und würden das Budget für dieses Projekt schnell überschreiten. Ein großer Vorteil an SPS- und IPC-Systemen ist, dass mit diesen Systemen viele verschiedene direkt kompatible und leistungsfähige Aktuatoren und Sensoren angeboten werden, die in einfacher Weise angeschlossen werden können. Dahingegen gibt es bei Mikrocontrollern oder Einplatinencomputern keine direkt kompatiblen externen Sensoren oder Aktuatoren, sondern diese müssen im Rahmen der allgemeinen Konnektivitäten und oftmals auch mit zusätzlicher Hardware und Programmierung angepasst werden, was den Aufwand stark erhöht und gleichzeitig die Auswahl stark begrenzt.

3.2.2 Konzept zum Messglied

Mit dem Messglied ist in diesem Fall ein Touchscreen gemeint, auf dem die Kugel die vorgegebenen Kurven abrollen und gleichzeitig die Position der Kugel für die Regelung bestimmt werden kann. Am Vorgängersystem wurde dies mit einem resistiven Touchscreen, das direkt mit einem Display fest verbunden ist, realisiert. Dabei haben die Erfahrungen am Vorgängersystem gezeigt, dass die Auflösung der Positionsmessung für eine gute Regelung möglichst hoch sein soll. Dies ist maßgeblich dem D-Anteil des PD-Reglers zuzuschreiben, der auf die Geschwindigkeit der Kugel reagiert und durch Messrauschen stark beeinfluss wird. Ist die Messung zu ungenau, fängt das System zunächst zu „zittern“ an und schwankt bei höheren D-Regelungsparameter im Zusammenhang mit Latenzen im Stellgliedsystem, was letztendlich zu Instabilität führen kann. Am Vorgängersystem konnte eine Auflösung von 7 - 8 bit erreicht werden, die dennoch für gutes Regelungsverhalten ungenügend waren. Wird dies auf das neue Tabledance bezogen, dessen Touchscreen mehrfach größer sein und ein besseres Regelungsverhalten erreicht werden soll, ist schon für eine gleichbleibende

Regelungsgüte proportional zur Touchscreengröße eine mehrfache Auflösung der Positionsmessung erforderlich. Zusätzlich sollten für ein gutes Regelungsverhalten die Positionsdaten möglichst verzögerungsarm und mit einer Abtastrate von größer als 50 Hz gewonnen werden können. Dadurch sind sehr hohe Anforderungen an den Touchscreen und dessen Ansteuerelektronik gegeben.
Touchscreens unterscheiden sich hauptsächlich in ihrer Funktionsweise und ob sie bereits fest mit einem Display verbunden oder noch frei auf einem Bildschirm montierbar sind. Die Funktionsweisen der Touchscreens werden im Wesentlichen zwischen resistiv und kapazitiv unterschieden.
Kapazitive Touchscreens bestehen aus einer dünnen Glasplatte, auf dessen Unterseite zwei voneinander isolierte Ebenen von transparentem und leitendem Metalloxid in Streifenform aufgebracht sind. Diese sind so zueinander angebracht, dass sich daraus eine Matrixform ergibt, wie es die Abbildung 2 zeigt. Die Kreuzungspunkte der Leitermatrix wirken als Kondensatoren, weil sich die Metalloxidstreifen dort sehr nahe kommen. Im Betrieb werden alle Leiterbahnen mit einer Spannung versorgt, wodurch an den Kondensatoren elektrische Felder entstehen. Nähert sich ein leitendes Objekt, wie z.B. ein Finger, der Glasplatte, wird das elektrische Feld der Kondensatoren beeinflusst, was einen Stromfluss zur Folge hat. Der Berührungspunkt wird mit einem Touchcontroller bestimmt, wobei es mehrere Auswerteverfahren gibt. Der Berührungspunkt kann z.B. mit der Messung der Ströme oder der Kapazität hergeleitet werden.

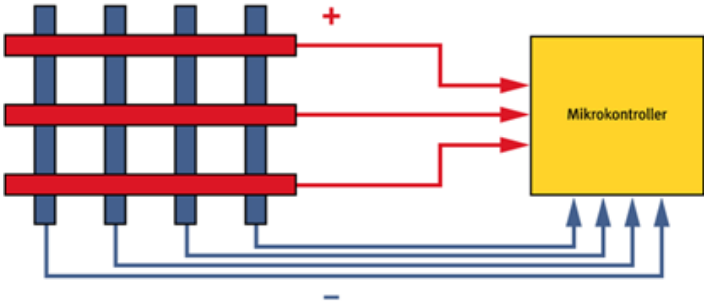


Abbildung 2: Funktionsprinzip eines kapazitiven Touchsensors Quelle [1]

Bei kapazitiven Touchscreens können mehrere Berührungspunkte gleichzeitig erkannt werden, was eine Mehrfingerbedienung erlaubt. Daher werden nahezu alle Smartphones und Tablets mit kapazitiven Touchscreens ausgestattet. Vorteilig für diese Verwendung ist auch die Glasoberfläche, die sehr robust gegen Kratzer und Abnutzung ist.
Resistive Touchscreens hingegen sind baubedingt viel empfindlicher gegenüber Kratzern und Abnutzung. Als Grundlage der resistiven Bauart dient ebenfalls eine dünne Glasplatte. Anders als bei kapazitiven Touchscreens ist auf der Oberseite der Glasplatte eine transparente Metalloxidschicht auf der ganzen Fläche aufgebracht. Darauf befindet sich dazu noch eine elastische Folie, zumeist aus weichem PET (Polyethylenterephthalat), die mit kleinen Abstandspunkten, sogenannten „spacer dots“, von der Glasplatte weggehalten wird. Diese Folie ist auf der Unterseite ebenfalls mit einer durchsichtigen und elektrisch leitenden Metalloxidschicht versehen, sodass bei der Glasplatte als auch bei der Kunststofffolie jeweils die Innenseite elektrisch leitend beschichtet sind. Wird Druck auf diese Folie ausgeübt, berühren sich die zwei leitenden Schichten und erzeugen so eine leitende

Verbindung. Die Position der Druckstelle wird durch Messung des Spannungsabfalls einer angelegten Spannung über die leitenden Schichten, die als Widerstandsstrecken fungieren, bestimmt. Die resistiven Touchscreens unterscheiden sich darauf Bezogen in mehrere Ausführungen der Anschlüsse und Arten der Messung. Nähergehend ist die Funktionsweise der resistiven Touchscreens im Kapitel 4.1.2 beschrieben.

Resistive Touchscreens reagieren folglich auf Druck und können systembedingt, bis auf wenige Ausnahmen, nur einen Druckpunkt gleichzeitig erfassen. Diese Art der Touchscreens wird oftmals in der Industrie eingesetzt, da sie einerseits unempfindlicher gegenüber Schmutz und Wasser sind und andererseits auch mit z.B. Handschuhen bedienen lassen.

Nach Analyse der auf dem Markt angebotenen Touchscreens wurde für die Realisierung des neuen Tabledance auf die Verwendung eines resistiven Touchscreens ohne fest angebauten Display entschieden, wobei nachfolgend erläuterte Argumente ausschlaggebend waren.

Sehr wichtig bei der Wahl der Touchscreenart war die Ansteuerbarkeit hinsichtlich der kompatiblen Hardware von Simulink. Sowohl bei kapazitiven als auch bei resistiven Touchscreens gibt es spezielle Touchcontroller, die teilweise über Bussysteme kommunizieren können. Dafür müssen jedoch spezielle Treiber programmiert oder angewendet werden und man ist auf die Eigenschaften der Touchcontroller angewiesen. In dieser Weise wurde dies am Vorgängersystem realisiert. Die höherwertigen Controllerplatinen bieten oftmals nur eine nominelle Auflösung von 11 bit bei einer maximalen Abtastrate von 50 Hz, was für das geforderte Regelungsverhalten ungenügend ist. Die für Regelungszwecke schlechten Eigenschaften der Touchcontroller sind auf das eigentliche Nutzungsfeld der einfachen Tastendruckerkennung zurückzuführen, wofür diese weitaus ausreichend sind. Als Option hat sich die direkte Ansteuerung des Touchscreens mit einem Mikrocontroller ergeben, was eine dedizierte Touchcontrollerplatine überschüssig macht. Das direkte Betreiben eines kapazitiven Touchscreens ist mit einem Mikrocontroller nicht ohne weiteres möglich. Dagegen kann ein resistives Touchscreen mit digitalen Ein- und Ausgängen und Analog-zu-Digital-Konvertern betrieben werden, welche durch Mikrocontroller oder Einplatinencomputer bereitgestellt werden können. Dazu sind resistive Touchscreens viel günstiger und vor allem auch besser in größeren Ausführungen zu erwerben, wobei speziell bei größeren Touchscreens selten ein Display integriert ist. Die Einschränkung der resistiven Touchscreens, dass nur ein Druckpunkt gleichzeitig bestimmt werden kann, ist in diesem Fall nicht ausschlaggebend, da nur die Position einer Kugel bestimmt werden muss. Mit diesen Argumenten konnte sich eindeutig für die Verwendung eines resistiven Touchscreens entschieden werden, gegenüber dem Vorgängersystem jedoch ohne fest verbauten Display und ohne integrierter Auswerteelektronik.

3.2.3 Konzept zum Stellglied

Das Stellglied umfasst hierbei eine Einheit zum Verstellen des Neigungswinkels vom Touchscreen. Die Erfahrungen mit dem Vorgängermodell haben gezeigt, dass eine hohe Regelungsgüte nur mit einer schnellen und vor allem genauen Neigungswinkelregelung mit einer Genauigkeit von <0,2° pro Schritt erreicht werden kann. Das Stellgliedsystem stellt eine zusätzliche Regelungsschleife im Gesamtsystem dar, weil das Anfahren von Neigungswinkeln nur mit geregelten Aktuatoren bewerkstelligt werden kann. Ausnahmen sind dabei die sogenannten „open loop“-Ansteuerungen von Schrittmotoren.
Ähnlich wie am Vorgängersystem wurde für die Aufhängung des Touchscreens eine Kardanlagerung vorgesehen, wobei die Achsen genau mittig zum Touchscreen angeordnet wurden. Die X- bzw. Y-Achse wird am Vorgängersystem jeweils mit einem direkt angekoppelten Modellbauservomotor angetrieben, welcher mit einem Drehmoment von ca. 0,8 Nm stark genug ausgelegt ist.
Demgegenüber sind am neuen System, aufgrund der Größe, viel höhere Drehmomente zum Verstellen des Neigungswinkels vom Touchscreen nötig. Überschlägige Berechnungen zeigten, dass allein für die Stahlkugel mit einem Gewicht von 350g ohne Berücksichtigung der Kinematik mindestens 1 Nm Drehmoment eingeplant werden müssen. Dabei gilt: Umso größer das bereitgestellte Drehmoment, desto genauer, schneller und stabiler ist die Regelung.
Mit vielen Mikrocontrollern und Einplatinencomputern können ohne viel zusätzlicher Hardware Modellbauservomotoren und Schrittmotoren angesteuert werden. Die beiden unteren Abbildungen 3 und 4 zeigen jeweils ein Beispiel eines Schrittmotors und eines Modellbauservomotors. Mit entsprechender Regelungs- und Leistungselektronik sind auch andere Motortypen steuerbar, wobei dafür z.B. die Bussysteme I²C und SPI zur Verfügung stehen.

Abbildung 3: Schrittmotor, Quelle [2]

Abbildung 4: Modellbauservo, Quelle [2]

3. Geräteplanung

Nach Abwägen vieler Möglichkeiten konnte sich für ein Stellgliedsystem auf Basis von Modellbauservomotoren im Verbund mit einem Zahnriemengetriebe entschieden werden. Modellbauservomotoren bieten sehr hohe Drehmomente von bis zu 4 Nm und sind aufgrund ihres eigentlichen Einsatzzwecks für z.B. Modellbauflugzeuge mit gewöhnlich 70 g besonders klein und leicht. Das hohe Drehmoment wird dabei mit mehrstufigen Zahnradgetrieben erreicht. Die Antriebseinheit für die Y-Achse ist bezüglich des Gewichts unproblematisch, wohingegen das Gewicht der Antriebseinheit von der X-Achse dem Motor von der Y-Achse als Drehmoment zur Last fällt. Berücksichtigt man die Größe des Touchscreens und die dadurch bedingte Hebellänge vom Motorgewicht ist klar, dass leichte Motoren mit hohem Drehmoment vom Vorteil sind. Außerdem wird durch ein geringes Gewicht die Trägheitsmasse der bewegten Einheit des Tabledance geringer, womit das Regelungsverhalten positiv beeinflusst wird. Je nach Modell können mit Modellbauservomotoren hohe Stellgenauigkeiten erreicht werden, die zwar im sogenannten Mikroschrittbetrieb auch bei Schrittmotoren erreicht werden können, letztere aber nicht für hohe Drehmomente ausgelegt sind. Im Gegensatz zu Servomotoren, die eine integrierte Regelung und einen Absolutwertgeber am Motor verbaut haben, können bei „open loop“ angesteuerten Schrittmotoren durch Überlastung des Haltedrehmoments ein sogenannter Schrittverlust und somit die der Verlust der Schaftposition auftreten. Moderne Ansteuerungen von typischen Hybridschrittmotoren können diese Schrittverluste detektieren und ausgleichen. Nichtsdestotrotz sollten Schrittmotoren für ihren Einsatzzweck überdimensioniert sein und sind bezüglich den angebotenen Leistungsklassen eher für leichtere und gewichtsunkritische Positionieraufgaben geeignet. Typischerweise haben Schrittmotoren bei einem maximalen Haltedrehmoment von 1 Nm schon ein Gewicht von 600 g und sind somit als direkt angekoppelter Antrieb ungeeignet. Überlegt wurde deshalb auch ein indirekter Antrieb der Neigungsverstellung über Gewindetriebe, wofür Schrittmotoren als Antrieb gut geeignet wären. Dennoch wurde sich letztendlich für die Variante mit Modellbauservomotoren in Verbindung mit Zahnriemengetrieben entschieden.

4. Konstruktion und Auslegung

4.1 Komponenten

4.1.1 Mikrocontroller

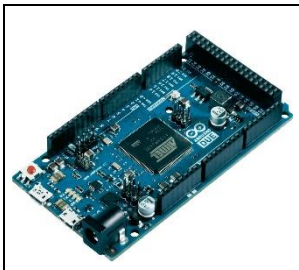
Als Regelungseinheit wurde das zu Simulink kompatible Mikrocontrollerboard Arduino Due für die Umsetzung dieses Projekts ausgewählt, wobei nachfolgend die Gründe erläutert sind.
Mikrocontroller, wie die von Simulink unterstützte Arduino-Baureihe, zeichnen sich gegenüber Einplatinencomputern damit aus, dass sie von Grund auf echtzeitfähig sind, solange der Mikrocontroller nicht überlastet wird. Das ist dadurch bedingt, dass Mikrocontroller ohne Betriebssystem arbeiten und so stets nur ihren Programmcode abarbeiten, während Einplatinencomputer, wie der in Simulink unterstützte und im Hobbybereich beliebte Raspberry Pi, ein Betriebssystem benötigen, das die Abarbeitung des Programmcodes zugunsten anderer Programme und Aufgaben zeitweise unterbrechen kann. Dies bedingt, abgesehen von den Konnektivitäten, auch, dass Einplatinencomputer eher für Projekte in den Bereichen „Internet der Dinge“ und Heimautomatisierung geeignet sind, während Mikrocontroller besser für zeitkritische Regelungsprojekte verwendet werden können. Auch die Peripheriefunktionen von Mikrocontrollern sind besser für technische Regelungsprojekte geeignet. Demgegenüber verleiht das Betriebssystem den Einplatinencomputern die Möglichkeit mehrere Programme gleichzeitig zu betreiben und gewährt viel höhere Kompatibilität zu anderen Geräten wie z.B. Webcams und Monitoren. Für die Realisierung des Tabledance sind die Echtzeitfähigkeit, systemrelevante Anschlüsse und vor allem direkt integrierte A/D-Wandler der Aruduino-Mikrocontroller wichtiger als die Flexibilität und Kompatibilität der Einplatinencomputer.
Simulink unterstützt diesbezüglich unter anderem die ganze Arduino-Baureihe. Bei Arduino-Boards handelt es sich um betriebsfertige „open hardware“ Mikrocontroller, die für das Kennenlernen von Mikrocontrollern für Studenten entworfen wurden und gerne für kleine Projekte eingesetzt werden. „Open hardware“ heißt in diesem Zusammenhang, dass jeder diese Hardware nach den offengelegten Konstruktionsdaten und Schaltpläne herstellen darf, weswegen diese Boards relativ günstig sind.
Das bekannte Arduino Uno gehört mit seinem übersichtlichen Aufbau und ca. 15 € Preis zum günstigen Einsteigermodell, während das Arduino Mega zur Mittelklasse und der verwendete Arduino Due zum leistungsstärksten Modell gehört. Von Simulink werden neben dem Betriebsmodus „deploy to hardware“ nur der Arduino Due und Mega mit dem Betriebsmodus „External“ unterstützt. Obwohl es bei dem verwendeten Arduino Due um das leistungsfähigere und modernere Board

handelt, hat der Arduino Mega gewisse andere Vorzüge, die im Laufe der Arbeit teilweise aufgezeigt werden.

In der Tabelle 1 sind die wichtigsten Merkmale vom Arduino Due zusammengefasst. Zu Vergleichszwecken sind auch die Merkmale vom Arduino Mega und dem Raspberry Pi aufgelistet.

In diesem Zusammenhang sei noch zu erwähnen, dass der Arduino Due und der Arduino Mega weitgehend pinkompatibel sind, um die Kompatibilität zu den zahlreichen optionalen Aufsteckboards sicherzustellen.

Tabelle 1: Technische Daten

			
	Arduino Due	**Arduino Mega**	**Rasperry Pi 3**
Prozessor	Arm Cortex-M3, 32 bit	ATmega2560, 8 bit	Arm Cortex-A53, 64 bit
Anzahl Prozessorkerne	1	1	4
Prozessortakt	84 MHz	16 MHz	1,2 GHz
Betriebsspannung	3,3 V	5 V	3,3 V
Speicher	512 kB Flash Speicher 96 kB SRAM	256 kB Flash Speicher 8 Kb SRAM	1 GB RAM + Festplatte
Pins	54 digital Pins	54 digital Pins	40 GPIO Pins
A/D-Wandler	12 x 12 bit	12 x 10 bit	-
D/A-Wandler	2 x 10 bit	-	-
SPI	ja	ja	ja
I²C	ja	ja	ja
seriell	ja	ja	ja
weitere Anschlüsse	2 x USB	1 x USB	4 x USB, HDMI, Ethernet, Bluetooth, WiFi, Micro SD, 3,5 mm Klinke

Im Arduino Due ist ein 32 bit ARM Cortex-M3 Prozessor verbaut, der auf 84 Mhz taktet. Demgegenüber besitzt der Arduino Mega einen auf 16 Mhz taktenden 8 bit ATmega2560. Auch wenn die Prozessorleistung noch von der Architektur abhängt, kann man die Leistungsfähigkeit mit den Daten schon grob einschätzen. Bei 32-bit Operationen ist der Arduino Due schätzungsweise 16-fach schneller als der Arduino Mega, was sich aus der Taktfrequenz und der Registerbitbreite berechnet. Wie sich herausstellte, wird die weitaus höhere Rechenleistung vom Aruduino Due für das neue Tabledance-Programm auch benötigt, weshalb unter den Arduino-Mikrocontrollern keine sinnvolle Alternative zum Due besteht. Vergleichsweise bietet der Raspberry Pi mit seinem 1,2 Ghz 64 bit Arm

Cortex A-53 Quadcore theoretisch zwar viel mehr Leistungsressourcen, die aber im eigentlichen Einsatzfeld der Einplatinencomputer auch gebraucht werden.

Für die effektive Nutzung der Leistungsressourcen der Arduino-Boards wird die für Mikrocontroller übliche Programmiersprache C/C++ verwendet. Der Nachteil von C/C++ gegenüber den meisten anderen Hochsprachen ist die aufwendigere Programmierung.

Für die Arduino Mikrocontroller gibt es trotz unterschiedlicher Prozessorarchitekturen eine einheitliche, übersichtliche und einfach anzuwendende IDE (integrated development enviroment), die für das Programmieren und das Übertragen der Programme per USB verwendet werden kann.

Neben den Prozessor ist die verwendete Betriebsspannung ein prägnanter Unterschied zwischen dem Arduino Mega und dem Arduino Due. Während der Arduino Due auf 3,3 V arbeitet, verwendet das Arduino Mega die weit verbreitete CMOS-Logik kompatible 5 V Spannung, die auch teilweise zur 3,3 V Logik kompatibel ist. Will man 5 V Logik mit dem Arduino Due verknüpfen, ist in den meisten Fällen ein Pegelwandler notwendig.

In der Abbildung 5 ist das Arduino Due in der Draufsicht dargestellt, auf der auch gut die zahlreichen beschrifteten Anschlüsse zu erkennen sind.

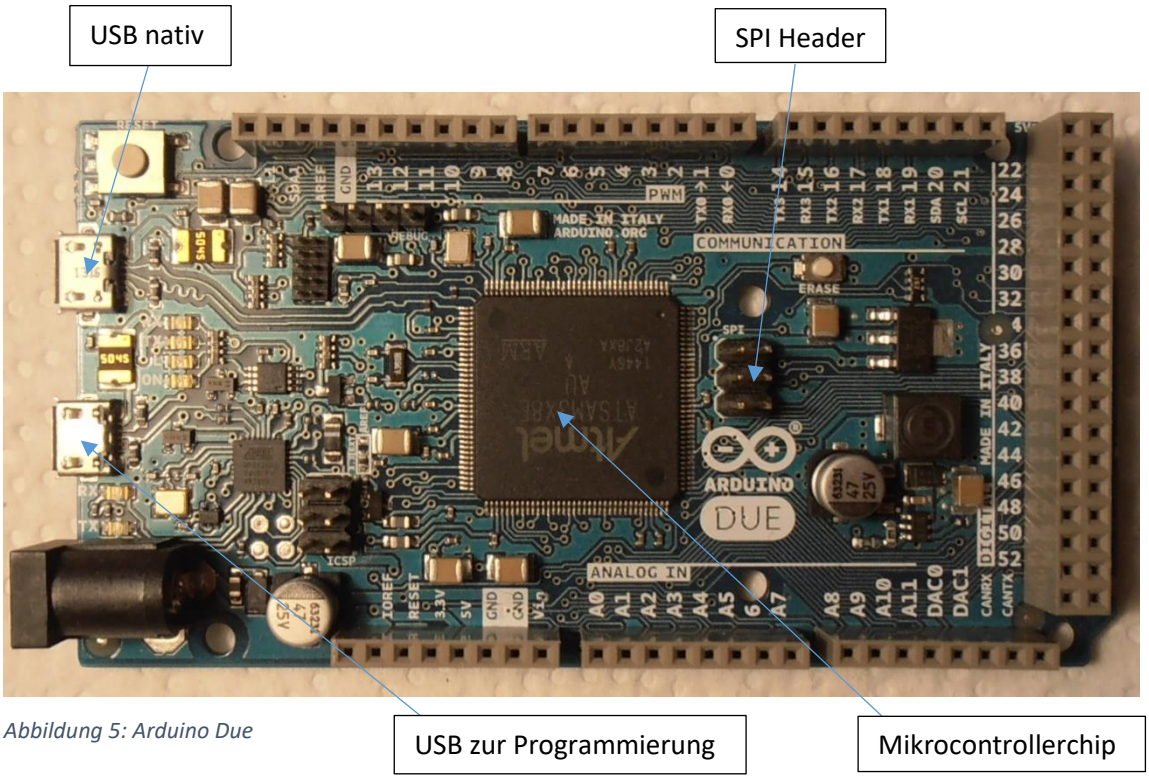


Abbildung 5: Arduino Due

Die Pins 0 – 53 sind universell einsetzbare digitale Ein-/Ausgänge, womit sich auch Modellbauservomotoren und Schrittmotoren ansteuern lassen. Jeder Anschluss kann laut seiner Spezifikation maximal 15 mA im „HIGH“-Zustand (3,3 V) liefern und 9 mA im „LOW“-Zustand (Ground) aufnehmen. Einige dieser digitalen Pins sind auch noch für andere Funktionen vorgesehen

und bieten somit insgesamt vier serielle Schnittstellen, zwei I²C-Bussysteme und 13 PWM-Pins. Weitergehend bietet das Arduino Due im wesentlichem einen SPI-Bus Header, 12 analog-zu-digital-Konverter mit 12 bit Auflösung, zwei digital-zu-analog-Konverter, GND-Pins, einen 3,3 V Ausgang und einen 5 V Ausgang.
Der Mikrocontroller kann entweder mit einem Hohlbuchsenstecker oder direkt über USB mit Spannung versorgt werden, von denen er zwei hat. Ein USB-Anschluss ist für die Programmierung vorgesehen. Dieser USB-Anschluss ist mit einem Atmega 16U verbunden, der als USB-Host fungiert. Dieser Chip kommuniziert wiederum mit einem seriellen Bus (Serial 0) mit dem Hauptprozessor. Der andere USB-Anschluss wird nativ vom Hauptprozessor bereitgestellt und ist dafür vorgesehen, das Arduino Due an anderen Geräten als HID-Gerät (Maus oder Tastatur) verwenden zu können. Zum Betreiben des Tabledance muss der USB-Anschluss, der für die Programmierung vorgesehen ist, verwendet werden. Dieser ist im gezeigten Bild vom Arduino Due dementsprechend beschriftet.

4.1.2 Touchscreen

Das Touchscreen stellt für das Tabledance ein wichtiges Bauteil dar, weshalb auch bei dieser Komponente besondere Aufmerksamkeit galt. Nach ausgiebigen Recherchen und Abwägungen wurde ein 17 Zoll großes, resistives 5-Draht-Touchscreen von AMT im Format 5:4 erworben, welches anschließend genauer untersucht werden konnte. Dies war nötig weil die Herstellerangaben zu den Touchscreens nicht ausreichend sind. Wie vom Hersteller beworben, ist dieses Touchscreen sehr haltbar, was unter anderem durch die für resistive Touchscreens ungewöhnlich hohe Resistenz gegen Kratzer bestätigt werden kann. Die hochwertige 5-Draht-Technik verleiht dem Touchscreen neben Haltbarkeit eine hohe Linearität, die für die korrekte Abbildung der tatsächlichen Position wichtig ist und auch für den Betrieb mit dem Arduino Due vom Vorteil ist. Diesbezüglich sei erwähnt, dass neben der 5-Draht-Technik noch die 4-Draht-Bauform und die davon abstammenden Sonderbauformen, wie z.B. 8-Draht, existieren. Die 4-Draht-Bauform ist weit verbreitet und günstiger als die 5-Draht-Bauform, wohingegen diese bei den wichtigen Eigenschaften Linearität, Genauigkeit und Haltbarkeit wesentlich besser ist. Wie der Name schon aussagt, werden bei der Ansteuerung fünf Drähte bei 5-Draht-Touchscreens und vier Drähte bei 4-Draht-Touchscreens benötigt. Bei 4-Draht-Touchscreens sind pro Schicht jeweils zwei Drähte an den gegenüberliegenden Kanten angeschlossen. Zur Messung der Position in einer Dimension wird an der ersten Schicht eine Gleichspannung angelegt, wobei die Spannung aufgrund des Wiederstands von einer Kante der Schicht zur gegenüberliegenden Kante gleichmäßig abfällt. Ausgehend von der zweiten Schicht

können zwei verschieden Spannungen zur ersten Schicht gemessen werden, womit sich die Position als Verhältnis der Spannungen einfach berechnen lässt.

Für die Bestimmung der zweidimensionalen Position werden zwei Messungen mit genau vertauschten Rollen der Anschlüsse benötigt. In Abbildung 6 ist dazu das Prinzip von 4-Draht-Touchscreens als Skizze dargestellt. Nachteilig an dieser Technik ist, dass beide Schichten zur Messung verwendet werden. Die weiche obere Schicht wird durch Benutzung stark beansprucht, wodurch die Gleichmäßigkeit der leitenden Beschichtung beeinträchtigt und nach längerer Benutzung zerstört wird. Die Gleichmäßigkeit dieser Beschichtung ist bei dieser Technik ein Maß für die Genauigkeit des Touchscreens. Dies erklärt, warum 4-Draht-Touchscreens schnell ungenau werden und nicht sehr haltbar sind. Eine große Verbesserung stellt die hier verwendete und qualitativ hochwertigere 5-Draht-Technik dar, bei der nur die auf Glas aufgebrachte Beschichtung der haltbareren unteren Schicht als positionsabbildende Widerstandsstrecke verwendet wird. Die mit nur einem Draht verbundene obere Schicht stellt lediglich eine niederohmige Verbindung zum Druckpunkt bereit, wodurch diese Schicht nicht als Maß für die Bestimmung der Position verwendet wird. Mit diesen zusätzlichen 5. Draht (Sensorleitung) der oberen Schicht kann die Position der Druckstelle in beiden Dimensionen bestimmt werden, weswegen nur noch ein A/D-Wandler für die Messung notwendig ist. Die untere Schicht kann man sich so vorstellen, dass die übrigen vier Drähte jeweils an den Ecken angeschlossen sind und durch korrekte Beschaltung jeweils abwechselnd ein gleichmäßig steigendes Spannungsfeld von links nach rechts und von unten nach oben erzeugen. Durch das jeweilige Messen der Spannung am 5. Draht kann ganz einfach auf die zweidimensionale Position geschlossen werden. Die Abbildung 7 zeigt dazu eine Prinzipskizze der 5-Draht-Technik.

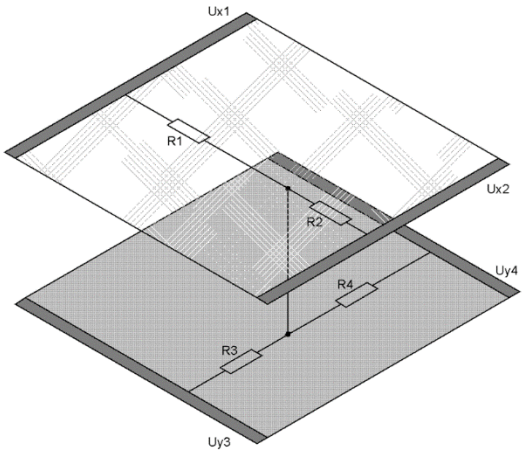


Abbildung 6: Funktionsprinzip 4-Draht-Touchscreen, Quelle [3]

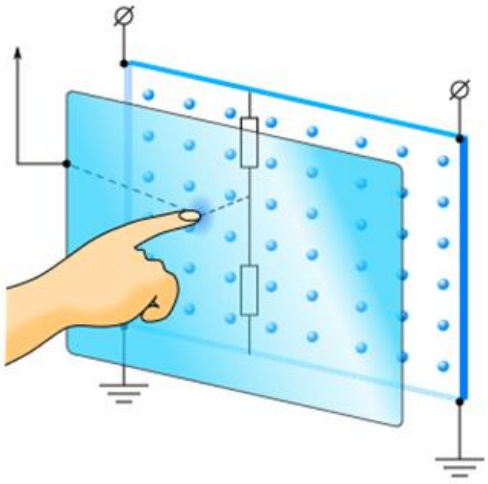


Abbildung 7: Funktionsprinzip 5-Draht-Touchscreen, Quelle [3]

Um bestmögliche Linearität bei hochwertigen Touchscreens zu gewährleisten, sind die Drähte in Wirklichkeit in spezieller Weise am Rand der leitenden Schicht angeschlossen. Dies ist in der

Abbildung 8 zu erkennen, bei dem ein kleiner Abschnitt vom Rand des erworbenen Touchscreens dargestellt ist.

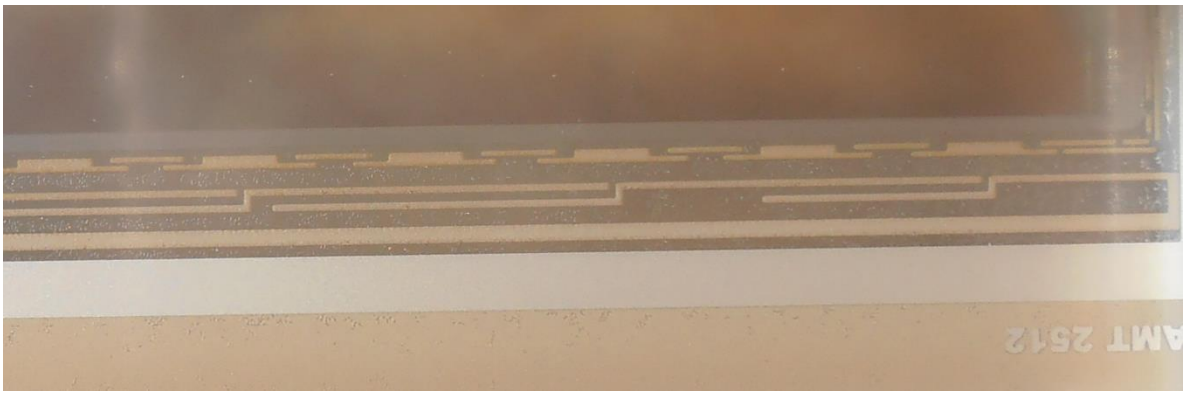


Abbildung 8: Rand eines 5-Draht-Touchscreens

Das AMT-Touchscreen konnte direkt mit dem Arduino Due in Betrieb genommen werden, wobei die logische Beschaltung der Anschlüsse zu beachten war. Für die Ansteuerung der vier Drähte der unteren Schicht wurden digitale Pins vom Arduino verwendet, wovon jeweils zur Messung einer Koordinate zwei auf 3,3 V und zwei auf Ground geschaltet sein müssen. Die Spannung am 5. Draht wurde mit dem integrierten 12 bit A/D-Wandler gemessen, wobei die gemessene Spannung proportional zur Position ist. Damit konnte die generelle Funktion zwischen dem Touchscreen und dem Arduino Due sichergestellt werden, welche im Laufe der Arbeit verbessert und den Bedürfnissen entsprechend angepasst wurde. Auch konnten einige Eigenschaften des Touchscreens definiert werden, die vom Hersteller selbst nicht angegeben wurden. Darunter zählt, dass das Touchscreen bei einzelnen Fingerdrücken schon ab ca. 0,5 N reagiert, wohingegen für einen durchgängigen Kontakt der Schichten bei einer abrollenden Kugel 3,5 N nötig waren. Das liegt daran, dass die Kugel nur auf einer sehr kleinen Fläche auf dem Touchscreen aufliegt und beim Abrollen gelegentlich direkt über einen Abstandshalter der zwei Schichten rollt. Ist das Gewicht der Kugel zu gering, kontakten die Schichten in solchen Situationen nicht mehr zuverlässig, was sich durch Aussetzer der Messung zeigt. Die Abstandshalter sind schwach sichtbar und in einem Raster mit 5 mm Abstand angeordnet. Aufgrund dieser Abstandshalter musste eine ca. 350 g schwere Stahlkugel mit 45 mm Durchmesser verwendet werden, welche für die Stelleinheit eine hohe Belastung darstellt und bei der Auslegung berücksichtigt werden musste.

4.1.3 Antriebseinheit

Die Stelleinheit wurde, ähnlich wie am Vorgängersystem, mit Modellbauservomotoren realisiert. Als Grundlage der Auslegung wurde eine überschlägige Berechnung in Bezug auf das benötigte Drehmoment aufgestellt. Dazu musste die theoretisch höchste Belastung an der Y-Achse betrachtet werden, die aufgrund des Motors an der X-Achse stärker belastet ist. Da die Aufhängung des Tabledance ansonsten weitgehend symmetrisch konstruiert wurde, musste im stationären Fall nur die Kugel und der Motor mit dem Getriebe als statische Last betrachtet werden, das bei ca. 1,2 Nm liegt. Dies ist das Mindestdrehmoment bei dem noch keine kinematischen Effekte wie z.B. Trägheit und Beschleunigung berücksichtigt sind. Außerdem gilt insbesondere bei Modellbauservomotoren, dass das System bei Überdimensionierung schneller und genauer bei der Schaftpositionsregelung ist. Nach einigen Tests mit Servomotoren konnte auch festgestellt werden, dass die Abweichung der Schaftposition bei einer Drehmomentbelastung der spezifizierten Maximalbelastung für den anberaumten Einsatz bereits signifikant hoch ist.
In ungünstigen Situationen kann das System zudem durch die Trägheit des beweglichen Aufbaus um das mehrfache belastet werden, wodurch sich bei zu schwacher Dimensionierung Schwingungen aufbauen können, die das System instabil werden lassen. Deshalb wurde, obwohl ein Servomotor alleine schon über 3 Nm Drehmoment bereitstellen kann, ein zusätzliches Getriebe eingeplant. Dieses Getriebe entlastet auch das Zahnradgetriebe der Modellbauservomotoren, wodurch eine längere Haltbarkeit dieser Motoren zu erwarten ist.
Modellbauservomotoren gibt es von einigen Herstellern in vielen verschiedenen Ausführungen, wobei der Großteil in einer einheitlichen Standardgröße von 20 x 40 mm Grundfläche gehalten sind. Diese Motoren werden vorrangig für Stellaufgaben im Hobbybereich eingesetzt, weswegen der Stellbereich von diesen Motoren anwendungsbedingt auf meistens 180° begrenzt ist. Angesteuert werden diese Motoren mit einem PWM-ähnlichen Impulssignal mit einer Frequenz von 50 Hz, das vom Arduino Due über ein digitalen Pin bereitgestellt werden kann. Mit einer Impulslänge von 1500 µs fährt der Servomotor auf die Mittelposition, wogegen gewöhnlich Impulslängen von 800 µs bis 2200 µs den gesamten anfahrbaren Bereich eines Servomotors abdecken. Manche Servomotoren können auch mit höheren Frequenzen betrieben werden, wobei 333 Hz die Obergrenze darstellt. Bei noch höheren Frequenzen würden sich die einzelnen Impulse überlappen und könnten von der Elektronik im Servomotor nicht mehr sinnvoll ausgewertet werden.
Neben wenigen analogen gibt es mittlerweile viele digitale Servomotoren, wobei sich die Bezeichnung für analog und digital auf die integrierte Regelungseinheit bezieht. Digitale Servomotoren sind schneller und genauer bei der Schaftpositionsregelung und können kleinen Abweichungen stärker entgegenregeln. Analoge Servomotoren haben regelungsbedingt hingegen

eine lineare Kennlinie zwischen Abweichung der Schaftposition und dem entgegengestellten Drehmoment, wodurch die Stellgenauigkeit unter Last stärker beeinträchtigt wird. Im Bereich der Servomotoren mit hohem Drehmoment sind zudem nur noch digitale Varianten verfügbar, sodass nur digitale Servomotoren für das neue Tabledance verwendet werden konnten. Der Nachteil der digitalen Bauweise ist die viel höhere Lärmentwicklung unter statischer Last. Dies ist auf die taktbasierte Ansteuerung des Motors durch die integrierte Regelungselektronik zurückzuführen, wodurch der Motor in hochfrequente Schwingungen versetzt wird und dementsprechende Geräusche verursacht. Neben digital und analog unterscheiden sich Servomotoren unter anderem noch vom eingebauten Motortyp, wovon die verschleißfreien bürstenlosen Motoren und Glockenankermotoren mittlerweile einen guten Standard gegenüber Bürstenmotoren darstellen. Der Glockenankermotor ist ähnlich dem bürstenlosen Motor aufgebaut, nur ohne Ferritkern, wodurch er weniger Trägheitsmasse besitzt und dementsprechend schneller in der Drehbeschleunigung ist.
Dabei gilt, dass Modellbauservomotoren mit hohem Drehmoment (Drehmoment: 3 Nm, Schaftgeschwindigkeit: 0,14 sec/60° typisch) tendenziell eine größere Untersetzung vom eingebauten Getriebe haben und dadurch langsamer in der Schaftgeschwindigkeit sind als jene mit geringem Drehmoment (z.B. Drehmoment: 1 Nm, Schaftgeschwindigkeit: 0,08 sec/60° typisch).
Letztlich wurde das leistungsfähige Modell SV-1270TG mit Glockenankermotor von der Firma Savöx ausgesucht, wobei das TG für Titangetriebe steht. Titangetriebe werden aufgrund des geringen Gewichts eingesetzt, wodurch der gesamte Servomotor nur 56g wiegt. Dieser Modellbauservomotor kann mit einer Spannung von 4,8 bis 7,4 V betrieben werden und stellt laut Datenblatt bei 7,4 V 3,5 Nm Drehmoment bei einer Schaftgeschwindigkeit von 0,11 sec/60° bereit. Die Leistung ist bei geringeren Spannungen überproportional kleiner. Für die Auswahl war neben der Leistung und dem hochwertigen Metallgetriebe noch die Möglichkeit wichtig, ein Impulssignal bis 333 Hz verwenden zu können, sodass eine sehr schnelle Regelung realisiert werden kann. Die maximale Stellwertauflösung des Servomotors ist mit 12 Bit sehr hoch, die reale Auflösung ist jedoch noch maßgeblich von der Qualität des Impulssignals abhängig. Hierbei sei noch zu erwähnen, dass die Schaftposition des Servomotors intern über ein Drehpotentiometer erfasst wird. Das Drehpotentiometer arbeitet hierbei als Absolutwertgeber, weswegen diese Modellbauservomotoren nicht kalibriert werden müssen. Mit der Abbildung 9 ist ein Foto des verwendeten Modells dargestellt. Die nebenstehende Abbildung 10 zeigt die mehrstufige Getriebeeinheit des Servomotors.

Abbildung 9: Modellbauservomotor Savöx SV-1270

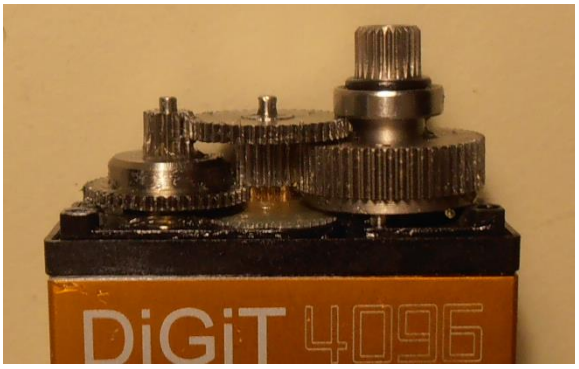


Abbildung 10: Getriebeeinheit vom Savöx SV-1270

Passend zu den Leistungsdaten des Servomotors musste eine zusätzliche Getriebestufe ausgesucht und ausgelegt werden, wobei vor allem die beengten Platzverhältnisse problematisch waren. Viele Getriebearten stellten sich in dieser Hinsicht als unpraktisch oder unbrauchbar heraus. Neben der Eignung für hohe Drehmomente und geringem Gewicht musste das Getriebe zumindest spielarm sein, sodass das ohnehin vorhandene Spiel des Servomotorgetriebes nicht noch vergrößert wird. Wie schon in der vorhergehenden Projektarbeit festgestellt wurde, wirkt sich das Spiel im Getriebe für die Stabilität und Genauigkeit der Regelung negativ aus und sollte deshalb so klein wie möglich gehalten werden. Nach ausgiebigen Recherchen konnte sich für ein einstufiges Zahnriemengetriebe entschieden werden, dass trotz der Eignung für Stellaufgaben nicht komplett spielfrei ist. Zahnriemengetriebe werden in vielen verschiedenen Ausführungen mit unterschiedlichen Zahnriemenprofilen, Zahnteilungen und Materialien angeboten, die unterschiedliche Vor- und Nachteile bieten und teilweise herstellerspezifisch sind. Weit verbreitet ist das T-Zahnriemenprofil, das in Abbildung 11 schematisch dargestellt ist. Dieses ist preiswert und wird gerne für die Übertragung von höheren Lasten eingesetzt, gehört aber nicht zu den spielarmen Profilen, weswegen es für die vorliegende Aufgabe ungeeignet war.

Abbildung 11: T-Profil, Quelle [4]

Für die Umsetzung der Getriebeeinheit wurde ein spielarmes Zahnriemengetriebe mit HDT-Profil ausgesucht, wobei die Zahnriemenräder aus leichtem Aluminium gefertigt sind. Die dazugehörigen Zahnriemen bestehen aus haltbaren Neopren mit einem Zugstrang aus Glasfasern. Diese sind nicht so dehnungsarm wie Zahnriemen mit Stahlzugstrang, haben in diesem Fall durch die Verwendung einer geringen Riemenlänge auf das Getriebespiel kaum Einfluss. In Abbildung 12 ist das HDT-Profil schematisch dargestellt, dessen abgerundetes Zahnprofil gut für spielarme Antriebe geeignet ist.

Abbildung 12: HDT-Profil Quelle[4]

Gewöhnlich werden bei Zahnriemengetrieben neben den Antriebs- und Abtriebszahnriemenräder auch Umlenkrollen verbaut, die für die Vorspannung des Zahnriemens genutzt werden können. Aus Platzgründen wurde auf die Umlenkrollen verzichtet, was aber bedeutete, dass bei der Konstruktion die Vorspannung des Riemens über spannbare Zahnriemenradwellen ermöglicht werden musste.
Das Zahnriemengetriebe wurde mit einer vom Anbieter bereitgestellten und speziell auf die angebotenen Zahnriemengetriebe zugeschnittenen Software ausgelegt und angepasst. Zu diesem Programm ist in Abbildung 13 ein Ausschnitt gezeigt, bei dem die Anordnung und Abmaße des für die X-Achse vorgesehenen Zahnriemengetriebes auf der Grafik zu erkennen ist.

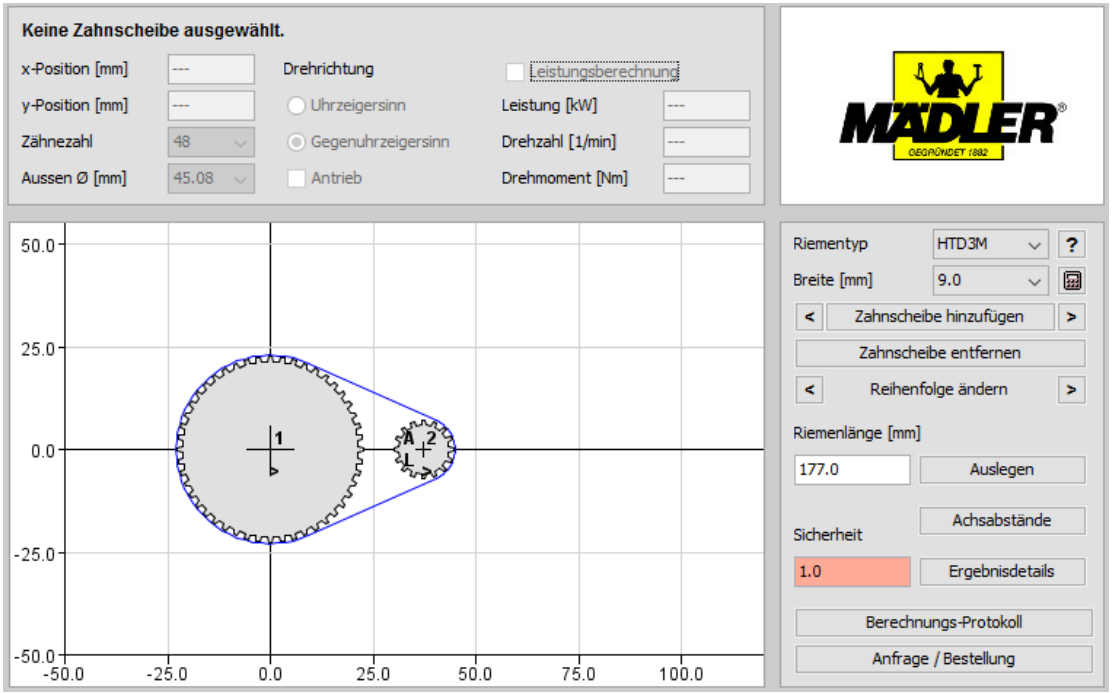


Abbildung 13: Auslegungsprogramm für Zahnriemenantriebe

Für die X-Achse wurde ein Antriebszahnriemenrad mit 16 Zähnen und Abtriebszahnriemenrad mit 48 Zähnen vorgesehen, was eine Untersetzung von drei ergibt. Mit 3,5 Nm Motordrehmoment ergibt sich ein maximales Drehmoment von 10,5 Nm. Mit einem so hohen Drehmoment ist eine schnelle und vor allem genaue Winkelpositionierung bei normaler Belastung möglich. Passend dazu wurde ein Zahnriemen in der Standardlänge von 177 mm bei einer Breite von 9 mm ausgesucht, wodurch sich ein Achsabstand der Zahnriemenradwellen von 37 mm berechnet.
Das Zahnriemengetriebe für die Y-Achse wurde mit einer Untersetzung von 3,75 etwas stärker ausgelegt. Begründet ist dies mit der erhöhten Belastung dieser Achse gegenüber der X-Achse und den besseren Platzverhältnissen, die ein größeres Getriebe zulassen und das Gesamtsystem trotzdem kompakt bleibt. Ausgesucht wurde dabei ein Abtriebszahnriemenrad mit 60 Zähnen bei

gleichbleibendem Antriebszahnriemenrad. Mit einem 210 mm langen Zahnriemen bei einem Achsabstand von 42,7 mm konnte das Getriebesystem dennoch relativ kompakt gehalten werden. Der vom Programm berechnete Sicherheitsfaktor für die vorliegende Belastungssituation beläuft sich bei der Auslegung beider Getriebe auf 1,0, wonach keine Sicherheitsreserven vorhanden wären. Mit dem Hintergrund, dass das Programm keine niedrigen Drehzahlen, die der normalen Belastungssituation vom Tabledance entsprechen würden, verarbeiten kann, ist die Sicherheit dennoch als ausreichend anzusehen. Andernfalls hätten breitere Riemen und Riemenräder verwendet werden müssen, wodurch das System an Kompaktheit verlieren würde.

4.2 Mechanische Konstruktion und Montage

Nachdem die wichtigsten Komponenten für das neue Tabledance festgelegt waren, konnte davon ausgehend die mechanische Konstruktion erstellt werden. Für die Umsetzung der Konstruktion wurde das CAD-Programm Solid Edge ST8 benutzt.
Als Konstruktionsmaterial wurde weitgehend Aluminium mit der Legierung AlMgSi0,5 und einer Streckgrenze von 220 N/mm² verwendet. Vorteilhalft bei Aluminium ist die Rostfreiheit und das geringe Gewicht gegenüber Stahl. Dem steht die viel geringere mechanische Belastbarkeit gegenüber. Da die Konstruktion des Tabledance an den meisten Stellen mechanisch gering beansprucht ist, konnte Aluminium sehr gut für die meisten Bauteile verwendet werden. Mechanisch kritische Stellen wurden mit Berechnungen und FEM-Simulationen überprüft, die in diesem Zusammenhang nicht weiter erläutert werden. Als Fertigungsverfahren wurde hauptsächlich Fräsen verwendet, wobei die gute Zerspanbarkeit von Aluminium das Fräsen erleichterte. Edelstahl zum Beispiel ist sehr zäh und ungeeigneter zum Fräsen. Die gute Zerspanbarkeit war insofern vorteilhaft, weil die Bauteile aus Kostengründen mit möglichst wenig Aufwand gefertigt werden sollten. Auch die Konstruktion der einzelnen Bauteile musste auf die Fräsfertigung ausgelegt werden, wobei die eigentlichen Funktionen nicht vernachlässigt werden durften. Um das Herstellen zu vereinfachen, wurden bevorzugt Normteile und allgemeine Maße verwendet.
Die Abbildung 14 zeigt das konstruierte CAD-Modell des Tabledance in der Draufsicht, wohingegen die darauf folgende Abbildung 15 eine isometrische Ansicht des Modells darstellt.

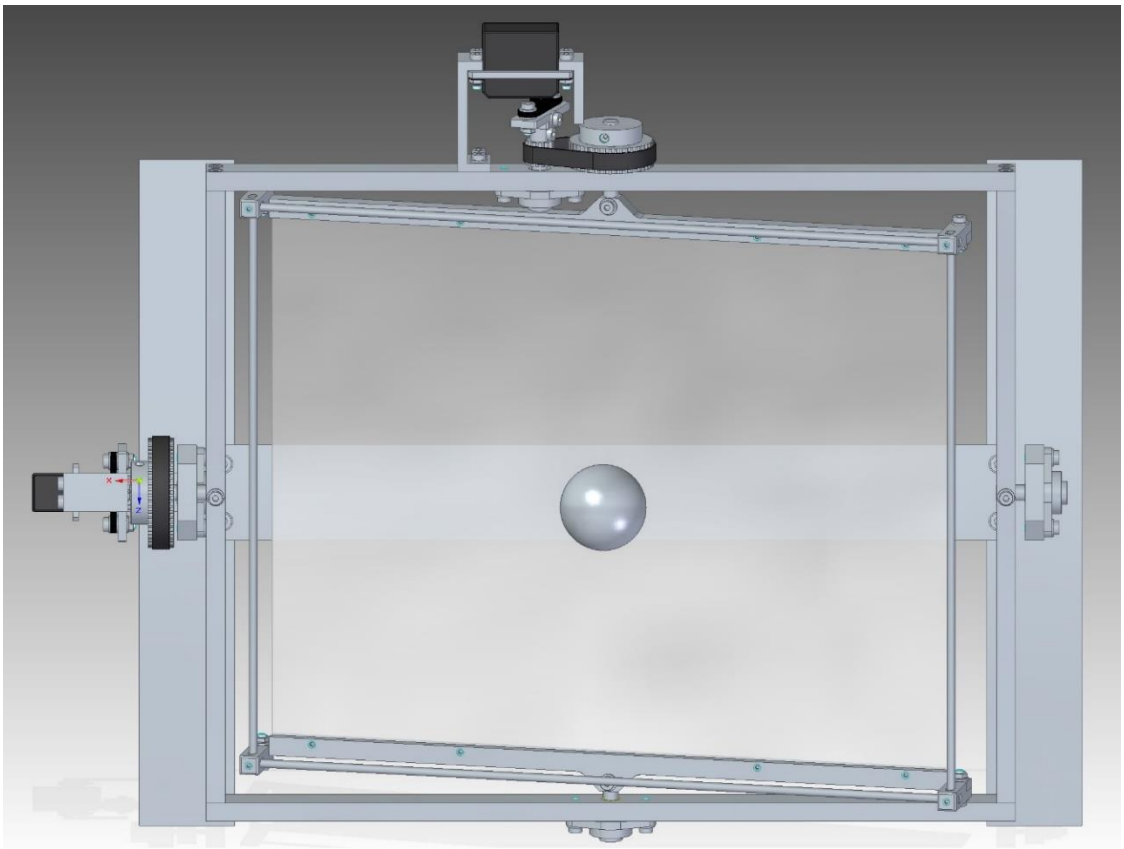


Abbildung 14: Draufsicht vom CAD-Modell

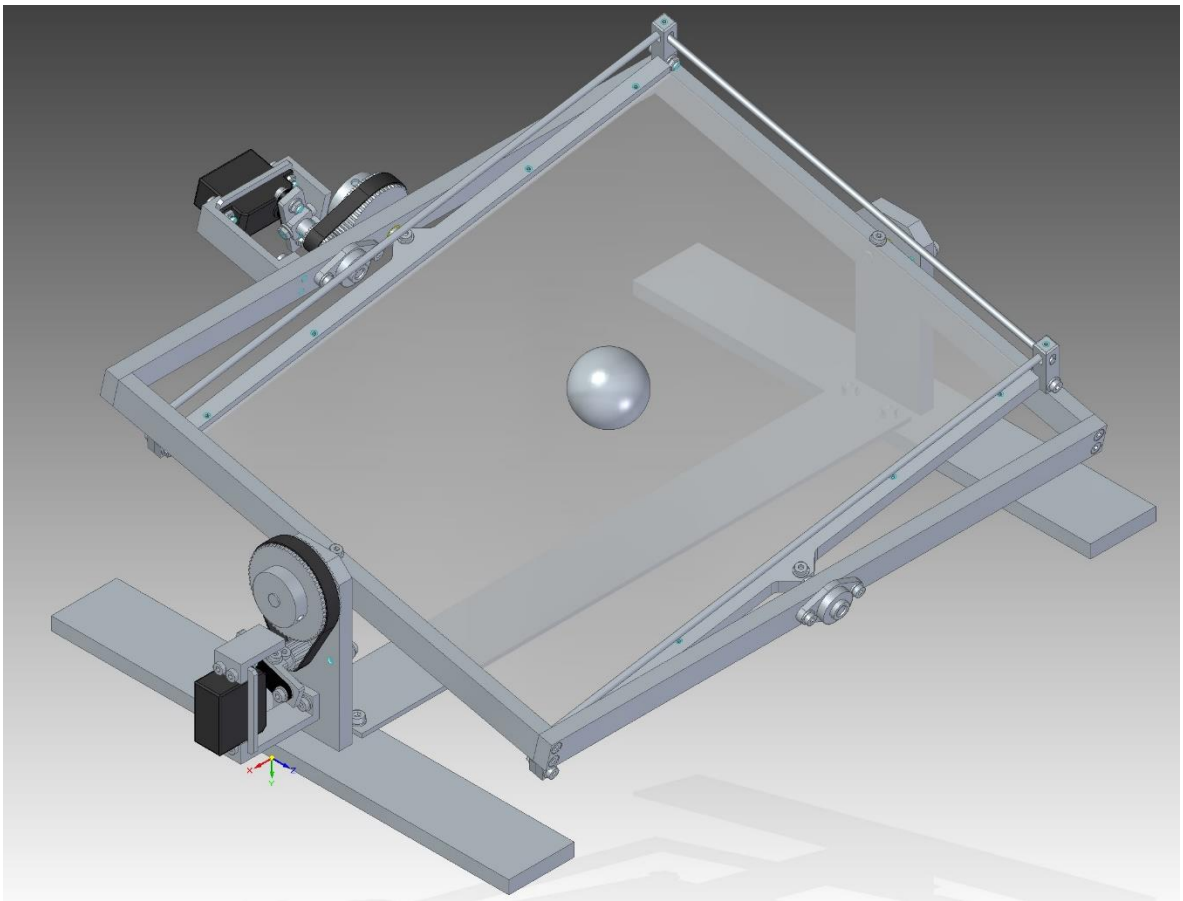


Abbildung 15: Isometrische Ansicht des CAD-Modells

Das CAD-Modell wurde vom Touchscreen ausgehend konstruiert, da dieses maßgeblich die Größe des Systems beeinflusst. Wie auf den Abbildungen zu erkennen ist, wurde das 3 mm dicke Touchscreen mit zwei Nutschienen mechanisch angebunden. Senkrecht zur Nut sind je Schiene von oben vier Madenschrauben angebracht, die das Touchscreen in der Nut fixieren. Das ist in dieser Weise nur möglich, weil das Touchscreen an den Randbereichen nicht berührungssensitiv ist und deshalb trotz Anpressdruck keine Verfälschung der gemessenen Position hervorgerufen wird.
Zum Touchscreen wurde ein passender Zaun entworfen, der die Kugel vor dem herunterfallen abhält. Dazu wurden an den Ecken des Touchscreens Aluminiumvierkantstangen mit Langlochdurchführungen entworfen. Vier stabile 4 mm dicke Rundstangen aus 115CrV3 bilden den eigentlichen Zaun und sind in den Langlöchern mit Madenschrauben von oben fixiert.
Das Touchscreen wurde mit Kunststoffgleitlagerbuchsen in Verbindung mit 8 mm dicken Welle ebenfalls aus 115CrV3 an einen Rahmen aus Aluminiumvierkantstangen beweglich gelagert. Die standardmäßig erwerbbare Welle im Toleranzmaß 8 h9 ist auf die Gleitlagerbuchsen, die im eingepressten Zustand das Toleranzmaß 8 H8 aufzeigen, als Spielpassung ausgelegt, wobei die Bohrung im Toleranzmaß 10 H8 gefertigt werden musste. Die Gleitlagerbuchsen konnten mithilfe einer vom Hersteller Igus bereitgestellten Software ausgewählt und ausgelegt werden. Ausgewählt wurden welche mit dem Kunststoff J3, die laut Berechnungen mit den veranschlagten Lasten ca. 100000 Stunden im Dauerbetrieb halten. Neben der Kunststoffart und der Belastungsstärke sind auch die Belastungsart und der verwendete Wellenwerkstoff maßgeblich für die Lebensdauer der Kunststoffgleitlager ausschlaggebend. Die Gleitlager können nur radiale Lasten aufnehmen, weswegen für die axialen Lasten Flanschkugellager eingesetzt wurden, die mittels Madenschrauben an der Welle befestigt sind und das Touchscreen gegenüber axiale Lasten sichern.
Für die Befestigung von Elementen an den Wellen wurde meistens eine M4 Schraubverbindung verwendet. Diese ist gegenüber anderen Befestigungsmechanismen leicht herzustellen und für die hier auftretenden Drehmomente ausreichend. Die Abtriebszahnriemenräder wurden mit M5 Madenschrauben auf den Wellen fixiert, während bei den Antriebsrädern hingegen eine M4 Schraubverbindung verwendet wurde. Dies hat den einfachen Grund, dass die kleinen Zahnriemenräder nicht genügend Durchmesser für eine Gewindebohrung aufweisen und die hierbei üblicherweise verwendeten Spannsätze nicht passen. Wichtig ist hierbei, dass nach der Montage der Nullpunkt vom Servomotor zum Touchscreen über den an der Welle flexibel feststellbaren Madenschrauben an den Abtriebzahnriemenrädern eingestellt werden kann. Die Antriebwellen, an den die Servomotoren gekoppelt sind, mussten für die Spannung der Zahnriemen verstellbar im Achsabstand zur Abtriebswelle konstruiert werden. Dazu wurde die Antriebswelle mit einem Flanschkugellager verschiebbar gelagert.

Modellbauservomotoren haben für den Anschluss an einer Stellmechanik sogenannte Servohebel, die für eine direkte Montage an einer Welle ungeeignet sind. Deshalb musste ein Adapter gebaut werden, der das Drehmoment vom Servohebel auf die Antriebswelle des Zahnriemengetriebes überträgt. Dieser ist sowohl am Servohebel als auch an der Antriebswelle mit Schraubverbindungen befestigt.

Als Grundgestell wurden zwei 50 x 10 mm Aluminiumflachstangen verwendet, die mit einer 50 x 5 mm Aluminiumflachstange quer verbunden sind. Die Oberseite der Flachstangen wurde auch als Platz für die benötigte Elektronik vorgesehen, wodurch das Tabledance trotz der Größe kompakt gehalten werden kann. Auch das Touchscreen wurde für geringere Kippneigung möglichst niedrig eingebaut, wobei eine ausreichende Kippfreiheit von 15° berücksichtigt wurde.

Die Bauteile wurden an der Hochschule Rosenheim vom Herrn Prexl und Herrn Schiefer gefertigt. Die Ausnahme bilden einfache Bauteile, die auch selbst gefertigt wurden. Alle dafür erstellten Fertigungszeichnungen sind im Anhang aufgeführt.

Die Abbildung 16 zeigt ein Foto vom fertig montierten Tabledance, bei dem auch bereits die Elektronik integriert ist.

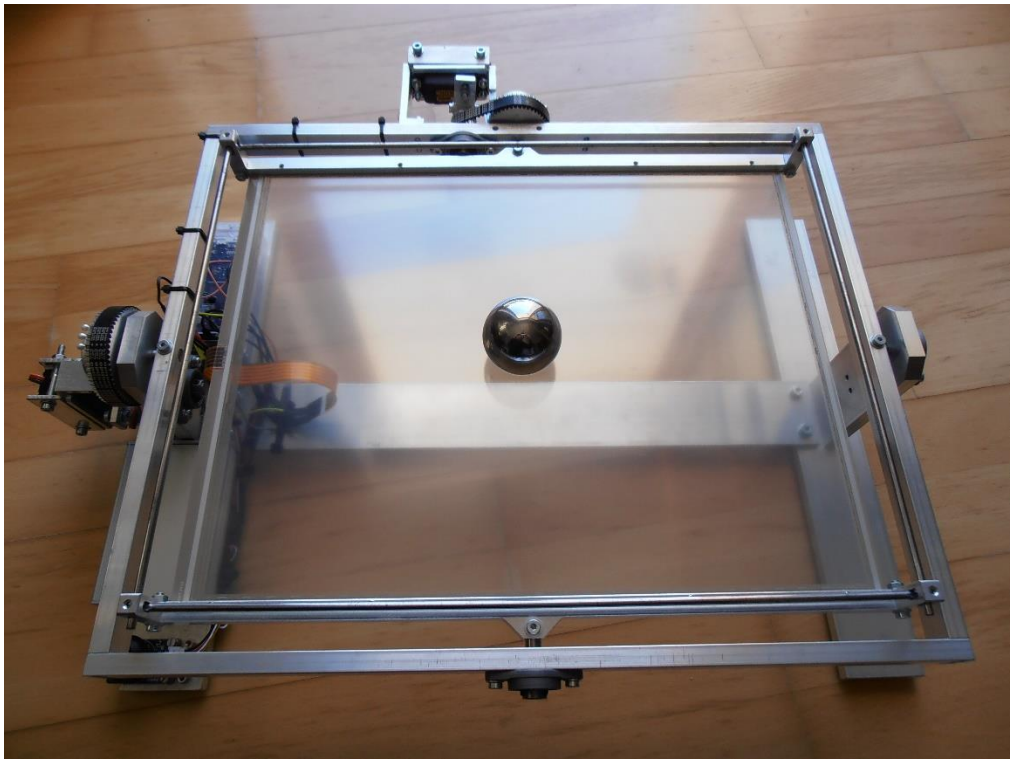


Abbildung 16: Foto des realisierten Regelungssystems

5. Hard- und softwaretechnische Umsetzung der Regelung

Neben interner Datenverarbeitung muss das zu programmierende Arduino Due vor allem Positionsmessdaten vom Touchscreen erfassen und nach Bearbeitung mit einem Regelungsalgorithmus entsprechende Stellsignale an die Servomotoren ausgeben. Dabei wurden die Positionsdatenerfassung und die Stellsignalausgabe zunächst einzeln programmiert und getestet, bevor das gesamte System zusammengestellt wurde.

5.1 Ansteuerung der Servomotoren

5.1.1 Elektronische Anbindung der Servomotoren

Die Qualität der Ansteuerung der Servomotoren beeinflusst das Regelungsverhalten des Tabledance maßgeblich, weswegen eine gute Umsetzung wichtig war. Servomotoren haben drei elektrische Anschlüsse, die farblich einheitlich codiert sind. Für die Spannungsversorgung ist der braune und rote Anschluss vorgesehen, wobei am roten Anschluss die positive Spannung und am braunen Anschluss Masse angeschlossen werden muss. Der orangene dritte Anschluss ist für das impulsförmiges Steuersignal vorgesehen, das vom Arduino Due über einen digitalen Pin ausgeben werden kann. Da die Signalausgabe nur einen digitalen Pin benötigt, können mit dem Arduino Due bis zu 54 Servomotoren gleichzeitig angesteuert werden.
Für die Spannungsversorgung der Servomotoren wurde ein leistungsstarkes und umschaltbares Steckernetzteil bereitgestellt, das über eine Hohlbuchsensteckverbindung neben dem Mikrocontroller an die Modellbauservomotoren angeschlossen werden kann. Das verstellbare Steckernetzteil ist entsprechend dem oberen Bereich der für die Modellbauservomotoren zugelassenen Spannung auf 7,5 V eingestellt. Zudem musste das Steckernetzteil zum Mikrocontroller hin geerdet werden, damit eine einbahnfreie Funktion des Impulssignals auf die Servomotoren sichergestellt ist.

5.1.2 Umsetzung der Servomotoransteuerung mit der Arduino IDE

Die Arduino IDE ist eine selbstständige Entwicklungsumgebung für alle Arduino Mikrocontroller, mit der alle Funktionen vom Mikrocontroller genutzt werden können. Diese Programmierumgebung bietet neben vielen C++-typischen Grundbefehlen einige auf bestimmte Aufgaben angepasste

Bibliotheken und Funktionen, mit denen die Programmierung des Mikrocontrollers stark vereinfacht werden kann. Diese Bibliotheken und Funktionen sind teilweise auf hardwarenahe und mikrocontrollerspezifische Befehle aufgebaut, die in den meisten Fällen aufgrund eben dieser Bibliotheken und Funktionen nicht benötigt werden. Dazu zählt auch die Ansteuerung von Modellbauservomotoren, für die es eine spezifische Bibliothek gibt.

Die Abbildung 17 zeigt die Entwicklungsumgebung mit einem einfachen Programm für die Ansteuerung eines Servomotors. Der Programmcode wird hierbei in drei funktionsspezifische Bereiche unterteilt. Der obere Bereich ist wie üblich für global gültige Befehle. Hier können z.B. Bibliotheken eingebunden und globale Variablen oder Objekte erstellt werden. Mit „#include <Servo.h>“ wurde hier die für die Servoansteuerung benötigte Bibliothek eingebunden. Diese stellt Klassen, Funktionen und Methoden zur Verfügung, wie es in C++ üblich ist. Eine Bibliothek besteht oft aus einer Headerdatei mit Endung „.h“ und einer Sourcedatei mit der Endung „.cpp“, wovon für die Bereitstellung der Bibliothek nur die Headerdatei eingebunden werden muss. Der darauf folgende Befehl „Servo Servo1“ erstellt das Objekt Servo1 von der Klasse Servo. Das Objekt Servo1 ist repräsentativ zu einem Servomotor zu sehen.

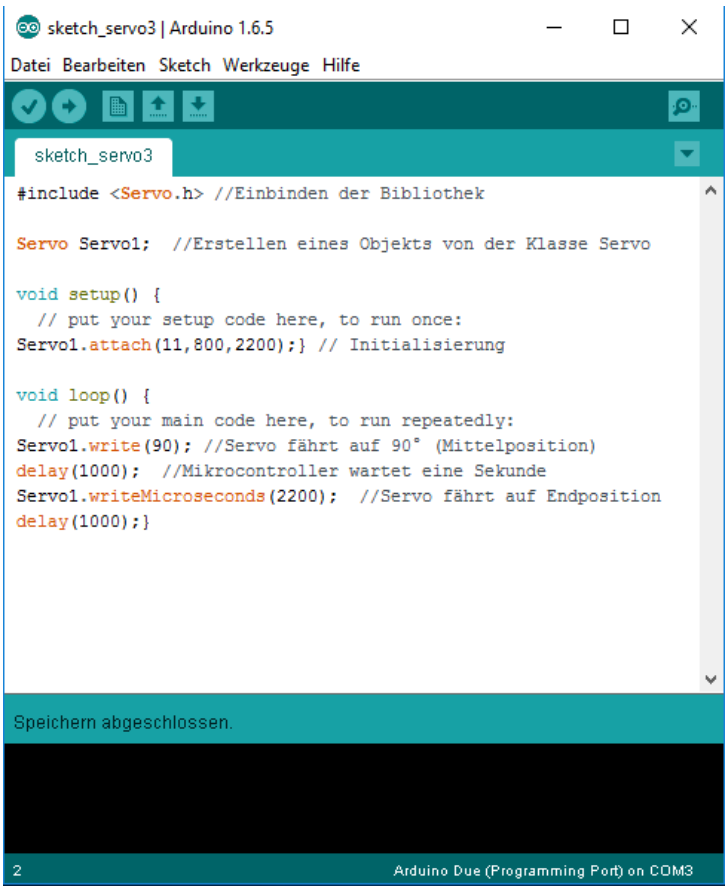


Abbildung 17: Arduino IDE mit Programm

Im mittleren Bereich, das mit der Funktion „void setup(){}“ eingeschlossen ist, können Funktionen initialisiert werden. Hier müssen alle Befehle reingeschrieben werden, die zum Programmstart nur einmal ausgeführt werden müssen. Mit der Anweisung „Servo1.attach(11, 800, 2200) wird an dem vorher erstellten Objekt „Servo1“ die Methode „attach“ aufgerufen und die Parameter übergeben, wobei 11 der Pin ist, an dem das Signal für den Servo ausgegeben wird. Die Zahlen 800 und 2200 sind die Minimal- und Maximalimpulslängen in Mikrosekunden, die den maximalen Bereich für die Ansteuerung des verwendeten Servomotors darstellen. Bei einigen anderen Servomotoren ist ein Bereich zwischen 1000 und 2000 Mikrosekunden üblich.

Der untere Bereich des Programms, der in der Funktion „void Loop(){}“ eingeschlossen ist, stellt das eigentliche Hauptprogramm dar. Dieses wird vom Mikrocontroller in einer Endlosschleife ausgeführt. Darin ist z.B. der Regelungsalgorithmus zu implementieren, der dauerhaft wiederholt werden muss. Für die Ansteuerung der Position des Servomotors stellt die Bibliothek zwei Methoden zur Verfügung. Mit der Methode „write()“ kann die Schaftposition des Servomotors zwischen 0° und 180° angesteuert werden. Diese Methode erwartet einen Integerwert, weswegen nur ganze Zahlen verwendet werden können und die Auflösung der Schaftposition mit 180 Schritten gering ist. Verbesserung wird durch die alternative Methode „writeMicroseconds()“ erreicht, die als Parameter ganzzahlige Impulslängen in Mikrosekunden entgegennimmt. Durch die mikrosekundengenau anpassbare Impulslänge ist eine extrem genaue Schaftpositionsansteuerung möglich. Im Stellbereich von 180° sind damit 1400 Positionen anfahrbar, womit sich eine Stellwinkelauflösung von 0,13° ohne Berücksichtigung der Zahnriemengetriebe ergibt. Durch die Zahnriemengetriebe kann annähernd der ganze Stellbereich der Servomotoren effektiv genutzt werden, wodurch die effektive nutzbare Auflösung fast so groß ist wie die absolute Auflösung im gesamten Stellbereich der Servomotoren. Mit der Befehl „delay()“ kann der Mikrocontroller eine dem Parameter entsprechende Zeit in Millisekunden verzögert werden, bis das Programm weiter ausgeführt wird. Mit dem kleinen Programm fährt der Servomotor wiederholend auf Mittelposition (90°) und mit einer Sekunde Abstand auf Endposition (2200 µs Impulslänge). Mit klicken auf Sketch=>Hochladen kann das Programm kompiliert und über die USB-Verbindung zum Arduino übertragen werden. Dieser führt den Code direkt danach dauerhaft aus, bis entweder die Spannungsversorgung abgesteckt wird oder ein neues Programm hochgeladen wird. Nachdem die Ansteuerung der Servomotoren erfolgreich mit der Arduino Entwicklungsumgebung programmiert und getestet werden konnte, wurde diese in Simulink implementiert.

5.1.3 Umsetzung der Servoansteuerung in Simulink mithilfe einer S-Funktion

Damit der Mikrocontroller mit Simulink programmiert werden kann, muss das entsprechende „Hardware Support Package“ installiert werden. Dieses ist im Matlabfenster unter „Addons=>Get Hardware Support Package=>Simulink Support Package for Arduino Hardware“ zu finden.
Mit der Installation der „Hardware Support Packages“ vom Arduino Due wird eine eigene Arduino-Toolbox in Simulink hinzugefügt. Diese stellt grafische Blöcke für die Verwendung einiger Funktionen des Mikrocontrollers bereit. Ein Teil der Toolbox ist in der Abbildung 18 dargestellt. Wie darauf zu erkennen ist, wird unter anderem neben Digital In, Digital Out, Analog In auch ServoWrite unterstützt. Des Weiteren können fast alle sonstigen Blöcke, die in Simulink verfügbar sind, mit dem

Arduino Due verwendet werden. Der Block „ServoWrite“ ist für die Ansteuerung von Modellbauservomotoren geeignet und hat als Parameter die Pinnummer, bei der das Impulssignal ausgegeben werden soll. Dieser Block nimmt jedoch nur ganzzahlige Werte von 0 – 180 entgegen und benutzt somit die beschriebene Methode „write“ der Arduino IDE. Diese ist wie beschrieben viel ungenauer als die alternative Methode „writeMicroseconds“, welche nicht implementiert wurde. Der „ServoWrite“-Block ist demnach nicht gut für diese Regelungsaufgabe geeignet und kann mit der Verwendung von vorgefertigten Blöcken nicht verbessert werden, was insofern einen Nachteil der unflexiblen blockgebundenen Programmierung gegenüber konventionellen Hochsprachen darstellt.

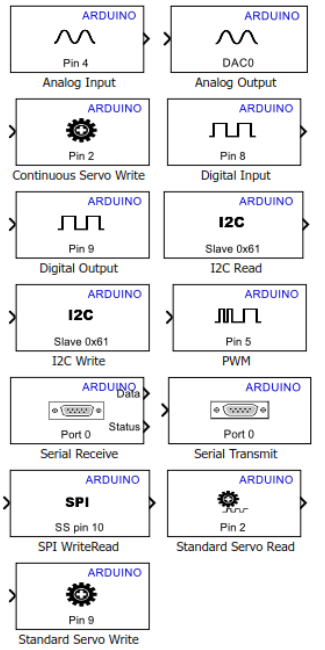


Abbildung 18: Arduino-Toolbox

Können Aufgabenstellungen in Simulink nicht durch die angebotenen Blöcke zufriedenstellend gelöst werden, stellt Simulink mehrere verschiedene Alternativen zur Verfügung, um eigenen Code in Simulink zu implementieren. Dabei unterstützt Simulink einige Programmiersprachen wie z.B. C/C++, Fortran und das eigene Matlab. Dazu muss jedoch ein entsprechender Compiler installiert und in Simulink eingerichtet sein. Hierbei sei zu erwähnen, dass in dieser Arbeit für die Implementierung von C/C++-Code ausschließlich sogenannte S-Funktionen und als Compiler der C++-Compiler von Visual Studio 2010 verwendet wurde. Zum Betreiben des Tabledance muss ein C++-Compiler als Standardcompiler in Matlab mit „mex –Setup“ eingerichtet sein.

Mithilfe der Implementierung von C++-Code von der Arduino IDE in eine S-Funktion konnte eine verbesserte Servoansteuerung realisiert werden, dessen Entwicklung nachfolgend beschrieben ist. Diese stellt auch gleichzeitig den Leitfaden für die Entwicklung anderer S-Funktionen zum Betrieb auf einem Arduino-Mikrocontroller dar.

Bei einer S-Funktion handelt es sich um einen Block von der Simulinkbibliothek, bei dem die Ein- und Ausgangssignale flexibel definiert und darüber hinaus die Signaldaten blockintern mit eigenen Code verarbeitet werden können. Dies ermöglicht eine flexiblere Programmierung als es mit nur fest definierten Blöcken möglich ist.

Konkret für die erstellte S-Funktion zur Servoansteuerung ist in Abbildung 19 das Parametermenü des Blocks dargestellt.

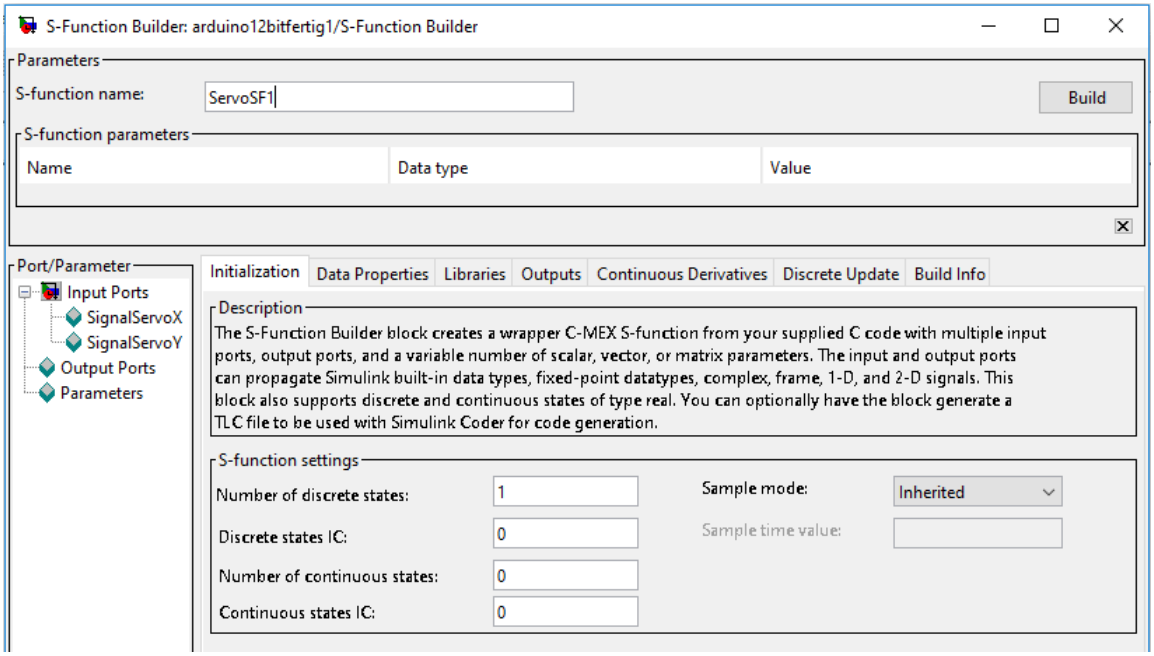


Abbildung 19: Parametermenü des S-Funktion-Blocks

Unter dem im Bild gezeigten Reiter „Initialisation“ wurde für den Parameter „Number of discrete states“ eine eins eingetragen, damit für die Funktion „Discrete Update“ eine Variable erstellt wird. Bei den anderen drei Parametern darunter kann null stehen, da kein Code für kontinuierliche Berechnungen verwendet wird. Hierbei sei zu erwähnen, dass das Programm für den Arduino als „fixed step discrete“ kompiliert wird. Damit wird das Programm in einer festgelegten Samplerate ausgeführt, wobei das diskret gegenüber kontinuierlich dafür steht, dass das Programm wiederholend mit diskreten Werten und einer diskreten Zeit ausgeführt wird. Dementsprechend ergänzend waren die Parameter „Sample mode“ und „Sample time value“ zu definieren. In diesem Fall wurde der Sample Mode jedoch auf „Inherited“ anstatt auf „Diskrete“ gestellt, damit die Samplerate durch andere Blöcke festgelegt wird und das Wechseln der Samplerate somit einfacher ist.

Unter dem Reiter „Data Properties“, der in Abbildung 20 gezeigt ist, sind die Ein- und Ausgangssignale und die Parameter der S-Funktion zu definieren. Konkret für die Ansteuerung der beiden Servomotoren waren nur zwei Eingänge für die Stellsignale zu erstellen. Diese wurden „SignalServoX“ und „SignalServoY“ genannt.

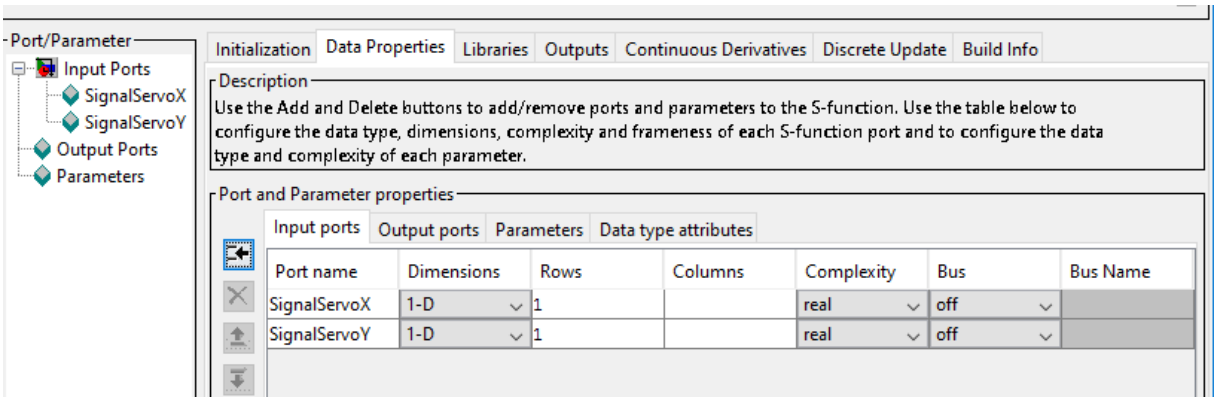


Abbildung 20: Reiter „Data Properties“ des S-Funktion-Blocks

Im Unterreiter „Data type attributes“ musste jeweils der Datentyp dieser Signale definiert werden. Dieser ist so zu wählen, dass der Datentyp des Ports mit dem Datentyp vom angeschlossenen Signal übereinstimmt. Als Datentyp für die Ports und der Signale wurde uint16 verwendet, der eine 16 bit Ganzzahl ohne Vorzeichen mit dem Wertebereich von 0 bis 65536 darstellt.
Weitergehend musste der Reiter „Libraries“, von dem in der Abbildung 21 ein Bild dargestellt ist, ausgefüllt werden. Während die Eingabefelder „Librarys/Objects/Source files“ und „External function declarations“ für die Programmierung am Arduino Due uninteressant sind, stellt das Eingabefeld „Include files and external function declarations“ das Äquivalent des oberen Codeabschnitts eines mit der Arduino IDE erstellten Programms dar. Hier können unter anderem Bibliotheken eingebunden und Variablen und Objekte erstellt werden. Gegenüber der Arduino IDE ist bei der S-Funktion jedoch mehr zu beachten. Alle Anweisungen müssen zwischen den Anweisungen „#ifndef MATLAB_MEX_FILE“ und „#endif“ geschrieben werden. Das #-Zeichen ist eine Präprozessordirektive, die den Compiler zu Codeeinfügungen veranlasst, bevor das Programm eigentlich kompiliert wird. Zum Beispiel wird die Anweisung „#include“ für die Einbindung von Bibliotheken durch den Quellcode der einzubindenden Bibliothek ersetzt. Bezüglich den Anweisungen „#ifndef MATLAB_MEX_FILE“ und „#endif“ wird der Code dazwischen im Gesamtprogramm eingebunden, wenn „MATLAB_MEX_FILE“ nicht definiert wurde.
Eine zweite Eigenheit bei S-Funktionen und speziell nur bei der Programmierung von Arduino Mikrocontroller nötig, ist die Anweisung „#define ARDUINO 100“. Ohne diese ist die S-Funktion nicht für Arduino-Boards funktionsfähig. Darunter kann der eigentliche Code geschrieben werden.

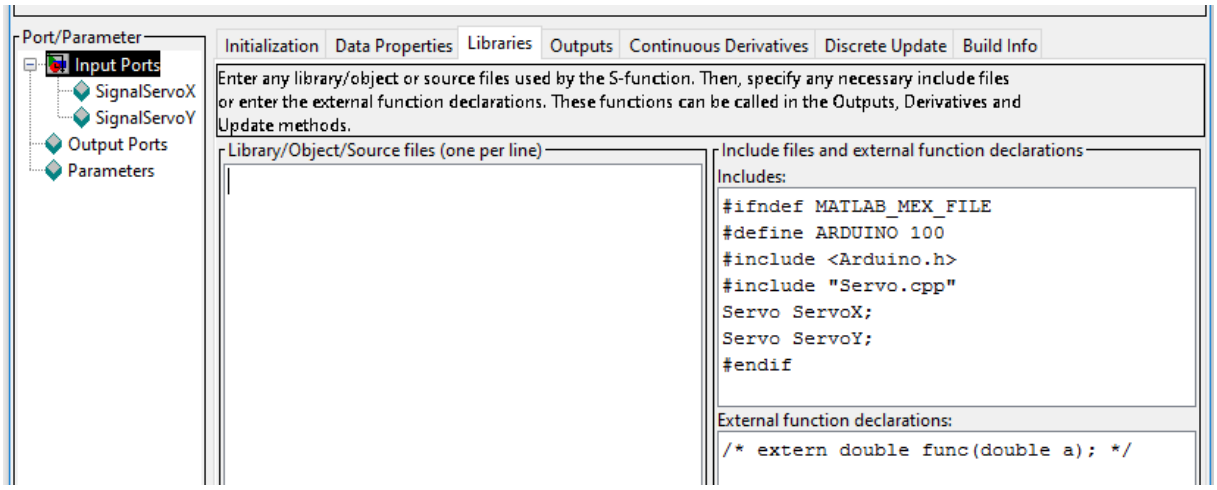


Abbildung 21: Reiter „Libraries“ des S-Funktion-Blocks

Anders als bei der Arduino IDE muss für die Bereitstellung von grundlegenden arduinospezifischen Funktionen die Bibliothek „Arduino.h“ mit der Anweisung „#include <Arduino.h>“ eingebunden werden. Die notwendige Bibliothek für die Servoansteuerung konnte mit „#include „Servo.cpp““ eingebunden werden. Dabei gilt es, die Bibliotheken auf richtige Weise einzubinden. Die grundlegenden Bibliotheken, wie z.B. die „Arduino.h“, dessen Speicherplatz dem Programm bekannt ist, können unter Verwendung der „<>“-Zeichen eingebunden werden. Die für die Servoansteuerung benötigte Bibliothek ist dem Programm unbekannt, weshalb es unter Verwendung von Hochkommas eingebunden werden muss. Zusätzlich müssen alle Dateien der Bibliothek im selben Ordner vom Simulinkmodell abgespeichert werden. Unter Verwendung einer Pfadangabe können auch andere Speicherorte ausgewählt werden.
Die Servobibliothek besteht aus den drei Dateien „Servo.h“, „Servo.cpp“ und „ServoTimers.h“, die beispielhaft unter dem Pfad C:\MATLAB\SupportPackages\R2016a\arduino-1.6.1\libraries\Servo\src\sam zu finden sind. Zu beachten ist, dass einerseits meistens nur die Bibliotheken vom Simulink Hardware Support Package und nicht von der Arduino IDE mit Simulink verwendet werden können und andererseits manchmal zwei Varianten einer Bibliothek für den Betrieb mit einem ATmega-Mikrocontroller (z.B. Arduino Mega) oder Arm-Mikrocontroller (Arduino Due) bereit stehen, wie es bei der Servo-Bibliothek ist. Sind die Bibliotheken getrennt, kann die Bibliothek für das Arduino Due meist im Unterordner namens „sam“ gefunden werden, wohingegen der Unterordner für ATmega-Mikrocontroller „avr“ genannt wird. Anders als typisch, musste die C++-Sourcedatei (Servo.cpp) anstatt der Headerdatei (Servo.h) eingebunden werden, was durch das lesen vom Code der Bibliothek erschlossen werden konnte. Die Headerdatei selbst bindet mit ihrem Code nur die „ServoTimer.h“ und nicht die „Servo.cpp“ ein, während die „Servo.cpp“ die „Servo.h“ einbindet. Mit der Einbindung der „Servo.cpp“ in der S-Funktion sind somit alle beteiligten Dateien der Bibliothek eingebunden. Im Sinne der Servoansteuerung musste in diesem Eingabefeld für die

beiden Servomotoren der X- und Y-Achse nur noch jeweils ein Objekt der Klasse Servo erstellt werden, die „ServoX“ und „ServoY“ genannt wurden.
Der Setupcodeabschnitt der Arduino IDE, der nur einmal ausgeführt wird, kann in der S-Funktion mit dem Eingabefeld im Reiter „Diskrete Update“ emuliert werden, der in Abbildung 22 mit dem für die Servoansteuerung entsprechenden Code gezeigt ist. Dafür wird die eine im Reiter „Initialisation“ mit dem „Number of Diskrete States“ erstellte Variable benötigt, auf die mit „xD[0]“ zugegriffen werden kann. Damit der Setupcode nur einmal ausgeführt wird, kann diese Variable in einer if-Anweisung mit einer entsprechenden Bedingung verwendet werden, die ab der zweiten Ausführung auf „false“ steht, wie es in der entsprechenden Abbildung dargestellt ist. Innerhalb dieser if-Anweisung kann der Setupcode geschrieben werden, wobei der Code auch hier zwischen „#ifndef MATLAB_MEX_FILE“ und „#endif“ stehen muss. Äquivalent zum zuvor beschriebenen Setupcode der Arduino IDE wurde hier zur Initialisierung der Servoansteuerung die Methode „attach()“ für die einzelnen Objekte aufgerufen.

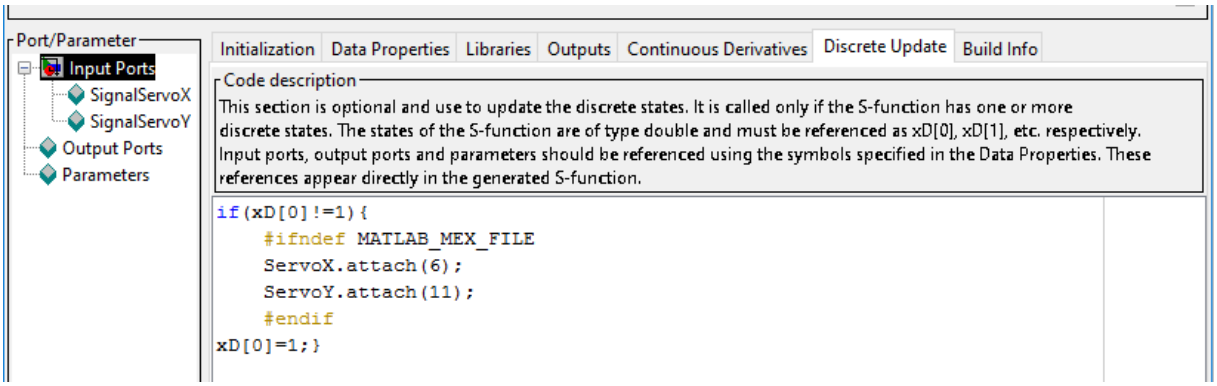


Abbildung 22: Reiter „Discrete Update“ des S-Funktion-Blocks für die Servoansteuerung

Als letztes muss der Code vom Hauptprogramm in die S-Funktion integriert werden. Dieser kann in das Eingabefeld des Reiters „Outputs“ geschrieben werden, wobei der Code wiederrum zwischen den Anweisungen „#ifndef MATLAB_MEX_FILE“ und „#endif“ stehen muss. Zusätzlich ist zu beachten, dass die im Reiter „Data Properties“ erstellten Ein- und Ausgangsvariablen der S-Funktion der Funktion Outputs als Zeiger übergeben werden. Deshalb mussten diese, um auf die Werte der Adressen zugreifen zu können, mit dem Operator „*“ dereferenziert werden, wie es in der Abbildung 23 zu erkennen ist.

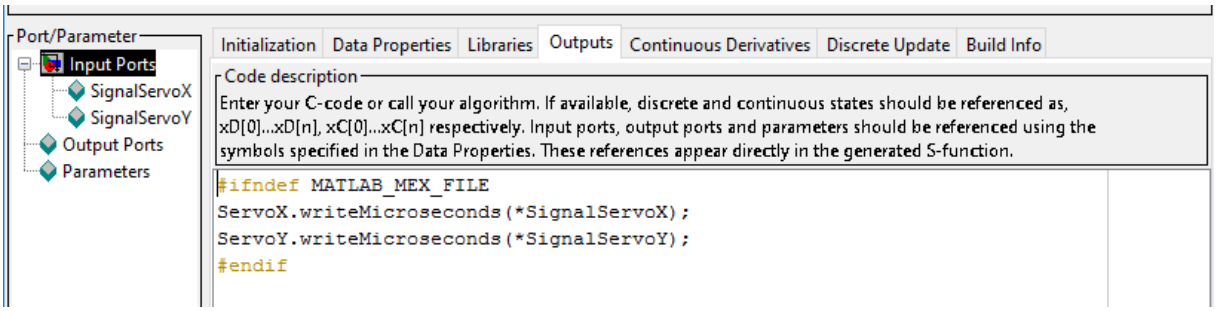


Abbildung 23: Reiter „Outputs“ des S-Funktion-Blocks

Nachdem die S-Funktion ausgefüllt ist, kann diese mit dem „Build“-Button erstellt werden. Dabei werden mehrere Dateien im aktuellen Matlabordner erzeugt, wovon die C-Sourcedatei mit der Endung „_wrapper.c“ die wichtigste darstellt.

Für eine einwandfreie Funktion der S-Funktion mussten Modifikationen vorgenommen werden. Zunächst wurde die Dateiendung dieser Datei von „.c“ auf „.cpp“ abgeändert, das den Dateityp von C-Quelldatei auf eine C++-Quelldatei ändert. Das ist insofern wichtig, weil das Programm bzw. der Compiler an dieser Stelle eine C++-Quelldatei erwartet. Dem zufolge musste auch der Quelltext dieser Datei für die Verwendung eines C++-Compilers angepasst werden. Dazu wurde vor allen Funktionen die Anweisung „extern „C““ geschrieben, wie es nachfolgend dargestellt ist:

```
extern "C" void ServoSF1_Outputs_wrapper(const uint16_T *SignalServoX,
            const uint16_T *SignalServoY,
            const real_T *xD)
extern "C" void ServoSF1_Update_wrapper(const uint16_T *SignalServoX,
            const uint16_T *SignalServoY,
            real_T *xD)
```

Diese Funktionen stellen C-Funktionen dar und sind ohne die Ergänzung „extern „C““ nicht für einen C++-Compiler geeignet. Die Anweisung „extern „C““ vor einer Funktion teilt dem C++-Compiler mit, diese Funktion wie ein C-Compiler zu behandeln. Dadurch wird als Symbol der Funktion nur der Funktionsname anstatt, wie beim C++-Compiler üblich, Funktionsname, Parameteranzahl und Parameterdatentypen als Symbol für die Funktion verwendet.

Bei normalen S-Funktionen für das Arduino Due sind nur zwei Funktionen als Quellcode in der Datei vorhanden. Eine stellt, wie es auch leicht am Namen zu erkennen ist, den Quellcode des Reiters „Outputs“ dar, während die andere Funktion den Quellcode des Reiters „Discrete Update“ beinhaltet. Ebenso kann man an den Funktionen erkennen, dass die Werte als Zeiger übergeben werden, weswegen diese in der S-Funktion dereferenziert werden müssen. Die Verwendung von Zeigern hat performancetechnische Vorteile und wird deshalb gerne für geschwindigkeitskritische Aufgaben, wie hier bei der Mikrocontrollerprogrammierung, eingesetzt. Im vorliegenden Fall werden Parameter als Zeiger mit dem Zusatz „const“ übergeben, woraus sich ableiten lässt, dass die Eingänge

der S-Funktion nicht überschrieben sondern nur ausgelesen werden können. Will man die Werte der Eingänge modifizieren, müssen zusätzliche Variablen in der S-Funktion erstellt werden, denen die Werte zuerst zugewiesen werden können.
Oberhalb der Funktionen sind in der automatisch generierten Datei der Quelltext des Reiters „Libraries“ und einige ergänzende Anweisungen aufgelistet.
Vor dem Ausführen des Simulinkmodells wurden die Einstellungen in Simulink für den Betrieb auf dem Arduino Due angepasst. Im „Model Configurations Parameters“-Menü wurde im Reiter „Solver“ die „Solver options“ auf „Fixed step“ und „discrete (no continuous states)“ gestellt. Weitergehend unter dem Reiter „Hardware Implementation“ wurde als Hardware-Board das Arduino Due ausgewählt. Unter diesem Reiter können noch weitere Einstellungen für das Arduino Due getroffen werden, die aber zunächst nicht benötigt werden.
Nach dem Einstellen vom Modus „External“ im Simulinkfenster konnte das Modell gestartet werden, wobei das Arduino über USB am Rechner angeschlossen sein musste. Den USB-Port, an dem das Arduino angeschlossen ist, findet Simulink in der Regel automatisch. Bei Verbindungsschwierigkeiten kann dieser unter „Model Configuration Parameters=>Hardware Inplementations=>Set host COM port“ manuell festgelegt werden. Der entsprechende COM-Port, an dem das Arduino angeschlossen ist, kann im Gerätemanager von Windows nachgeschaut werden.
Ist die S-Funktion fehlerhaft, wird dies meistens erst beim kompilieren oder hochladen des Codes auf das Arduino-Board erkannt und eine zumeist sehr unspezifische Fehlermeldung ausgegeben, wodurch das Finden und Ausbessern von Fehlern aufwendig ist.
Schlussendlich konnte mit der beschriebenen S-Funktion die bestmögliche Servoansteuerung vom Arduino auch in Simulink erfolgreich implementiert werden.

5.2 Positionsdatenerfassung mit integriertem A/D-Wandler

5.2.1 Hardwaretechnische Umsetzung

Neben der Ansteuerung von Servomotoren ist die Positionserfassung der Kugel über das resistive 5-Draht-Touchscreen für dieses System von maßgebender Bedeutung. Wie schon beschrieben, sollte die Positionsmessung, aufgrund des D-Anteils im Regler, möglichst genau und mit möglichst geringer Latzenz realisiert werden. Dazu wurden zunächst die 12 bit genauen, integrierten A/D-Wandler des Arduino Due verwendet und die Möglichkeit zur Einbindung besserer A/D-Wandler als Verbesserungsmaßnahme offengehalten. Für die Positionserfassung an einem 5-Draht-Touchscreen wird nicht nur ein A/D-Wandler benötigt, sondern auch vier digital steuerbare Spannungsausgänge,

die vom Arduino Due mit den meisten Pins bereitgestellt werden können. Zu beachten ist dabei, dass manche Pins des Arduino weniger Stromstärke standhalten können als die allgemeine Spezifikation vorgibt. Dazu gibt es eine Pinkarte, bei der die Pins einzeln spezifiziert sind. Das ist hierbei insofern wichtig, weil eine Messung des Gesamtwiederstands vom Touchscreen ergeben hat, dass dieser im Bereich von 70 Ohm liegt. Legt man die Spannung von 3,3 V des Arduino zugrunde, werden für das Betreiben des Touchscreens ca. 47 mA benötigt. Die digital-Pins würden damit deutlich über den Spezifikationen betrieben werden und die entsprechenden Transistoren auf Dauer Schaden nehmen. Außerdem wäre eine stabile Spannungsausgabe nicht mehr gewährleistet, die sich direkt auf die Qualität der Positionsmessung auswirken würde. Um dies zu verhindern, wurden für einen Anschluss am Touchscreen jeweils drei von den leistungsfähigen Pins zusammengelegt. Somit sind jeweils sechs Anschlüsse für das Bereitstellen der Stromstärke und als Masse vorgesehen, womit sich eine Gesamtstromstärke von $90\ mA$ beim Bereitstellen und $54\ mA$ beim Aufnehmen der Stromstärke berechnet. In diesem Hinblick hat das Arduino Mega einen Vorteil, da dessen digital-Pins bis zu 50 mA bereitstellen können und somit deutlich robuster sind.
Das verwendete Touchscreen ist, wie fast alle, für den Betrieb mit 5 V Spannungen ausgelegt. Dieses kann ohne Komplikationen auch mit den hier verwendeten 3,3 V betrieben werden. Die A/D-Wandler des Arduino Due haben ohnehin nur einen Messbereich von 0 bis 3,3 V und sind damit genau für die Betriebsspannung des Arduino ausgelegt. Der Messbereich der A/D-Wandler kann, im Gegensatz zum Arduino Mega, nicht verändert werden. Die untere Abbildung 24 zeigt das Arduino Due am Tabledance mitsamt der kompletten Verdrahtung, die für den Betrieb mit den integrierten A/D-Wandler nötig war.

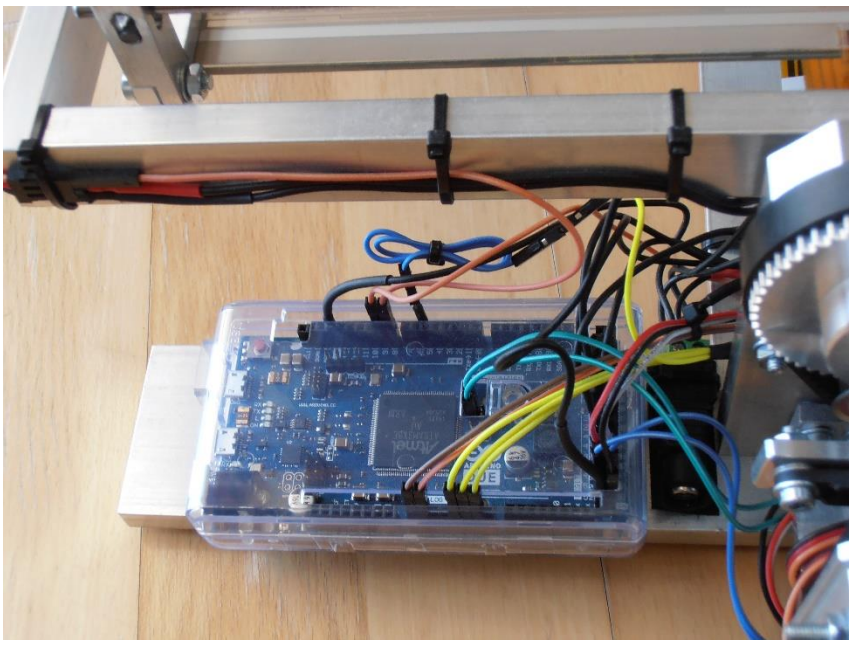


Abbildung 24: Arduino Due mit Verdrahtung

5.2.2 Umsetzung der Positionsdatenerfassung in Simulink

Für die Ansteuerung des Touchscreens werden die arduinospezifischen Funktionen „digital out“ und „analog in“ gebraucht, die in der Simulinkbibliothek bereitgestellt werden und deshalb zunächst keine S-Funktion benötigt wird.
Abbildung 25 zeigt ein einfaches Simulinkmodell für Kugelpositionserfassung, wobei dieses als Subsystem im Gesamtmodell verschachtelt ist.

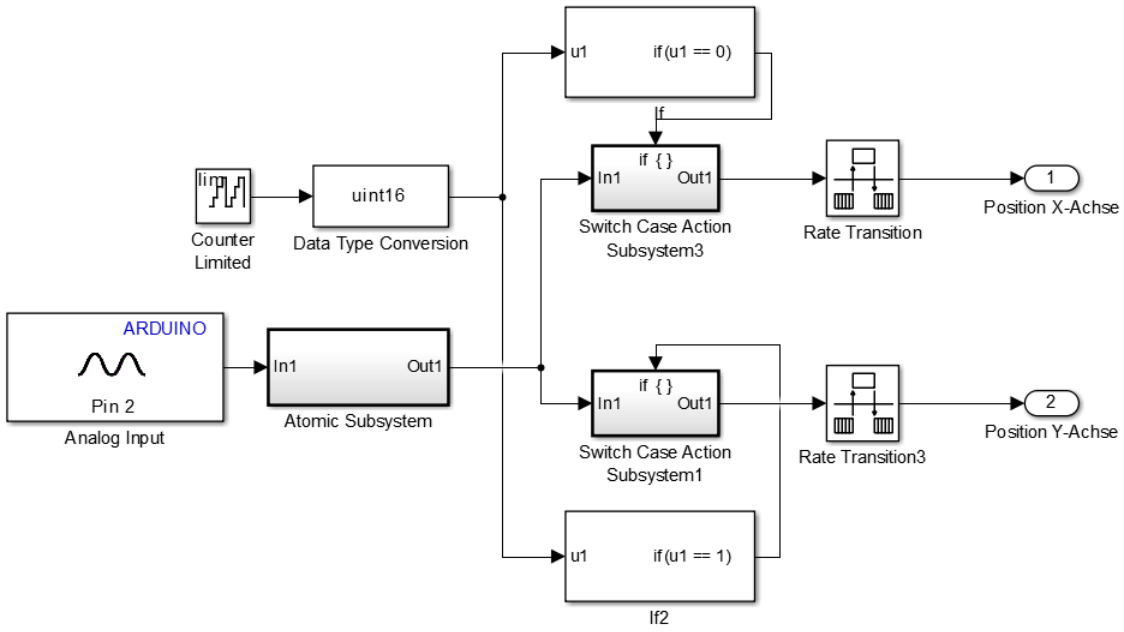


Abbildung 25: Simulinkmodell für die Positionserfassung

Pro Takt werden mit dieser Programmarchitektur die digitalen Spannungen jeweils zur Messung einer Koordinate umgeschaltet und eine Koordinate gemessen, weswegen für ein X-Y-Koordinatenset zwei Takte benötigt werden. Am Ende sind jeweils „Rate-Transition“-Blöcke, die die Samplerate von den Blöcken außerhalb des Subsystems halbieren. Dadurch ist im weiteren Programmverlauf pro Takt ein komplettes X-Y-Koordinatenset vorhanden. Mit anderen Worten erklärt, wird dieser Teil des Programms doppelt so oft ausgeführt wie der Rest vom Programm. Das hat gewisse Nachteile, weswegen die Programmarchitektur später verändert wurde.
Ein X-Y-Koordinatenset mit einem Takt zu messen funktioniert nicht ohne weiteres, da die digitalen Spannungen nach jeder Messung auf die andere Koordinate umgeschaltet werden müssen und es ca. 300 µs braucht bis die endgültige Spannung am Messdraht anliegt. Das Touchscreen selbst besitzt eine bestimmte Kapazität, die hauptsächlich durch die großen gegenüberliegenden und leitfähigen Flächen gebildet wird. Dies verursacht Lade- und Entladezeiten, die für eine genaue Messung beachtet werden müssen. Deshalb wird im erstellten Programm erst die Position mit dem „Analog-

In“-Block gemessen und direkt anschließend die digitalen Spannungen für das Messen der nächsten Koordinate umgeschaltet, wodurch die maximale Zeit für das Einpendeln der Spannung sichergestellt ist. Die Blöcke zur Spannungsumschaltung sind in einem Subsystem verschachtelt, dessen Inhalt in Abbildung 26 dargestellt ist.

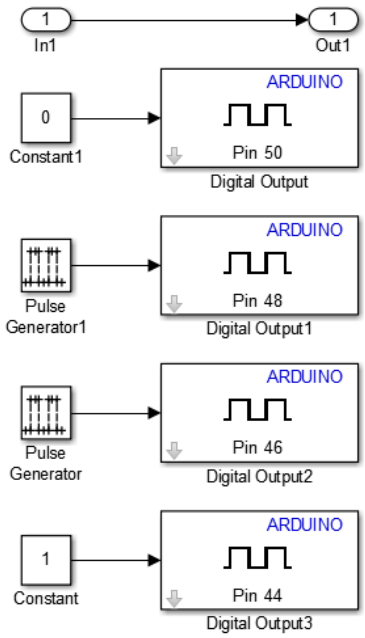


Abbildung 26: Simulinkmodell zur Spannungsumschaltung

Obwohl der Messwert für das Umschalten der Spannungen nicht benötigt wird, ist es wichtig, dieses Umschalten im Programmablauf festzulegen. Werden diese Blöcke allgemein in der oberen Ebene des Modells platziert, ist es rein zufällig, wann diese im Programmablauf ausgeführt werden. Demnach kann die Spannung fälschlicherweise zu früh oder zu spät gewechselt werden, was zwingend zu falschen Messergebnissen führt. Es werden jeweils drei Pins mit dem gleichen Spannungswert angesprochen. Auch müssen für eine korrekte Ansteuerung nur zwei Anschlüsse die Spannungen umschalten. Legt man den Touchscreen so, dass der Nullpunkt für X und Y links unten ist, dann muss der Anschluss links unten immer auf Ground und rechts oben immer auf 3,3 V gelegt werden. Ein Versuch, diese Anschlüsse mit nativen 3,3 V- und Ground-Pins zu verbinden, die vom Arduino bereitgestellt werden, ist aufgrund einer deutlichen Linearitätsverfälschung nicht realisiert worden. Das ist damit zu erklären, dass die über Transistoren geschalteten Spannungen einen anderen Wiederstand besitzen als welche, die direkt bereitgestellt werden und somit der am Touchscreen gemessene Wiederstand verfälscht wird. Das äußert sich in einem nichtlinearen Verlauf der gemessenen gegenüber der echten Position, wodurch die Position nicht mehr korrekt abgebildet wird.

Für die Messung der X-Koordinate wird links oben Ground und rechts unten 3,3 V angelegt. Diese werden zur Messung der Y-Koordinate genau vertauscht. Realisiert wurde dieses Umschalten mit „Pulse-Generator“-Blöcken, die taktbasiert zwischen 0 und 1 wechseln.

Die gemessenen X-Y-Positionswerte sind nach dem Messblock zusammengelegt und müssen für die weitere Verarbeitung in X- und Y-Koordinaten getrennt werden. Dazu wurde ein Zähler verwendet, der wiederholend von 0 zu 1 zählt und als Bedingung in den zwei if-Blöcken entsprechend jeden zweiten Messwert durchlässt. Dabei steht 0 für eine X-Koordinate und 1 für eine Y-Koordinate. Die if-Blöcke sind so konfiguriert, dass sie den letzten berechneten Wert beibehalten, wenn sie nicht ausgeführt werden. Somit liegt nach zwei Takten immer ein neues X-Y-Koordinatenset an den „Rate-Transition“-Blöcken an.

Zu diesem Modell sei zu erwähnen, dass der Block „Analog In“ mit 10 bit Genauigkeit als Grundeinstellung misst, obwohl der Mikrocontroller 12 bit A/D-Wandler integriert hat. Das ist der Kompatibilität zu anderen Arduino-Boards geschuldet, die allesamt mit 10 bit maximaler Auflösung messen. Die Auflösung kann auch nicht bei den Blockparametern eingestellt werden. Um dennoch mit 12 bit Auflösung messen zu können, muss die für das Arduino Due spezifische Funktion „AnalogReadResolution(12);“ im Setupcode des Mikrocontrollers ausgeführt werden. Damit nicht extra eine S-Funktion dafür erstellt werden musste, wurde diese Funktion in das „Discrete Update“-Fenster der S-Funktion für die Servoansteuerung geschrieben.

5.3 Erstellung eines Regelungstestprogramms

In Abbildung 27 ist das erstellte Regelungstestprogramm gezeigt, welches eine komplette Regelungsschleife darstellt. Dazu sei erwähnt, dass dieses Programm eine minimalistische Grundausführung darstellt, die dennoch ausreichend ist, um eine Kugel auf einem Punkt auf dem Touchscreen zu regeln.

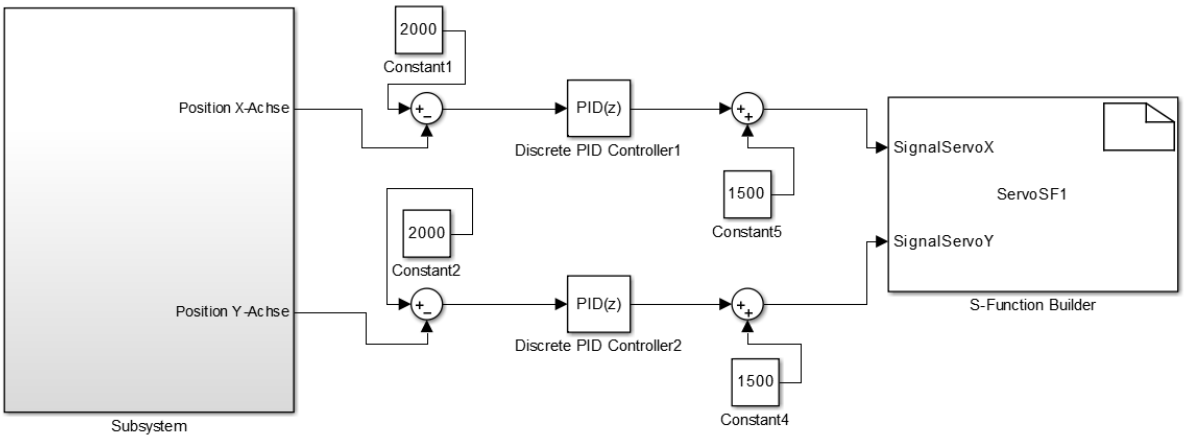


Abbildung 27: Gesamtübersicht zum Regelungstestprogramm

An der linken Seite ist das beschriebene Subsystem für die Positionsmessung platziert, während an der rechten Seite die S-Funktion für die Servoansteuerung ist. Dazwischen sind lediglich Blöcke für die Berechnung der Regelungsabweichung als Differenz zwischen Ist- und Sollwert, ein PID-Regler, der als diskreter Regler parametriert ist und Blöcke zur Addierung des Stellwertoffsets. Die Servomotoren wurden über das Zahnriemengetriebe so ausgerichtet, dass der Stellwert von 1500 für die Mittelposition der Stellmotoren auch das Touchscreen in etwa waagerecht ausrichtet. Mit dem Ändern dieses Wertes kann das System auf unterschiedlich geneigte Unterflächen eingestellt

werden, ohne dass die Neigung mechanisch durch z.B. Stellschrauben angepasst werden muss. Es ist nur darauf zu achten, dass das Touchscreen in X- und Y-Richtung letztendlich möglichst waagerecht ausgerichtet ist. Abweichungen zur Waagerechten führen zu dauerhafter Regelungsabweichung und können das System abhängig vom Regler auch in Dauerschwingung versetzen.
Beim Testen des Programms wurde schnell klar, dass die Positionserfassung noch deutlich zu ungenau ist. Dies äußert sich durch starkes Zittern der Servomotoren und einem ungenügenden Regelungsverhalten. Implementiert ist ein PD-Regler, wovon der D-Anteil des Reglers, wie schon beschrieben, auf Messrauschen sehr empfindlich reagiert und ein störendes „zittern“ der Servomotoren verursacht. Mit einem geringem D-Anteil und einem hohem Filterkoeffizienten vom D-Anteil konnte dennoch eine stabile Regelung erreicht werden. Der Filterkoeffizient mittelt, je nach Höhe, mehr oder weniger vorangegangene Messpunkte, sodass das Rauschen im Signal geringer wird. Dies wiederrum verzögert das daraus gewonnene Positionssignal gegenüber der echten Position der Kugel, was sich negativ auf das Regelungsverhalten auswirken kann, wie es auch schon in der vorangegangenen Projektarbeit beschrieben ist. Deshalb ist es auch nicht sinnvoll einen zu starken Filterkoeffizienten zu verwenden, der die ansonsten latenzarme Regelung mit einer beachtlichen Latenz im D-Regler belegt.
Die Analyse der Positionssignale hat ergeben, dass diese lediglich eine effektive Auflösung von 8-9 bit haben und somit deutlich unter der möglichen Auflösung von 12 bit der A/D-Wandler sind. Es ist aber auch üblich, dass A/D-Wandler eine geringere effektive Auflösung als die angegebene Nennauflösung besitzen. Messungen mit stabilisierten Spannungsquellen haben gezeigt, dass die 12 bit A/D-Wandler des Arduino Due eine effektive Auflösung von bis zu 11 bit bieten. Demnach ist ein rauschendes Messsignal vom Touchscreen ein wesentlicher Faktor für die geringe effektive Auflösung. Untersuchungen haben gezeigt, dass das verrauschte Messsignal sowohl durch die nicht hochstabile Spannungsausgabe der digitalen Pins als auch durch das Touchscreen als Sensor selbst verursacht wird. Dazu wird in einem späteren Kapitel nochmals genauer eingegangen.
Ausgehend vom erstellten Programm konnte nichtsdestotrotz die grundlegende Funktion des Systems sichergestellt und Verbesserungsansätze eruiert werden, die den gestellten Anforderungen bezüglich des Regelungsverhaltens genügen. Die vorgenommenen softwaretechnischen Verbesserungen und die geforderten Funktionserweiterungen sind in den nächsten Kapiteln beschrieben.

6. Aufbau des weiterentwickelten Regelungshauptprogramms

6.1 Überblick vom Regelungsprogramm

Durch konsequente Weiterentwicklung der Software konnte ohne Änderung der Hardware letztendlich eine gute Regelung erreicht werden. Zudem wurden einige geforderte Features ins Regelungsprogramm eingebaut, welches in Abbildung 28 im Überblick gezeigt ist. Wie zu erkennen ist, ist das Programm grob in zwei Hälften gegliedert. Die obere Hälfte ist für die Bedienung des Programms vorgesehen und selbst in mehrere beschriftete Bereiche unterteilt. Neben der Sollwertvorgabe und der Regelungsparametereinstellung sind in der oberen Hälfte noch Blöcke für Visualisierung, Auswertung, Stellwertstörung, Sprunggenerierung und für die Offsetkalibrierung implementiert.

Die untere Hälfte vom Programm beinhaltet das eigentliche Regelungsprogramm mit dem Regelungsalgorithmus. Dieser ist übersichtlich strukturiert und besteht im Wesentlichen aus den regelungsspezifischen Bestandteilen Positionserfassung, Vergleich Soll-Istwert, Regler und der Servoansteuerung. Hierbei sei zu erwähnen, dass einige Signalverbindungen für die Übersichtlichkeit durch sogenannte „Goto“- und „From“-Blöcke ausgetauscht wurden. Dadurch werden unübersichtliche Signalverbindungen vermieden und die wesentlichen Signalverbindungen hervorgehoben. Für die Codegenerierung hat diese Maßnahme keine Auswirkung.

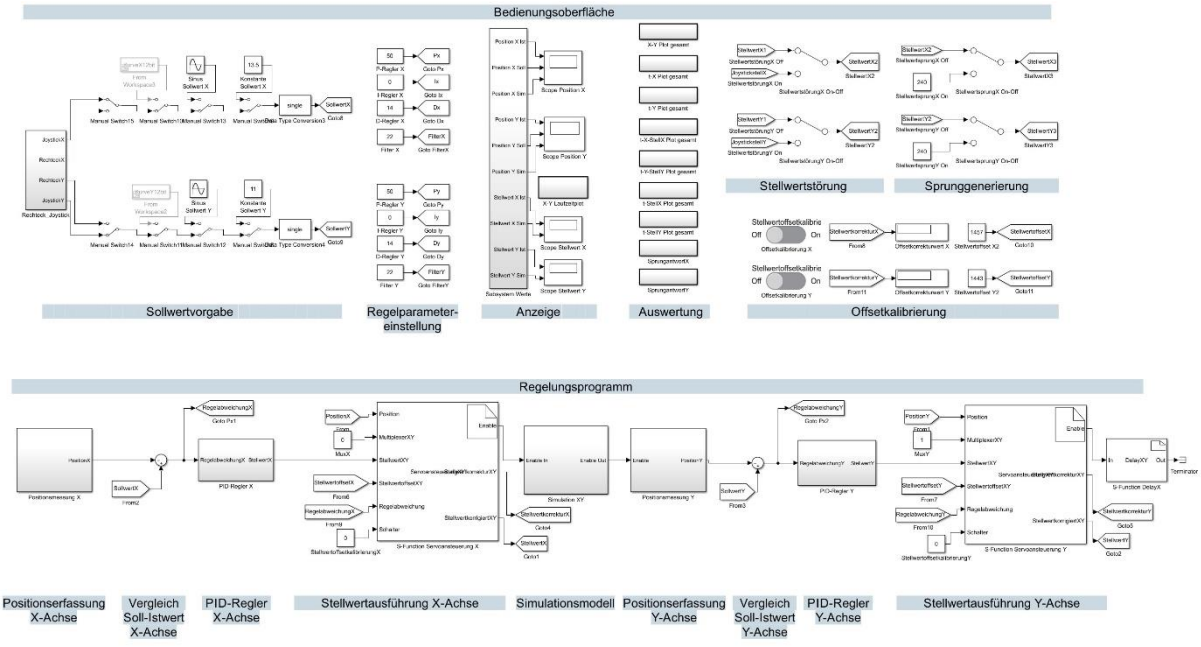


Abbildung 28: Gesamtes Simulinkmodell für die Regelung

6.2 Struktur und Programmierung der Regelungsschleife

Das zuerst verwendete softwaretechnische Konzept für den Aufbau der Regelung wurde im Rahmen der Verbesserung stark umgebaut. Waren vorher für Positionsdatenerfassung der Kugel in X- und Y-Richtung zwei Takte nötig, die mit einem „Rate-Transition“-Block vom Rest des Programms getrennt waren, so wird der ganze Regelkreis für die X- und die Y-Achse nun mit einem Takt ausgeführt. Dies bringt einige Vorteile aber auch mehrere kleine Nachteile mit sich, die vor allem die Programmierung betreffen. Vorteilhaft ist im besonderem die verbesserte Verteilung der Rechenlast für den Mikrocontroller, der für jeden Takt etwa gleich viel zu berechnen hat, während der Mikrocontroller davor im ersten Takt nur die Bestimmung der Position der Kugel auf der X-Achse und im zweiten Takt alles restliche für die X- und Y-Achse berechnen musste. Mit der verbesserten Rechenlastverteilung kann man den Regelkreis auch mit höheren Sampleraten betreiben, was auch angestrebt wurde. Ein anderer Vorteil liegt aber auch darin, dass so schnell wie möglich nach dem Messen der Position einer Achse das berechnete Stellsignal dem Servomotor ausgegeben wird. Das minimiert die Latenz zwischen der Messung und der tatsächlichen Stellwertausführung. Vor allem bei geringeren Sampleraten hat dies gravierende Auswirkungen, da die Latenz zuvor von der Messung zur Stellwertausgabe über einen Takt betragen konnte.

Für die erneuerte Programmstruktur mussten aber auch mehrere softwaretechnische Probleme bewältigt werden, die es mit der anderen Struktur nicht gab. Zwischen dem Umschalten der digitalen Spannungen vom Touchscreen und der Messung der Position muss eine gewisse Mindestzeit eingehalten werden, damit sich die Messspannung am Touchscreen einpendeln kann. Die Zeit für das Einpendeln beträgt in etwa 300 µs.

Aus diesem Grund musste die Berechnungsreihenfolge vom Regelkreise festgelegt und zeitlich angepasst werden. Die Berechnungsreihenfolge wurde durch zusätzliche abhängige Variablen und Signale festgelegt. Sind keine Abhängigkeiten zwischen den verschieden Programmabschnitten vorhanden, wie es z.B. beim X-Achsenregelkreis und Y-Achsenregelkreis wäre, so wird das Programm mit einer zufälligen Reihenfolge kompiliert. Damit das umgangen werden konnte, mussten künstliche Abhängigkeiten generiert werden, sodass keine Freiheiten bei der Reihenfolge der Kompilierung blieben. Mit der Abbildung 29 und der fortführenden Abbildung 30 ist das Regelungsprogramm gezeigt, bei dem gut zu erkennen ist, dass jeder Block und jedes Subsystem mit mindestens einem Signal verbunden ist und damit die Reihenfolge der Kompilierung von links nach rechts festliegt.

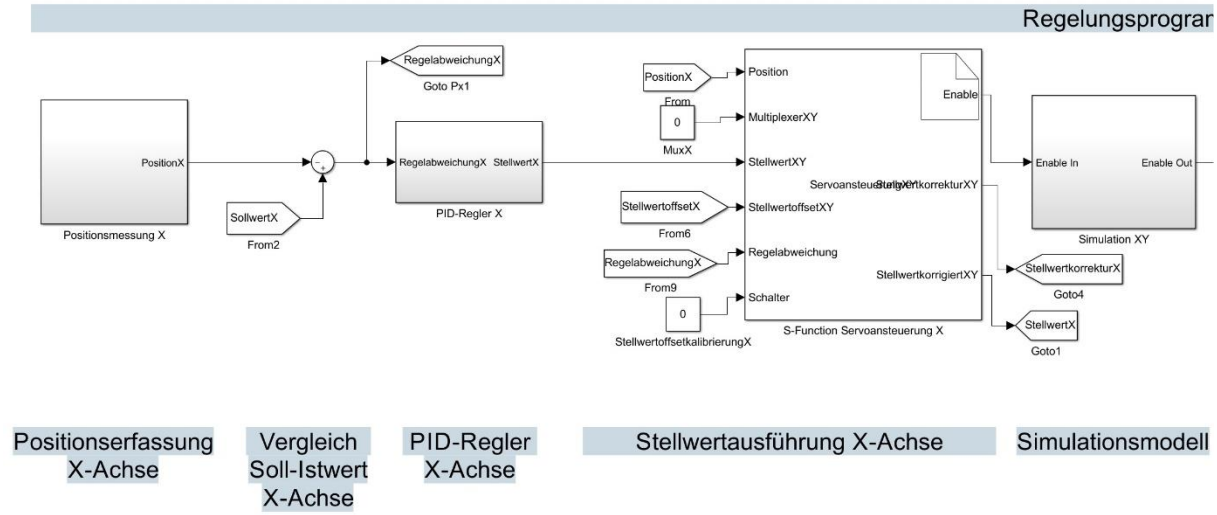


Abbildung 29: Regelungsprogrammabschnitt für die X-Achse

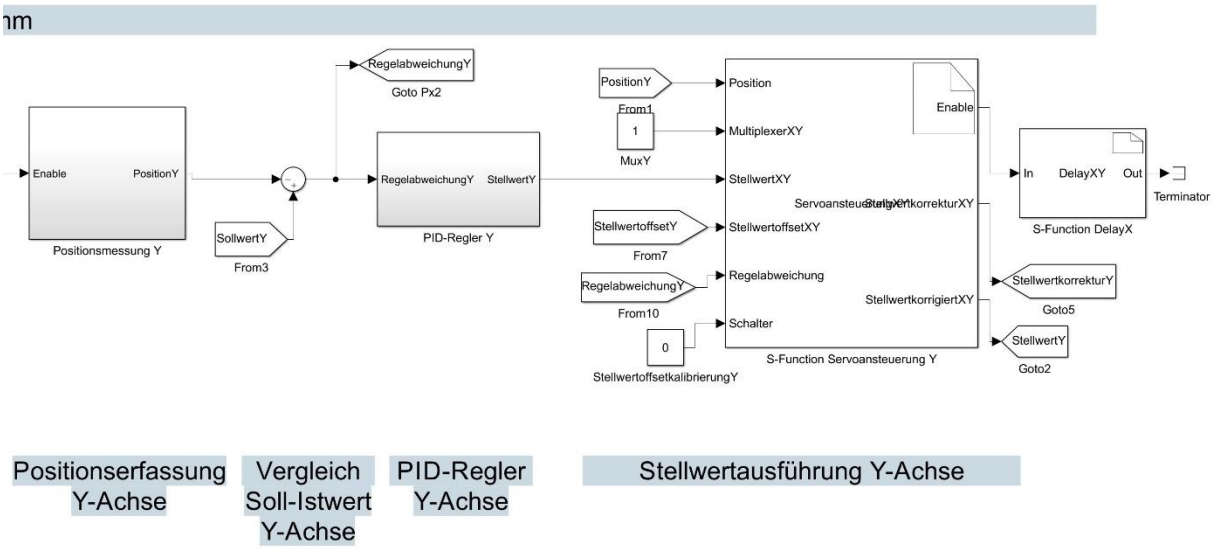


Abbildung 30: Regelungsprogrammabschnitt für die Y-Achse

Wie auf den Abbildungen zu sehen, wird als erstes die Regelschleife für die X-Achse ausgeführt. Darunter zählt die Positionserfassung, Berechnung der Regelungsabweichung, Berechnung des Stellwerts mit dem Regelungsalgorithmus und die S-Funktion für die Stellwertausgabe, bei dem auch das Umschalten der digitalen Spannungen integriert ist. Anschließend ist ein Subsystem für die Berechnung der Simulation angesetzt. Nach dem wird die Regelschleife für die Y-Achse ausgeführt, die ähnlich zur Regelschleife für die X-Achse ist. Damit eine Mindestzeit von 300 µs zwischen dem Umschalten der digitalen Spannungen und der Spannungsmessung eingehalten wird, sind im Programm zwei S-Funktionen mit dem bereits beschriebenen Befehl „delayMicroseconds()“ eingebaut. Um für den Mikrocontroller die Wartezeit zu verkürzen und Leistungsressourcen zu sparen, wurden bevorzugt unabhängige Aufgaben, wie die Simulationsberechnung, zwischen dem

Umschalten der Spannungen und der Messung platziert. Der Wert für den Befehl „delayMicroseconds()“ musste daher letztendlich durch ausprobieren herausgefunden werden, ist aber nicht viel geringer als 300 µs, was darauf schließen lässt, dass der Mikrocontroller nicht übermäßig belastet ist und noch einige Ressourcen zur Verfügung hat.

6.2.1 Positionserfassung

Für ein besseres Regelungsverhalten war es offensichtlich, dass die Positionserfassung stark verbessert werden musste. Dabei wurde eine Überabtastung (Oversampling) angewandt, bei dem mehrere Werte gemessen und zu einem Wert gemittelt werden. Dazu mussten die A/D-Wandler das Arduino bezüglich der maximalen Abtastraten genauer untersucht werden. Muss das Arduino Due nur Spannungen messen, so kommt dieser auf über 50 kHz bei der A/D-Konvertierung, wobei der Mikrocontroller damit voll ausgelastet ist und keine anderen Aufgaben abarbeiten kann. Da das verwendete Programm umfangreich ist und auch Wartezeiten für die Spannungsänderung eingehalten werden müssen, liegt die effektiv für Überabtastung verwendbare Samplerate weit darunter.
Abbildung 31 zeigt das verbesserte Subsystem für die Positionserfassung der Kugel auf der X-Achse.

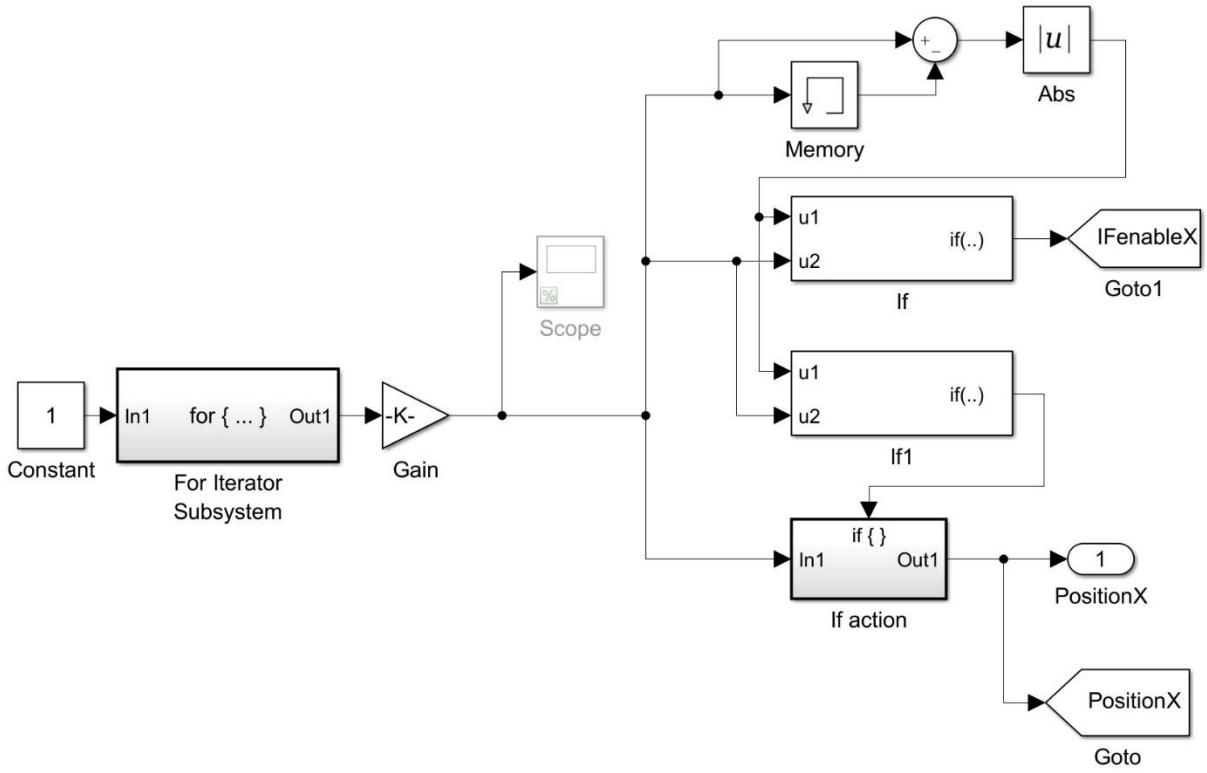


Abbildung 31: Positionserfassung für die X-Achse

Die Samplerate vom gesamten Programm wird durch die erste Konstante an der linken Seite der X-Achsenpositionserfassung vorgegeben. Alle anderen Blöcke und S-Funktionen sind so konfiguriert, dass die Samplerate vom ersten Block die Samplerate vom ganzen Programm vorgibt. Dadurch kann die Samplerate bei Bedarf einfach geändert werden. Verwendet wird eine Samplerate von 333 Hz bzw. eine Samplezeit von 0,003 s, die an die maximal verwertbare Signalsamplerate der Servomotoren von ebenfalls 333 Hz angepasst ist. Die höhere Samplerate vom Programm verbessert das Regelungsverhalten gegenüber Konfigurationen mit niedrigeren Sampleraten aber gleich hoher effektiver Überabtastung, dass sich aus der Samplerate und der pro Takt vorgegeben A/D-Konvertierungen berechnet, deutlich. Grund ist hierfür hauptsächlich neben der geringeren Latenz des Systems auf Kugelpositionsänderungen auch die geringere Auswirkung von einzelnen Stellwertausreißern auf den tatsächlichen Stellwinkel. Die Zeit zwischen zwei Takten als auch die maximal mögliche Latenz zwischen Messung und Stellwertausgabe beträgt somit lediglich 3 ms, wodurch eine latenzarme Regelung garantiert ist. Würde die Taktrate von 333 Hz dem Mikrocontroller zu viel Leistung nehmen, so wäre eine geringere Taktrate mit höherer Überabtastung sinnvoller. Tests haben aber gezeigt, dass die Taktrate den Mikrocontroller nicht übermäßig belastet und noch genug Ressourcen für eine mehrfache A/D-Wandlung vorhanden sind, obwohl alleine die Wartezeit von 600 µs pro Takt bei der hohen Taktrate ziemlich genau 20% der Gesamtzeit beträgt. Für jeden Takt werden pro Achse in einer for-Schleife, dessen Inhalt in der Abbildung 32 abgebildet ist, zunächst 15 Werte gemessen und aufaddiert. Als Zwischenspeicherelement dient der „Memory“-Block. Insgesamt ergibt sich mit der Taktrate von 333 Hz des Programms eine Samplerate von 9990 Hz für den A/D-Wandler. Zu erwähnen ist hierbei eine Eigenheit von Simulink, die die mehrfache Verwendung von einigen Blöcken mit der gleichen Pinnummer als Parameter nicht zulässt. Konkret mussten die zwei benötigten „Analog In“-Blöcke für die X- und Y-Positionserfassung mit unterschiedlichen Pinnummern versehen werden, obwohl das gleiche Signal nur zu einem anderen Zeitpunkt nochmals gemessen werden sollte. Deshalb mussten zwei „Analog In“-Anschlüsse für die Messung zum Arduino verdrahtet werden.

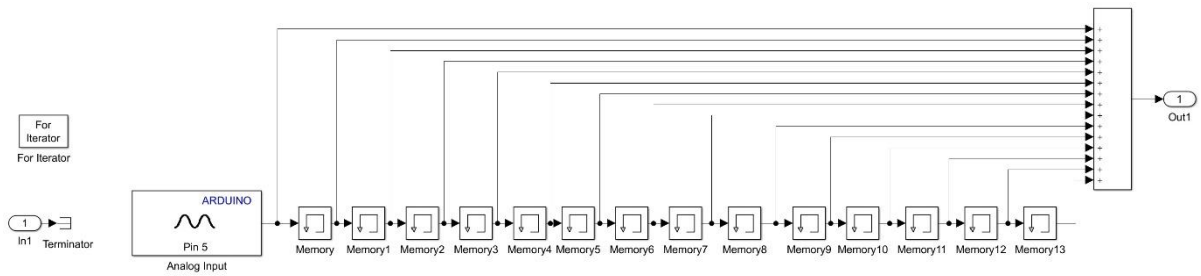


Abbildung 32: For-Schleife für die Überabtastung

Nach der for-Schleife wird das Positionssignal per Multiplikation mit 0,06667 fertiggestellt und vom ursprünglichen Datentyp uint16 in den genaueren Gleitkommadatentyp single umgewandelt, welcher für diese und den darauf folgenden Berechnungen besser geeignet ist. Danach sind zwei gleichartige if-Anweisungen angesetzt, von die eine die Weitergabe neuer Positionsdaten bedingt und die andere als Bedingung für die Ausführung des Regelungsalgorithmus fungiert. Die if-Bedingungen sind falsch, wenn der Wert außerhalb der kausalen Grenzen liegt oder sich zu schnell ändert. Damit kann das Abheben der Kugel vom Touchscreen erkannt und negative Auswirkungen, wie z.B. starke Stellwertänderungen, vermieden werden. Dazu musste das Sensorsignal über einen Wiederstand geerdet werden, sodass die Spannung beim Abheben der Kugel auf null sinkt. Verwendet wurde ein Widerstand mit 220 kΩ, der deshalb so gewählt wurde, weil ein geringerer Widerstand zunehmend die Messgenauigkeit verschlechtert, während bei einem größeren Widerstand die Spannung zu langsam auf null abfällt.
Sind die if-Bedingungen wahr, so wird der neue Positionswert mit dem „If action“-Block weitergegeben, bei dem das Signal noch zur besseren Verwendbarkeit modifiziert wird. Das Touchscreen deckt nicht den ganzen Spannungsbereich des A/D-Wandlers mit einem entsprechenden Positionssignal ab, weshalb der Nullpunkt durch Subtraktion einer ermittelten Konstante angepasst wurde. Zudem wird im Subsystem das Signal mit einem berechneten Faktor multipliziert, sodass das daraus berechnete Signal die Position der Kugel in Zentimeter wiedergibt.

6.2.2 Regler

In Abbildung 33 ist der entsprechende Abschnitt für den Regler und der vorrausgehenden Regelungsabweichungsberechnung aufgezeigt, bei dem der Sollwert über einen „From“-Block zugeführt ist.
Als Regelungsalgorithmus wird, wie beim Testprogramm, weiterhin ein diskreter PID-Regler-Block verwendet, der jedoch mehrfach in ein Subsystem verschachtelt ist. Dies dient zum einen der Übersichtlichkeit an der obersten Ebene vom Hauptprogramm und ist aufgrund der if-bedingten Ausführung des Reglers, das nur im Subsystem möglich ist, zudem auch notwendig.

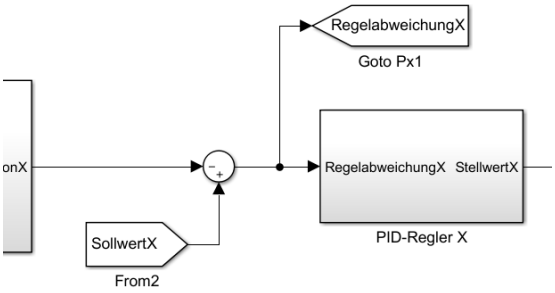


Abbildung 33: Übersicht vom Regler mit Regelungsabweichungsberechnung

Dazu ist mit Abbildung 34 die erste Unterebene vom Regelungssubsystem dargestellt, bei dem alle für den Regler benötigten Signale an dem if-bedingten Ausführungsblock angeschlossen sind.

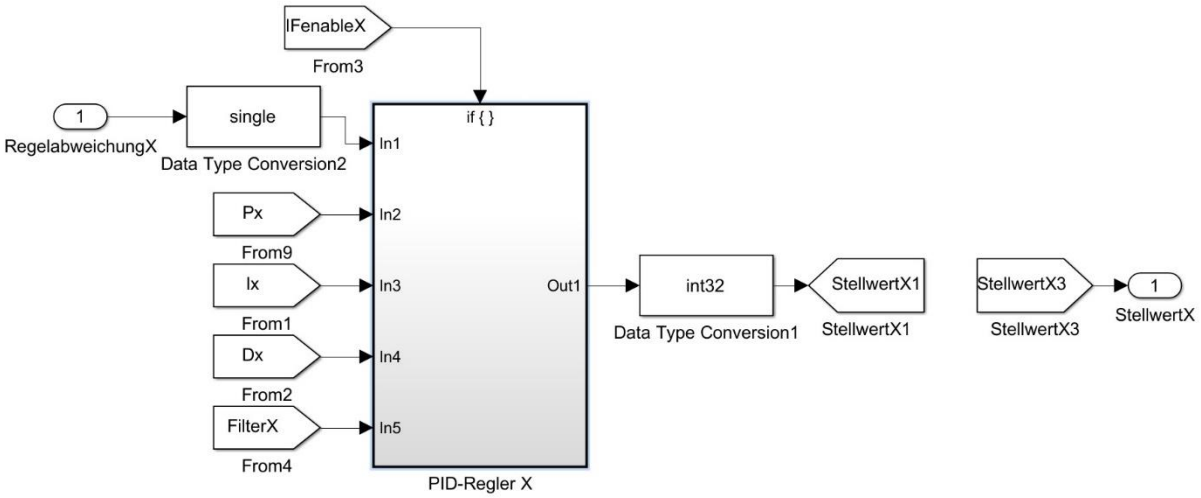


Abbildung 34: Subsystem vom Regler der X-Achse

Im Ausführungsblock befindet sich lediglich der diskrete PID-Regler, dessen Parameter extern über die Signale definiert werden. Somit konnte die Reglereinstellung übersichtlich an der obersten Ebene vom Programm platziert werden. Die verwendeten „From“-Blöcke für die Parametersignale mussten außerhalb des if-Ausführungsblocks platziert werden, da diese in solchen Ausführungsblöcken nicht erlaubt sind. Diese if-Bedingung verbessert das Regelungsverhalten letztendlich dahingehend, dass bei invaliden Positionsmessdaten der Regelungsalgorithmus nicht ausgeführt sondern der vorangegangene Stellwert verwendet wird. Damit können abrupte Stellwertänderungen aufgrund falscher Messdaten vermieden werden, wodurch die Regelung stabiler wird.
Als Regelparameter wird in dem Programm neben den typischen PID-Parametern auch der optionale Filterkoeffizient für den D-Anteil vom Regler verwendet, der besonders bei der hohen Samplerate gut verwendet werden kann, um das Stellwertrauschen zu verringern.
Für den Regelungsalgorithmus wird der Datentyp single verwendet. Der berechnete Stellwert wird anschließen in den Datentyp int32 umgewandelt, weil die Methode für die Servosignalgenerierung nur Integerwerte verarbeiten kann.
Mit den Blöcken „StellwertX1“ und „StellwertX3“ wird der Stellwert für die Realisierung der stellwertmodifizierenden Funktionen der Stellwertstörung und der Sprunggenerierung abgeleitet und wieder zugeführt.

6.2.3 Servoansteuerung

Für die softwareseitige Signalgenerierung der Servomotoransteuerung wird je Achse eine S-Funktion verwendet, wovon beispielhaft die S-Funktion für die X-Achsenservoansteuerung in Abbildung 35

gezeigt ist. Die beiden S-Funktionen sind lediglich Kopien voneinander und damit völlig gleich, obwohl für jede Achse eine angepasste S-Funktion sinnvoller gewesen wäre. Dieser Umstand ist dadurch begründet, dass Simulink nicht mehrere verschiedene S-Funktionen mit Befehlen der verwendeten Servobibliothek zulässt. Um dieses Problem zu lösen wurde der ganze Code für die X- und Y-Servoansteuerung in eine S-Funktion zusammengefasst und Kopien davon verwendet. Mittels einem dafür erstellten Multiplexer wird jeweils der entsprechende Codeabschnitt den beiden Achsen zugewiesen. Realisiert wurde die Funktion des Multiplexers mithilfe einer if-Schleife im Code und einem Signaleingang an der S-Funktion, der in der gezeigten Abbildung als „Mux“ beschriftet zu erkennen ist. Konfiguriert wurde das so, dass eine Konstante mit dem Wert „0“ die if-Bedingung der X-Achse und eine Wert ungleich „0“ die if-Bedingung der Y-Achse entspricht.

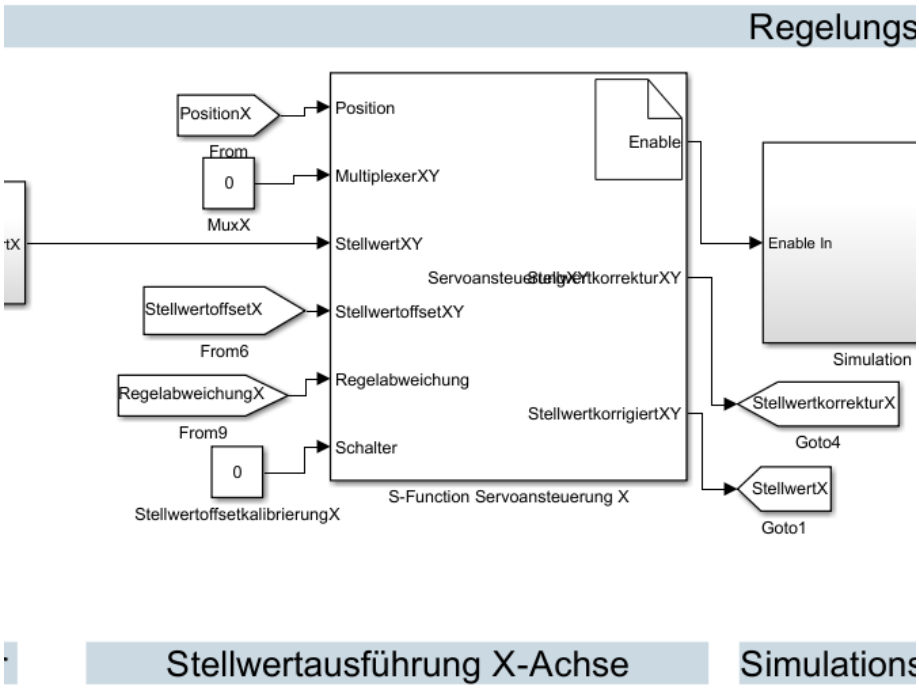


Abbildung 35: S-Funktion für die Servoansteuerung der X-Achse

Gegenüber der S-Funktion vom Testprogramm wurden in dieser S-Funktion neben der Servosignalgenerierung zusätzliche Funktionen inklusive der Spannungsumschaltung integriert, womit das Programm im Gesamten übersichtlich gehalten werden konnte, die S-Funktion selber jedoch dadurch umfangreicher ist.

Insgesamt wurden sechs Signaleingänge und drei Signalausgänge definiert, wobei die Funktion der einzelnen Anschlüsse mit deren Namen ersichtlich ist. Nachfolgend ist der C++-Code der S-Funktion mitsamt Kommentaren aufgelistet, anhand dessen die Funktion der S-Funktion nachvollzogen werden kann. Dieser Code wird anschließend noch erläutert.

Reiter „Libraries“:

```
#ifndef MATLAB_MEX_FILE
#define ARDUINO 100
#include <Arduino.h>
#include "Servo.cpp"
Servo ServoX;
Servo ServoY;
int Signal;                  //Deklaration der
int DiffmaxX = 10;            //benötigten Variablen
int DiffmaxY = 10;
int DiffX;
int DiffY;
int MemX = 1500;
int MemY = 1500;
int ZaehlerX = 0;
int ZaehlerY = 0;
int StellwertoffsetX;
int StellwertoffsetY;
int StellwertkorrekturX = 0;
int StellwertkorrekturY = 0;
int GetriebespielX;
#endif
```

Reiter „Diskrete Update“:

```
if(xD[0]!=1){
    #ifndef MATLAB_MEX_FILE
    analogReadResolution(12);    //Einstellung der A/D-Wandler auf 12 bit
    ServoX.attach(8);            //Festlegen der Pins für die
    ServoY.attach(9);            //Signalausgabe
    pinMode(24, OUTPUT);         //Konfigurierung der
    pinMode(26, OUTPUT);         //digitalen Pins als Spannungsausgänge
    pinMode(28, OUTPUT);
    pinMode(32, OUTPUT);
    pinMode(34, OUTPUT);
    pinMode(36, OUTPUT);
    pinMode(40, OUTPUT);
    pinMode(42, OUTPUT);
    pinMode(44, OUTPUT);
    pinMode(33, OUTPUT);
    pinMode(35, OUTPUT);
    pinMode(37, OUTPUT);
    digitalWrite(33, LOW);       //Schalten auf Ground
    digitalWrite(35, LOW);
    digitalWrite(37, LOW);
    #endif
xD[0]=1;}
```

Reiter „Outputs”:

```
#ifndef MATLAB_MEX_FILE
Signal = *StellwertXY;
if (*MultiplexerXY == 0){          //Multiplexer-Bedingung für X-Achse
    StellwertoffsetX = *StellwertoffsetXY;
//Mechanismus für die automatische Stellwertkalibrierung an der X-Achse
//falls Schalter auf 1
    if (*Schalter == 1){
        ZaehlerX = ZaehlerX + 1;
        if (ZaehlerX > 200){
            if (*Regelabweichung < 0.05){
```

```
                ZaehlerX = 0;
                StellwertkorrekturX = StellwertkorrekturX + 1;}
            else if (*Regelabweichung > 0.05){
                ZaehlerX = 0;
                StellwertkorrekturX = StellwertkorrekturX - 1;}}
        *StellwertkorrekturXY = StellwertkorrekturX;
        StellwertoffsetX = *StellwertoffsetXY + StellwertkorrekturX;}
//Positionsabhängige Korrektur des Stellsignals für den Ausgleich
//des Getriebespiels auf der X-Achse
    if (*Position > 22){
        GetriebespielX = 24;
        Signal = -Signal + 24 + StellwertoffsetX;}
    else if (*Position < 8){
        GetriebespielX = 0;
        Signal = -Signal + StellwertoffsetX;}
    else {GetriebespielX = (*Position - 8) * 1.714;
        Signal = -Signal + GetriebespielX + StellwertoffsetX;}
//Einschränkung der maximalen Stellwertänderungsgeschwindigkeit
//der X-Achse mit dem Wert der Variable DiffmaxX
    DiffX = Signal - MemX;
    if (DiffX > DiffmaxX){
        MemX = MemX + DiffmaxX;
        Signal = MemX;}
    else if (DiffX < -DiffmaxX){
        MemX = MemX - DiffmaxX;
        Signal = MemX;}
    else {MemX = Signal;}
//Begrenzung des Stellwerts der X-Achse
    if (Signal > 2050){
        Signal = 2050;
        MemX = 2050;}
    if (Signal < 970){
        Signal = 970;
        MemX = 970;}
//Berechnung und Ausgabe des effektiven Stellwerts der X-Achse für Scopes
    *StellwertkorrigiertXY = -Signal + StellwertoffsetX + GetriebespielX;
//Stellwertmodifikation für angleich X/Y-Achse mit Schrittweite 0,027°
    Signal = ((Signal - *StellwertoffsetXY) * 0.8) + *StellwertoffsetXY;
//Stellwertausgabe X-Achse
    ServoX.writeMicroseconds(Signal);
//Umschalten der Spannungen des Touchscreens für eine anschließende Messung
//der Position an der Y-Achse
    digitalWrite(24, HIGH);
    digitalWrite(26, HIGH);
    digitalWrite(28, HIGH);
    digitalWrite(32, LOW);
    digitalWrite(34, LOW);
    digitalWrite(36, LOW);
    digitalWrite(40, HIGH);
    digitalWrite(42, HIGH);
    digitalWrite(44, HIGH);}
//Anweisungen für Servo Y-Achse
else{
    StellwertoffsetY = *StellwertoffsetXY;
//Mechanismus für die automatische Stellwertkalibrierung an der Y-Achse
//falls Schalter auf 1
    if (*Schalter == 1){
        ZaehlerY = ZaehlerY + 1;
        if (ZaehlerY > 200){
            if (*Regelabweichung < 0.05){
                ZaehlerY = 0;
                StellwertkorrekturY = StellwertkorrekturY - 1;}
```

```
            else if (*Regelabweichung > 0.05){
                ZaehlerY = 0;
                StellwertkorrekturY = StellwertkorrekturY + 1;}}
        *StellwertkorrekturXY = StellwertkorrekturY;
        StellwertoffsetY = *StellwertoffsetXY + StellwertkorrekturY;}
    Signal = StellwertoffsetY + Signal;  //Berechnung des Gesamtstellwerts
//Einschränkung der maximalen Stellwertänderungsgeschwindigkeit
//der Y-Achse mit dem Wert der Variable DiffmaxY
    DiffY = Signal - MemY;
    if (DiffY > DiffmaxY){
        MemY = MemY + DiffmaxY;
        Signal = MemY;}
    else if (DiffY < -DiffmaxY){
        MemY = MemY - DiffmaxY;
        Signal = MemY;}
    else {MemY = Signal;}
//Begrenzung des Stellwerts der Y-Achse
    if (Signal > 2000){
        Signal = 2000;
        MemY = 2000;}
    if (Signal < 1000){
        Signal = 1000;
        MemY = 1000;}
//Berechnung und Ausgabe des effektiven Stellwerts der Y-Achse für Scopes
    *StellwertkorrigiertXY = Signal - StellwertoffsetY;
//Stellwertausgabe Y-Achse
    ServoY.writeMicroseconds(Signal);
//Umschalten der Spannungen des Touchscreens für eine anschließende Messung
//der Position an der X-Achse
    digitalWrite(24, HIGH);
    digitalWrite(26, HIGH);
    digitalWrite(28, HIGH);
    digitalWrite(32, HIGH);
    digitalWrite(34, HIGH);
    digitalWrite(36, HIGH);
    digitalWrite(40, LOW);
    digitalWrite(42, LOW);
    digitalWrite(44, LOW);}
*Enable = 0;    //Signalausgang für Programmablauffestlegung
#endif
```

Die obere Hälfte vom Code des Reiters „Outputs“ für die X-Achse und die untere Hälfte für die Y-Achse sind weitgehend gleich aufgebaut und werden wie bereits beschrieben per if-Bedingung an den entsprechenden Stellen im Programm ausgeführt. Danach ist ein Mechanismus angesetzt, der die Abweichung des Stellwinkels gegenüber der Horizontalen automatisch ermittelt. Durch Korrektur des Stellwertoffsets mit der Stellwinkelabweichung kann das Touchscreen relativ leicht zur Horizontalen kalibriert werden. Um die Kalibrierung zu starten, muss lediglich die Konstante „Offsetkalibrierung“ in Simulink während des Betriebs mit dem Schalter im Bedienungsfeld zu eins gesetzt werden und ein fester Wert als Sollwert eingestellt sein. Der Mechanismus nutzt bei der Festwertregelung die Regelungsabweichung und verringert diese durch Stellwertoffsetänderung bis auf ein Minimum, wie es am gezeigten Code nachvollzogen werden kann.
Mit dem darauf folgenden Codeabschnitt wird das Getriebespiel der X-Achse mit einer positionsabhängig berechneten Stellwertkorrektur verringert und somit das Regelungsverhalten

verbessert. Vor allem die X-Achse zeigt ein Winkelspiel von ca. 0,5° auf, welches durch das Getriebespiel des Servomotors hervorgerufen wird und das Regelungsverhalten leicht beeinträchtigt. Die Y-Achse weist dagegen weniger Spiel auf, was an der einseitigen Gewichtsverteilung aufgrund des angekoppelten Servomotors zurückzuführen ist. Deshalb wurde für die Y-Achse keine Korrektur vorgenommen. An der X-Achse wirkt sich das Spiel im statischen Fall in Abhängigkeit der Kugelposition annähernd linear auf den tatsächlichen Stellwert aus. Mit einer im Code nachvollziehbaren, kugelpositionsabhängigen Korrektur des Stellwerts konnte die Auswirkung des Getriebespiels auf den tatsächlichen Stellwert verringert werden. Die Gewichtsverteilung des gelagerten Touchscreens ist hierbei genau ausgeglichen, sodass der Belastungsumkehrpunk des Servomotorgetriebes genau in der Mitte der Platte liegt. An diesem Umkehrpunkt ist die Auswirkung des Getriebespiels auf den tatsächlichen Stellwert unterschiedlich und deutlich stärker von der Dynamik des Stellwerts und der Kugel abhängig, weshalb die Korrektur des Stellwerts an dieser Stelle nicht zum erwünschten Ergebnis führt. Dennoch konnte mit dieser Maßnahme die negativen regelungstechnischen Auswirkungen des Getriebespiels, das ohnehin verhältnismäßig klein ist, auf ein geringes Maß reduziert werden.

Mit dem darunter anschließenden Codeabschnitt wird die maximale Stellwertänderungsgeschwindigkeit künstlich begrenzt, wobei die Variable „DiffmaxX/Y“ die maximale Stellwertänderung zum vorherigen Wert beschreibt. Die Begrenzung verhindert das Abheben der Kugel bei zu hohen Stellwertgeschwindigkeitsänderungen und verringert darüber hinaus ruckartige Stellwertänderungen, die das System im Extremfall instabil werden lassen.

Im Code unterhalb der Begrenzung der Stellwertänderungsgeschwindigkeit werden die maximal sinnvollen und von den Servomotoren und der Mechanik anfahrbaren Stellwertgrenzen festgelegt, sodass keine ungültigen Stellwertsignale auf die Servomotoren gegeben werden.

Bei der X-Achse ist das Stellwertsignal mit der Multiplikation von 0,8 so angepasst, dass beide Achsen einen einheitlichen Schrittwinkel von 0,027°/Stellschritt aufweisen. Erst danach ist die eigentliche Funktion für die Servosignalgenerierung, die im Testprogramm schon beschrieben wurde, angesetzt.

Da der vom Regelungsalgorithmus berechnete Stellwert in dieser S-Funktion mit dem systemspezifischen Stellwertoffset und dem Getriebespielausgleich modifiziert wird, kann dieser nicht sinnvoll als Vergleich für simulierte Stellwerte verwendet werden. Deshalb wird, wie im Code ersichtlich, für die Verwendung des Stellwerts zu Vergleichszwecken ein bereinigter Stellwert berechnet und für die Weiterverwendung einem Ausgang der S-Funktion zugewiesen.

Als letztes wird im jeweiligen Code für die X- und die Y-Achse die Spannungsausgabe für das Touchscreen geändert. Im Code für die X-Achse wird die Spannungsausgabe auf die Messung der Position der Kugel in Y-Richtung konfiguriert. Entsprechendes gilt auch für den Code der Y-Achse. Die Spannungsumschaltung konnte nicht mehr, wie zuvor im Testprogramm, mit Blöcken realisiert

werden, sondern wurde nun in die S-Funktion eingebaut. Der Grund dafür ist, dass Simulink von den „Digital Out“ pro Pin auch nur ein Block zulässt und bei dieser Programmstruktur aber jeweils zwei Blöcke mit der gleichen Pinnummer gebraucht werden würden.
Zu ergänzen ist, dass die Servobibliothek das Impulssignal lediglich mit der standardmäßigen Wiederholfrequenz von 50 Hz generiert, das Regelungsprogramm aber auf 333 Hz ausgelegt wurde, das an der maximal verwendbaren Frequenz der verwendeten Savöx Servomotoren angelehnt ist. Die Bibliothek stellt keine Funktionen zur Verfügung, mit der die Frequenz verändert werden könnte. Die Analyse der Bibliothek selbst hat aber ergeben, dass die Frequenz dennoch relativ leicht durch die Manipulation einer Definition im Quellcode der Bibliothek festgelegt werden kann. Die entsprechende Definition steht in der Headerdatei „Servo.h“ in der Zeile 75. Der Zahlenwert wurde hier von 20000, was 50 Hz entspricht, auf 3000 verringert, wie es unterhalb gezeigt ist:

```
#define REFRESH_INTERVAL    3000
```

Damit arbeitet das Regelungsprogramm und die Signalausgabe gleich schnell wie die Servomotoren, wodurch jeder berechnete Stellwert durch einen Impuls auch auf die Servomotoren ausgegeben wird.

6.2.4 Simulationsmodell

Als Anforderung an das Tabledance wurde unter anderem definiert, dass die Werte der Regelungsgrößen zusammen mit den Werten der Simulation in Echtzeit angezeigt werden können sollen, anstatt diese, wie zuvor, nachträglich in den Plots einzufügen. Um dies zu erreichen, musste das Simulationsmodell in das Regelungsprogramm integriert werden. Wie schon erwähnt, wurde das in einem Subsystem zusammengefasste Simulationsmodell im Programmablauf direkt nach der S-Funktion für die X-Achsenservoansteuerung angesetzt, da der Mikrocontroller für die anschließende Messung der Kugelposition an der Y-Achse ansonsten länger warten müsste. Das Simulationsmodell wurde so entwickelt und konfiguriert, dass es genau die gleichen Eigenschaften wie das Regelungsprogramm selbst aufweist, nur dass die Regelungsstrecke durch ein mathematisches Modell ersetzt wurde. Die Abbildung 36 zeigt das gesamte Simulationsmodell, an dem die beiden Regelungsschleifen für die X- und Y-Achse leicht zu erkennen sind.

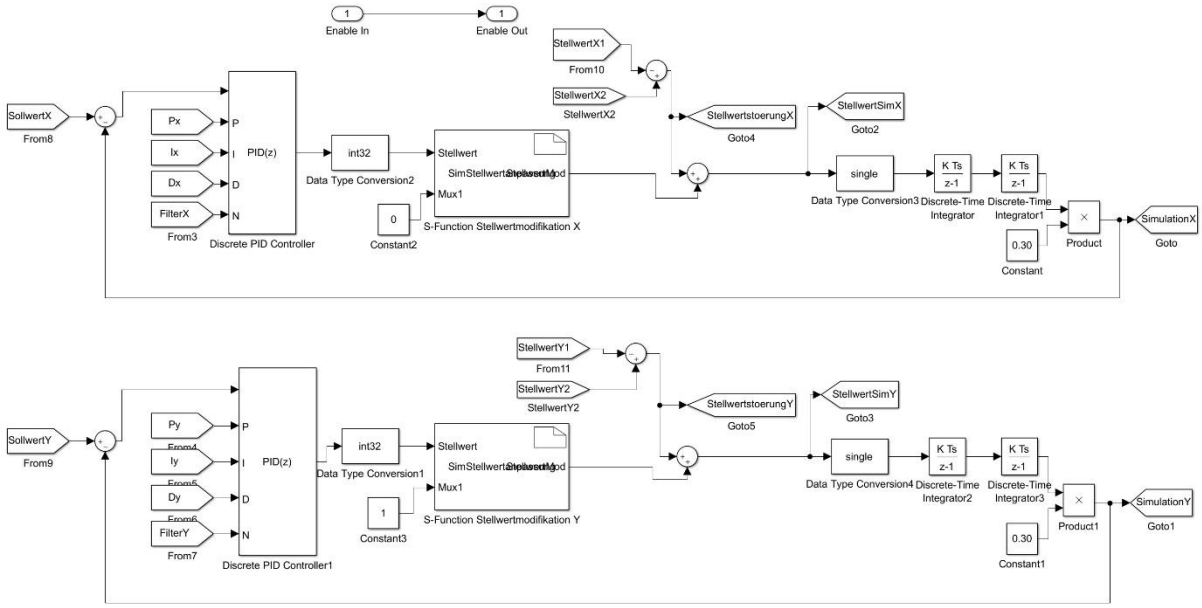


Abbildung 36: Simulationsmodell für die X- und Y-Achse

Der Sollwert für die Simulation wird direkt vom Sollwert für die Regelung abgeleitet. Darauf folgt die Berechnung der Regelungsabweichung mit anschließender Berechnung des Stellwerts über einen zur Hauptregelungsschleife gleichwertigen PID-Reglers, der ebenfalls extern mit den gleichen Signalen parametriert ist. Dadurch wird bei der Eingabe der Regelungsparameter an der Benutzeroberfläche gleichzeitig die Parameter der Simulation und der echten Regelungsschleife geändert, wodurch diese Parameter nicht doppelt eingegeben werden müssen.

Damit sich das simulierte System gleich wie das echte verhält, wurden die relevanten digitalen Nachbearbeitungen des Stellwerts auch bei der Simulation mittels einer S-Funktion implementiert. Zu den relevanten Stellwertmodifikationen zählen die Begrenzung der maximalen Stellwertänderungsgeschwindigkeit und die Begrenzung der Stellwerte. Diese S-Funktion wird aufgrund der Ähnlichkeiten zur bereits beschriebenen S-Funktion nicht weiter erläutert.

Durch das Addieren des Stellwerts mit der Stellwertstörung berücksichtigt das Simulationsmodell auch die Funktion der optional zuschaltbaren Stellwertstörung.

Nach der Stellwertmodifikation ist das eigentliche Modell der Regelungsstrecke angesetzt, bevor das Signal anschließend zur Regelungsabweichungsberechnung rückgekoppelt wird und die Regelungsschleife somit geschlossen ist. Das Regelungsstreckenmodell wurde bereits mit der vorangegangenen Projektarbeit hergeleitet und analysiert. Für die Herleitung des Modells, das nachfolgend in Gleichung (1) gezeigt ist, sei deshalb auf die vorangegangene Projektarbeit verwiesen.

$$Gs(s) = \frac{5}{7} * g * \frac{1}{s^2} \quad (1)$$

Wie im Simulinkmodell zu erkennen ist, wird der Stellwert, äquivalent zum Modell, zweimal mit einem Block für diskrete Integration integriert und anschließend mit einer berechneten Konstante multipliziert. Das Ergebnis ist die Position der Kugel.
Der Wert der Konstante in der Modellgleichung gilt nur für die Kugelposition in Metern und den Neigungswinkel im Bogenmaß, weswegen dieser mit je ein Umrechnungsfaktor für die Positionsmessung (2) und für das Stellglied (3) modifiziert werden musste. Die Modellgleichung mit der daraus berechneten Konstante ist in Gleichung (4) dargestellt. Diese Gleichungen gelten sowohl für die X-Achse als auch für die Y-Achse, weil die Positions- und Stellwerte durch die vorgenommenen Umrechnungen die gleichen Einheiten haben. Dabei wurde die Positionsangabe in Zentimetern und die Stellwertangabe in 0,027°/Stellschritt definiert.

$$\frac{Bildpunkte}{Länge} = \frac{1\ cm}{0{,}01\ m} = 100\ \frac{cm}{m} \quad (2)$$

$$Schrittwinkel = \frac{Drehwinkel}{Stellschritte} = \frac{23{,}5 * \pi}{875 * 180} \approx 0{,}027° \quad (3)$$

$$Gs(s) = \frac{5}{7} * 100\ \frac{cm}{m} * \frac{23{,}5° * \pi}{875 * 180} * 9{,}81\ \frac{m}{sec^2} * \frac{1}{s^2} \approx \mathbf{0{,}328}\ \frac{\boldsymbol{cm}}{\boldsymbol{sec^2}} * \frac{\mathbf{1}}{\boldsymbol{s^2}} \quad (4)$$

Das Simulationsmodell arbeitet genau wie die Regelung selbst mit einer Samplerate von 333 Hz und diskreten Zuständen, wodurch eine absolute Gleichheit bei der digitalen Verarbeitung vorliegt. Das Streckenmodell selber, das mathematisch kontinuierlich beschrieben ist, wird in diesem Simulationsmodell zwangsweise auch diskret berechnet, wodurch Ungenauigkeiten entstehen können. Bei der hohen Samplerate ist der Fehler durch die Diskretisierung im Vergleich zur kontinuierlichen Berechnung der Strecke nur sehr gering und kann daher problemlos vernachlässigt werden.

6.3 Struktur und Programmierung der Bedienungsoberfläche

6.3.1 Sollwertvorgabe

Die Sollwertvorgabe ist im Bereich der Bedienungsoberfläche an der linken Seite untergebracht. Insgesamt wurden fünf verschieden Sollwerte zur Auswahl in das Programm eingebaut. Diese können entsprechend mit den Umschaltblöcken ausgewählt werden. Zum Umschalten des Signalflusses und damit des Sollwerts müssen diese nur per Doppelklick angeklickt werden. Der Bereich im Programm für die Sollwertvorgabe ist in Abbildung 37 gezeigt, in dem auch die Blöcke zum Umschalten der

Signale erkennbar sind. Auswählbar sind die fünf Sollwerte Konstante, Sinus, Rechteck, aufgenommene Kurve, und Sollwertvorgabe per Joystick. Dabei kann der Sollwert für jede Achse unabhängig von der anderen ausgewählt werden, wodurch sich viele Kombinationsmöglichkeiten ergeben. Dabei gilt, dass z.B. für einen Kreis als Sollwert jeweils ein Sinussignal an der X- und Y-Achse gewählt werden muss. Die beiden Sinussignale sind bereits mit der entsprechenden Phasenverschiebung konfiguriert, sodass sich bei Verwendung des Sinussignals an beiden Achsen ein Kreis ergibt. Entsprechendes gilt auch für das Rechtecksignal.

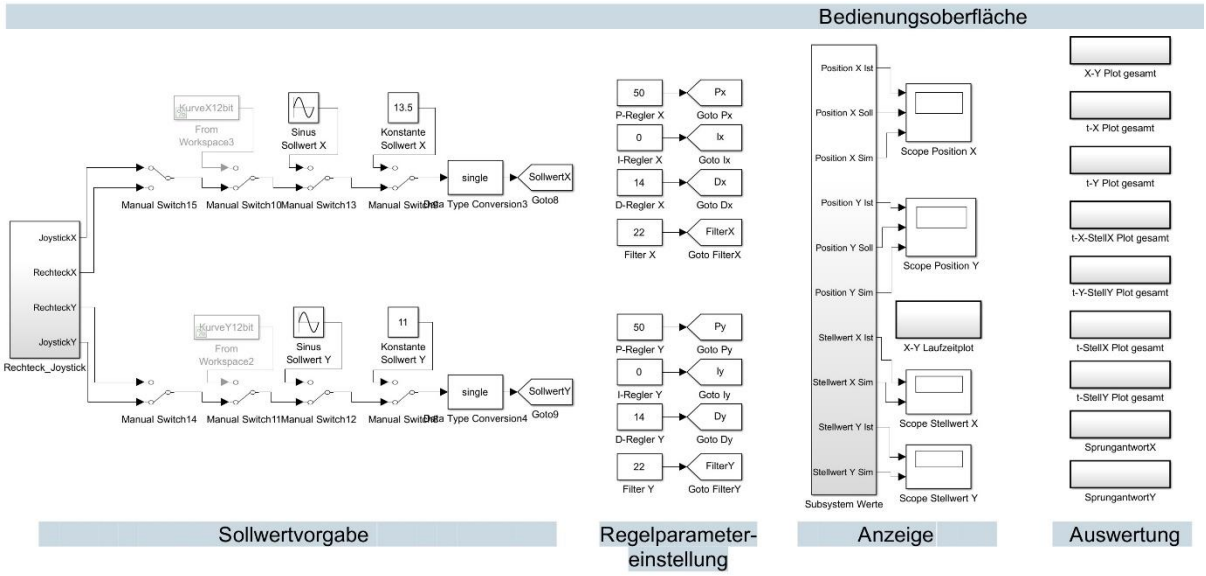


Abbildung 37: Linke Seite der Bedienungsoberfläche

Die drei Sollwertvorgaben Konstante, Kreis und Rechteck wurden auch am alten Tabledance realisiert, wogegen die beiden anderen Sollwerte neu sind. Mit dem Block „KurveX/Y12bit“ kann eine zuvor am Touchscreen aufgezeichnete beliebige Kurve als Sollwert vorgegeben werden. Die Kurve wird mit einem speziell dafür entwickelten, separaten Programm aufgenommen. Die maximale Aufnahmelänge ist aufgrund des geringen Speichers des Mikrocontrollers auf ca.4 Minuten begrenzt. Mit der Inkaufnahme einer geringeren Kurvenabtastrate lässt sich diese Zeit auch deutlich anheben.

Bei dem anderen neuen Sollwert, Joystick genannt, kann der Sollwert über einen kleinen Fingerjoystick manuell für die X- und die Y-Achse vorgegeben werden. Dieser Joystick, der in Abbildung 38 zu sehen ist, ist fest am Tabledance angeschraubt und über fünf Kabelleitungen mit dem Mikrocontroller verbunden. Bei diesem Joystick handelt es sich um ein analoges Modell mit zwei 10 kΩ Drehpotentiometern, jeweils einer zum Abbilden der X- und Y-Achse, und einem Taster, der digital

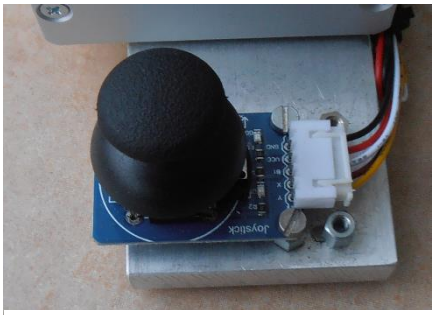


Abbildung 38: Analoger Fingerjoystick

ausgewertet werden kann. Die Tasterfunktion wird nicht verwendet, weshalb auch nicht näher darauf eingegangen wird. Gebraucht werden daher letztendlich nur vier Kabel, zwei für die Spannungsversorgung und jeweils eines für das Potentiometersignal der X- und Y-Achse. Die Spannungsversorgung wurde mangels nativer Spannungsanschlüsse mit einem digital arbeitenden Pin bereitgestellt. Die maximal spezifizierten 9mA pro Pin werden aufgrund des hohen Minimalwiederstands der Drehpotentiometer nicht überschritten, weswegen ein Pin ausreichend ist. Die beiden analogen Drehpotentiometersignale vom Joystick sind zur Auswertung mit den integrierten 12 bit A/D-Wandler vom Arduino verbunden. Dadurch, dass es sich um einen analogen Joystick handelt, kann nicht nur die Richtung sondern auch die Geschwindigkeit der Sollwertänderung präzise mit dem Finger vorgegeben werden.

Programmtechnisch ist die Funktion der Sollwertvorgabe des Joysticks und des Rechtecks neben normalen Simulinkblöcken zusätzlich mit einer S-Funktion realisiert worden, obwohl auch die von Simulink bereitgestellten Blöcke ausgereicht hätten. Eine Umsetzung mit graphischen Blöcken alleine wäre in diesem Fall aber sehr umständlich und unübersichtlich gewesen, weshalb eine S-Funktion zu Hilfe genommen wurde. Dazu zeigt die Abbildung 39 den Inhalt des Subsystems „Rechteck_Joystick“, an dem die gleich benannte S-Funktion erkennbar ist. An der linken Seite werden die analogen Drehpotentiometersignale erfasst und über vier Werte gemittelt. Dadurch wird das Messsignalrauschen verringert, wodurch auch das daraus generierte Sollwertsignal gleichmäßiger ist. Neben der Sollwertgenerierung wird in diesem Subsystem auch das Störsignal für die optionale Funktionalität der Stellwertstörung generiert. Wie an der rechten unteren Seite des Subsystems zu erkennen ist, wird das vom Messsignal des Joysticks innerhalb der S-Funktion abgeleitete Stellwertstörsignal mit der Addition des Stellwerts vom Regler zum Gesamtstellwertstörsignal „JoystickstellX/Y“ fertiggestellt.

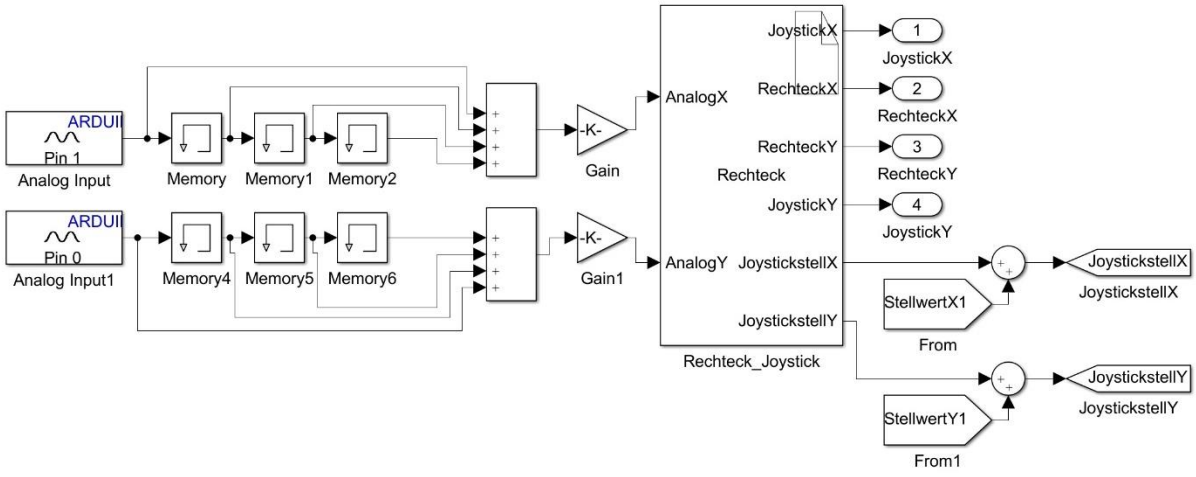


Abbildung 39: Subsystem „Rechteck_Joystick“

Die S-Funktion erhält die gemittelten 12 bit Drehpotentiometersignale als int16 Signale und ist folgend aufgebaut:

Reiter „Libraries“:

```
#ifndef MATLAB_MEX_FILE
#define ARDUINO 100
#include <math.h>
#include <Arduino.h>
float x = 6;
float y = 6;
float AnfangX = 6;
float EndeX = 27;
float AnfangY = 6;
float EndeY = 18;
float Delta = 0.02;
float Xmax = 30;
float Ymax = 21;
float Xmin = 3;
float Ymin = 3;
float Xjoystick = 16.5;
float Yjoystick = 12.1;
float AnalogXmem;
float AnalogYmem;
#endif
```

Reiter „Diskrete Update“:

```
if(xD[0]!=1){
#ifndef MATLAB_MEX_FILE
pinMode(48, OUTPUT);
digitalWrite(48,HIGH);    //zur Spannungsversorgung vom Joystick
#endif
xD[0]=1;}
```

Reiter „Outputs“:

```
#ifndef MATLAB_MEX_FILE
// Algorithmus für die Rechtecksollwertgenerierung mit der
//Schrittweite Delta und den Grenzen AnfangX, EndeX, AnfangY und EndeY
if (x <= EndeX & y <= AnfangY){
    x = x + Delta;}
else if (x > EndeX & y < EndeY){
    y = y + Delta;}
else if (x >= AnfangX & y >= EndeY){
    x = x - Delta;}
else {y = y - Delta;}
//Ausgabe auf Ausgangssignal der S-function
*RechteckX = x;
*RechteckY = y;

//Joysticksollwertberechnung aus Joysticksignal AnalogX und AnalogY
if (*AnalogX > 2040 || *AnalogX < 2020){
    AnalogXmem = *AnalogX;}
else {AnalogXmem = 2030;}
if (*AnalogY > 2040 || *AnalogY < 2020){
    AnalogYmem = *AnalogY;}
else {AnalogYmem = 2030;}
Xjoystick = (AnalogXmem - 2030) * -0.000025 + Xjoystick;
```

```
Yjoystick = (AnalogYmem - 2030) * 0.000025 + Yjoystick;
if (Xjoystick > Xmax){
    Xjoystick = Xmax;}
else if (Xjoystick < Xmin){
    Xjoystick = Xmin;}
if (Yjoystick > Ymax){
    Yjoystick = Ymax;}
else if (Yjoystick < Ymin){
    Yjoystick = Ymin;}
//Ausgabe auf Ausgangssignal der S-function
*JoystickX = Xjoystick;
*JoystickY = Yjoystick;
//Berechnung und Ausgabe der Stellwertstörung
*JoystickstellX = (AnalogXmem - 2030) * -0.2;
*JoystickstellY = (AnalogYmem - 2030) * 0.2;
#endif
```

Der obere Codeabschnitt vom Reiter „Outputs“ ist für die Rechtecksollwertgenerierung, während der untere Abschnitt das Sollwertsignal vom Joysticksignal ableitet. Mit dem Wert der Variable „Delta“, die die Schrittweite pro Takt angibt, kann die Geschwindigkeit der Rechtecksollwertvorgabe eingestellt werden. Die Grenzen des Rechtecks sind mit den Variablen „AnfangX“, „EndeX“, „AnfangY“ und „EndeY“ einstellbar. Mit dieser Art der Umsetzung konnte ein kontinuierliches und gleichmäßiges Rechtecksignal generiert werden, das mit dem zuvor genutzten „Pulse Generator“-Simulinkblock nicht möglich war.
Das Sollwertsignal „Joystick“ wird aus den Messwerten der Drehpotentiometer und den davon abgeleiteten Variablen „AnalogX“ und „AnalogY“ berechnet. Da die Signale ein Messrauschen aufweisen und der Sollwert bei ungenutzten Joystick nicht verändert werden soll, wurde um den Ausgangspunkt, der bei 2030 ist, ein Bereich festgelegt, bei dem das Sollwertsignal nicht geändert wird. Für die Berechnung der Sollwertänderung wird der 32 bit Gleitkommadatentyp float verwendet, weil Ganzzahldatentypen hier zu ungenau wären. Der verwendete Skalierungsfaktor 0,0025 gibt an, wie schnell sich der Sollwert relativ zum Drehpotentiometermesswert ändern soll und kann je nach Bedarf angepasst werden. Danach ist die Begrenzung der minimalen und maximalen Sollwerte angesetzt, während die S-Funktion mit der Berechnung des Stellwertstörsignals abschließt.

6.3.2 Regelparametereinstellung

Die Regelparametervorgabe ist direkt neben der Sollwertvorgabe platziert. Neben den typischen Regelungsparametern P, I und D kann auch ein Filterkoeffizient, jeweils für die X- und Y-Achse, definiert werden, wie es auch in der Abbildung 40, bei der die Regelungsparametereinstellung gezeigt ist, ersichtlich ist. Diese Parameter können auch während der Laufzeit geändert werden. Die hier verwendete Filteroption mittelt vorangegangene Messwerte, sodass das daraus gewonnene

Signal weniger rauschen aufweist. Dabei gilt: Je kleiner der Koeffizient, umso mehr vorangegangene Werte werden in die Mittelung einbezogen. Das gefilterte Signal wird nur für den D-Anteil im Regler angewendet, weil dieser besonders empfindlich auf das Messrauschen reagiert und folglich das meiste Stellwertrauschen verursacht. Höhere D-Parameter als auch steigende Sampleraten von der Regelung verstärken die Auswirkung des Messwertrauschens auf das Stellwertrauschen. Mit der Filterung kann diesem Effekt entgegengesteuert werden. Die Filterung selber verursacht wiederrum eine Latenz im gefilterten Signal zum eigentlichen Positionssignal, womit bei der ansonsten latenzarmen Regelung eine künstliche Latenz im D-Regler eingebaut wird. Deshalb sollte beim Betrieb der Filterkoeffizient so groß wie möglich gewählt werden, um die Latenz möglichst gering zu halten und dennoch ein akzeptables Stellwertrauschen zu erreichen.

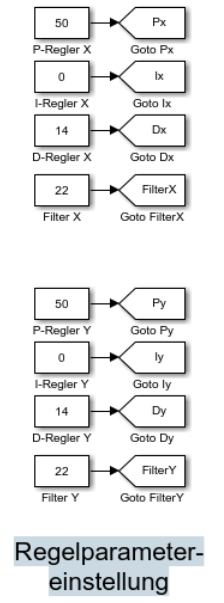


Abbildung 40: Blöcke für Regelungsparameter

6.3.3 Visualisierung

Die Abbildung 41 zeigt den Programmabschnitt „Anzeige“, bei dem Blöcke für die Visualisierung der Regelungssystemgrößen während der Laufzeit zusammengefasst sind. Darunter sind vier konfigurierte Scopes und ein Laufzeitplot, der mit Matlabcode realisiert wurde. Je ein Scope für die X- und Y-Achse zeigt die Ist-, Soll- und Simulationswerte der Kugelposition in Abhängigkeit der Zeit an. Die zwei anderen Scopes wurden für die Visualisierung der Ist-Stellwerte mit den dazugehörigen simulierten Stellwerten in Abhängigkeit der Zeit erstellt.

Für die Übersichtlichkeit an der Bedienungsoberfläche wurden die Signale für die Scopes und des Graphen über ein Subsystem, das hier „Subsystem Werte“ genannt wurde, bereitgestellt. Neben der Bereitstellung der entsprechenden Signale ist auch die Aufzeichnung der Messwerte mittels einem „To Workspace“-Block in dem Subsystem realisiert. Hierbei sei zu erwähnen, dass die aufgenommenen Messwerte erst nach dem stoppen des Regelungsprogramms in den Workspace von Matlab geschrieben werden und somit keine Auswertung der Messwerte während des Programmablaufs möglich ist. Die Variable der Messwerte wurde hierbei „PositionsdatenXY“ genannt und beinhaltet zwölf Signale, die zuvor über ein „Mux“-Block zusammengefasst sind. Sinnvoll ist das Zusammenfassen besonders, weil damit

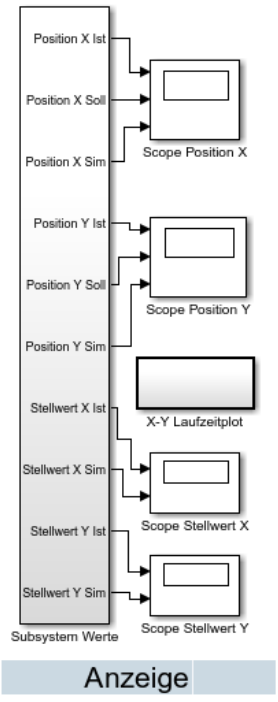


Abbildung 41: Blöcke zur Visualisierung

nur eine Zeitvariable erstellt werden muss, wodurch das Transfervolumen verringert wird. Es vereinfacht zudem die Auswertung der aufgenommenen Werte und verringert auch die Rechenleistung für den „Laufzeitplot XY“.
In Simulink war es nicht möglich, mehrere Linien mit den XY-Graphen gleichzeitig anzuzeigen, sodass der Istwert, Sollwert und der simulierte Positionswert nicht in einem X-Y-Graphen verglichen werden können. Über eine Implementierung mit Matlabcode konnte das Problem gelöst werden und ein Plot, „Laufzeitplot XY“ genannt, in der gewünschten Form erstellt werden. Dieser zeigt die Ist-, Soll-, und simulierten Positionsverläufe der Kugel an, wobei die Linien nach einer bestimmten Zeit verschwinden. Dies verhindert, dass der Plot durch zu lange Linien unübersichtlich wird und der Leistungsverbrauch beim Laptop während der Programmlaufzeit nicht durch immer längere Linien dauerhaft ansteigt.
Als Ausgangspunkt zum Erstellen des Plots wurden sogenannte „Callback functions“ verwendet. Callback functions in Simulink sind Funktionen, die bei gewissen Aktionen an einem Block aufgerufen werden, sofern entsprechender Matlabcode dem Block hinterlegt wurde. Es gibt verschiedene Callback functions, wovon in dieser Arbeit ausschließlich die „OpenFcn“ verwendet wurde, die bei einem Doppelklick auf dem Block aufgerufen wird. Als Grundlage wurde hier ein Subsystemblock, der selbst keinen Einfluss auf das Programm hat, verwendet. Mit Rechtsklick auf den Block und dann Properties=>Callbacks=>OpenFcn kann das Fenster aufgerufen werden, bei dem der Matlabcode integriert wurde. Die Callback functions funktionieren sowohl während das Programm ausgeführt wird als auch wenn das Programm gestoppt ist.
Der Code in den Callback functions wird vom Hostgerät, also dem Laptop, ausgeführt, wodurch dem Arduino keine Rechenbelastung durch die Callback function zukommt. Für die Übersichtlichkeit wurde der Code so wenig wie möglich auf externe Funktionen ausgelagert. Im gesamten besteht der Plot aus dem Matlabcode im OpenFcn-Fenster und einer Funktion, die mit dem „Event Listener“ aufgerufen wird. Der „Event Listener“ stellt den Kern des Codes dar, denn dieser kann die Daten eines Simulinkblocks während der Laufzeit abgreifen und für weitere Berechnungen bereitstellen. Wie schon erwähnt, geht das z.B. mit dem simulinkeigenem „To workspace“-Block nicht während der Laufzeit des Regelungsprogramms. Der Event Listener funktioniert grundsätzlich mit den meisten Blöcken von Simulink. Da das Regelungsprogramm hier im Modus „External“ auf dem Arduino läuft, ist die Nutzbarkeit des Event Listeners dennoch ausschließlich auf Simulinkblöcke beschränkt, die Daten vom Arduino zum Laptop übertragen. Das ist dadurch begründet, dass der Matlabcode und der Event Listener vom Hostgerät ausgeführt werden und die abzugreifenden Daten auch dort vorhanden sein müssen. Verwendet wurde letztendlich der „To workspace“-Block „Messwerte“, der alle nötigen Signale auf einem Block gebündelt hat. Dadurch ist in diesem Fall nur ein Event Listener

nötig. Nachfolgend ist der kommentierte Matlabcode der OpenFcn aufgeführt, bei dem der Event Listener zum Schluss aufgerufen wird:

```
%neues Fenster
hFig = figure(1);
%Erstellung global gültiger Vektoren mit der Länge 150 und dem
%Datentyp single
%Diese Vektoren dienen als Messwertspeicher in der Funktion
plot_adapt_coefs2
global X1;
global Y1;
global X2;
global Y2;
global X3;
global Y3;
X1 = ones(1,150,'single');
Y1 = ones(1,150,'single');
X2 = ones(1,150,'single');
Y2 = ones(1,150,'single');
X3 = ones(1,150,'single');
Y3 = ones(1,150,'single');
%Initialisierung des Plots, dessen Werte  mit der Funktion
plot_adapt_coefs2
%während der Laufzeit angepasst werden
plot1 = plot(X1, Y1, 'b', X2, Y2, '-.r', X3, Y3, '--g', 1, 1, 'b*', 1, 1,
'r*', 1, 1, 'g*');
%Beschriftung und Konfigurierung des Plots
axis ([-5 38 0 24]);
title('Kugelpositionsverlauf');
xlabel('Kugelposition X in [cm]');
ylabel('Kugelposition Y in [cm]');
legend('Istwert','Sollwert','Simulation');
grid on;
%Der Handle plot1 wird als "UserData" dem Simulinkblock "Messdaten"
%angehängt
%Dadurch kann in der Funktion plot_adapt_coefs2 auf dem Handle und somit
%auf dem hier erstellten Plot zugegriffen werden
set_param('Arduino12bit_Tabledance/Subsystem
Werte/Messdaten','UserData',plot1);
%Simulinkblockpfad für den Eventlistener
blk = 'Arduino12bit_Tabledance/Subsystem Werte/Messdaten';
%Aufruf der Eventlistenerfunktion, der die Signale des Simulinkblocks
%Messdaten während der Laufzeit "mithöhrt"
h = add_exec_event_listener(blk, 'PreOutputs', @plot_adapt_coefs2);
```

Wie am Code nachzuvollziehen ist, wird zunächst ein neues Fenster geöffnet, damit ein eventuell vorhandenes Fenster nicht überschrieben wird. Danach werden global gültige Vektoren im Datentyp single mit einer Länge von 150 erstellt. Diese dienen als Messwertspeicher, wobei die Vektorlänge letztendlich die Zeit, bis der Messpunkt wieder gelöscht wird, bestimmt. Anschließend wird der Plot mithilfe der Vektoren erstellt und konfiguriert. Der erste und aktuellste Wert der Vektoren wird in diesem Plot zusätzlich noch mit einem Marker dargestellt, womit die aktuelle Position der Kugel auf dem Plot besser nachvollzogen werden kann. Als nächstes wird im Code das Handle „plot1“ vom soeben erstellten Plot mit der Funktion „set_param“ als „UserData“ dem Block Messdaten angehängt. Dieses Handle wird später in der Funktion „plot_adapt_coefs2“ benötigt, um auf den

erstellten Plot zugreifen und dynamisch auf die neuen Werte abändern zu können. Ohne die Übertragung des Handle mit set_param ist dieser in der externen Funktion nicht verfügbar.
Am Schluss der OpenFcn wird der Event Listener aktiviert. Dieser erwartet drei Eingangsparameter. Davon ist der erste Parameter die Pfadangabe des Blocks, bei dem die Werte abgegriffen werden sollen. Der zweite Parameter bestimmt den Modus des Event Listeners. Das verwendete „PreOutputs“ bedeutet, dass die Werte vor der Modifikation mit der Ausgabefunktion des Blocks abgegriffen werden. Als letzten Parameter erwartet der Event Listener eine Funktion mit zwei Übergabeparametern. Der erste Parameter ist hier das Laufzeitobjekt vom Block Messdaten. Der zweite Übergabeparameter ist eine Instanz der Klasse „EventData“, mit dem Daten des Events ausgelesen werden können. Die hier verwendete Funktion plot_adapt_coefs2 wird vom Event Listener jedes Mal aufgerufen, wenn ein neuer Wert vom „Messdaten“-Block verfügbar ist, während der Matlabcode in der OpenFcn nur einmal beim Doppelklick ausgeführt wird. Dem Funktionsnamen musste zudem ein @-Zeichen vorangestellt werden.
Mit der wiederkehrend aufgerufenen Funktion plot_adapt_coefs2 werden die Messwerte erfasst und der bestehende Plot mit den neuen Werten aktualisiert. Dazu ist nachfolgen der kommentierte Matlabcode der Funktion dargestellt:

```
%Funktion, die durch den Eventlistener wiederkehrend
%aufgerufen wird
function plot_adapt_coefs2(block, event)

global X1;
global Y1;
global X2;
global Y2;
global X3;
global Y3;
%Verschieben der Messwerte um eins nach hinten, sodass der neue
%Messwert vorne eingefügt werden kann
for c = 149:-1:1
    X1(1,c+1) = X1(1,c);
    Y1(1,c+1) = Y1(1,c);
    X2(1,c+1) = X2(1,c);
    Y2(1,c+1) = Y2(1,c);
    X3(1,c+1) = X3(1,c);
    Y3(1,c+1) = Y3(1,c);
end
%Einfügen des neuen Messwerts am Anfang des Vektors
X1(1,1) = block.InputPort(1).Data(1);
Y1(1,1) = block.InputPort(1).Data(4);
X2(1,1) = block.InputPort(1).Data(2);
Y2(1,1) = block.InputPort(1).Data(5);
X3(1,1) = block.InputPort(1).Data(3);
Y3(1,1) = block.InputPort(1).Data(6);
%Holen des am Block "Messwerte" angehängten Handles vom zuvor
%erstellten Plot
plot1 = get_param(block.BlockHandle,'UserData');
%Abänderung des Plots auf die neuen Werte mithilfe des Handles
set(plot1(1),'XData',X1);
set(plot1(1),'YData',Y1);
```

```
set(plot1(2),'XData',X2);
set(plot1(2),'YData',Y2);
set(plot1(3),'XData',X3);
set(plot1(3),'YData',Y3);
set(plot1(4),'XData',X1(1,1));
set(plot1(4),'YData',Y1(1,1));
set(plot1(5),'XData',X2(1,1));
set(plot1(5),'YData',Y2(1,1));
set(plot1(6),'XData',X3(1,1));
set(plot1(6),'YData',Y3(1,1));
%Zeichnungsbefehl mit Frameratenlimitierung (ressourcenschonend)
drawnow limitrate nocallbacks;
```

Wie zu erkennen ist, werden in der Funktion die Messwerte der Messwertvektoren mit einer for-Schleife um eins nach hinten verschoben, bevor die neuen Messwerte am Anfang der Vektoren eingefügt werden. Damit bleibt die korrekte Reihenfolge der Messwerte erhalten, sodass diese auch richtig im Plot angezeigt werden können. Die aktuellen Messdaten werden dabei durch Zugriff auf das Laufzeitobjekt „block“ erfasst.
Danach wird das am Messdaten-Block angehängte Handle mit dem Befehl „get_param“ geholt. Unter Verwendung des Handle und dem „set“-Befehl wird anschließend der Plot auf die aktuellen Messwerte angepasst.

6.3.4 Auswertung

Dieser Bereich beinhaltet verschiedene Blöcke mit denen die vom „Messdaten“-Block aufgenommenen Werte visualisiert als auch für die Verwendung mit pzMove vorbereitet werden können. Weil die Messdaten erst nach dem Stoppen des Programms in den Workspace geschrieben werden, funktioniert die Auswertung erst nach dem Beenden des Regelungsprogramms. In Abbildung 42 ist der Bereich der Auswertung im Programm gezeigt, wobei die Namen der einzelnen Blöcke deren Funktion beschreiben.
Erstellt wurden diese Blöcke für die Auswertung in ähnlicher Weise wie der bereits beschriebene „X-Y Laufzeitplot“. Auch hier wurde die Callback-Funktion „OpenFcn“ verwendet, um Matlabcode in Simulink anwenden zu können. Der implementierte Code der verschiedenen Blöcke überschneidet sich oftmals stark, weshalb hier nachfolgend nur der „t-X-StellX Plot gesamt“-Block und „SprungantwortX“-Block erläutert werden.

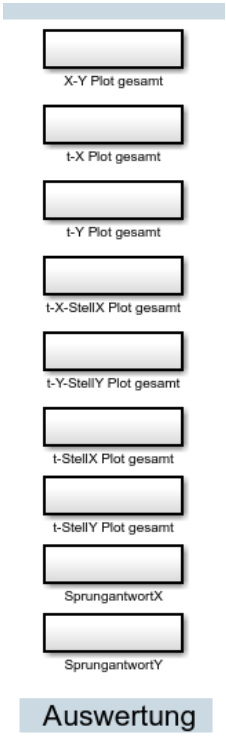


Abbildung 42: Blöcke für die Auswertung

Der „t-X-t-StellX Plot gesamt“ stellt die Kugelposition in X-Richtung und die X-Achsenstellwerte in Abhängigkeit der Zeit in zwei untereinander abgebildeten Plots dar. Des Weiteren kann man mit der Zuweisung entsprechender Zeitwerte für „tAnfang“ und „tEnde“ in der „OpenFcn“ die Zeitachse auf ein interessierendes Zeitintervall zurechtschneiden, sodass der Plot, gerade bei langen Aufnahmezeiten, übersichtlich gestaltet werden kann. Der nachfolgend dargestellte Matlabcode des Blocks kann gut mithilfe der Kommentare nachvollzogen werden, weshalb an dieser Stelle auf eine weitere Erläuterung verzichtet wird:

```
%neues Fenster
hFig = figure(5);
%Variablen für die Zeitbereichseinschränkung des Plots in [s]
tAnfang = 0;
tEnde = 1000;
%Erstellen von Vektoren mit den benötigten Messdaten
IstX1 = PositionsdatenXY.signals(1).values(:,1);
SollX1 = PositionsdatenXY.signals(1).values(:,2);
SimX1 = PositionsdatenXY.signals(1).values(:,3);
StellX1 = PositionsdatenXY.signals(1).values(:,7);
StellsimX1 = PositionsdatenXY.signals(1).values(:,8);
Time1 = PositionsdatenXY.time;
%Algorithmus für das Auffinden vom Anfang (Variable i) und
%Ende (Variable u) des relevanten Bereichs des Zeitvektors Time1
%entsprechend der definierten Variablen tAnfang und tEnde
i = 1;
while tAnfang > Time1(i,1)
    i = i + 1;
end
if i > 1
    i = i - 1;
end
L = length(Time1);
for r = 1:1:L
    if Time1(r,1) < tEnde
        u = r;
    end
end
Time1 = Time1 - Time1(i,1);
%Extrahieren des relevanten Messwertbereichs
Time2 = Time1(i:u,1);
IstX2 = IstX1(i:u,1);
SollX2 = SollX1(i:u,1);
SimX2 = SimX1(i:u,1);
StellX2 = StellX1(i:u,1);
StellsimX2 = StellsimX1(i:u,1);
%plotten der Positionswerte und Stellwerte in jeweils einem Subplot
ax1 = subplot(2, 1, 1);
ax2 = subplot(2, 1, 2);
plot(ax1, Time2, IstX2, 'b', Time2, SollX2, '-.r', Time2, SimX2, '--g');
plot(ax2, Time2, StellX2, 'c', Time2, StellsimX2, '--m');
%Beschriftung und Konfigurierung des Plots
grid(ax1, 'on');
grid(ax2, 'on');
axis (ax1, [0 inf 0 33]);
axis (ax2, [0 inf -600 600]);
legend(ax1, 'Istwert','Sollwert','Simulation');
legend(ax2, 'Istwert', 'Simulation');
title(ax1, 'Kugelkurvenverlauf X-Achse');
title(ax2, 'Stellwertverlauf X-Achse');
```

```
xlabel(ax1, 'Zeit in [s]');
xlabel(ax2, 'Zeit in [s]');
ylabel(ax1, 'Kugelposition X in [cm]');
ylabel(ax2, 'Stellwert X');
```

Der „SprungantwortX“- bzw. der „SprungantwortY“-Block ist dagegen nicht primär für die Visualisierung sondern für das Aufbereiten der aufgenommenen Werte einer Sprungantwort gedacht. Per Doppelklick wird im Workspace von Matlab automatisch die Variable „Sprungantwort“ erstellt, die ohne weitere Modifikation für die Analyse mit pzMove verwendet werden kann. Die aufgenommen Messwerte weisen trotz fester Samplerate des Regelungsprogramms keine gleichen Zeitdeltas zwischen den Werten auf, was jedoch zum Verarbeiten eine Voraussetzung für das Lernprogramm pzMove ist. Das Ausführen des Blocks rechnet die Werte auf einen normalisierten Zeitvektor um und schreibt diese in einer zu pzMove kompatiblen Form in eine Variable. Außerdem wird durch Vergleichen der Stellwerte der Beginn der Sprungantwort ermittelt und die Werte darauf normiert, sodass der Sprung nach der Bearbeitung der Werte genau bei null Sekunden mit der Position null beginnt. Diese automatische Anpassung funktioniert jedoch nur, wenn der Stellwert vom Sprungzeitpunkt bis zum Ende der Messung gleich ist.

Der Block „SprungantwortX“ beinhaltet folgenden Matlabcode, wobei für die Erläuterung des Codes auch hier auf die Kommentare verwiesen sei:

```
%neues Fenster
hFig = figure(9);
%Erstellen von Vektoren mit den benötigten Messdaten
IstX1 = PositionsdatenXY.signals(1).values(:,1);
StellX1 = PositionsdatenXY.signals(1).values(:,7);
Time1 = PositionsdatenXY.time;
%Auffinden vom Anfang der Sprungantwort durch Vergleich
%der Stellwerte vom Ende aus
Zende = length(StellX1);
i = 1;
while StellX1(Zende - i,1) == StellX1(Zende,1)
    i = i + 1;
end
Zanfang = Zende - i + 1;
if i < 10
   Zanfang = 1;
end
%Extrahieren der Sprungantwort und Nullabgleich
Time2 = Time1(Zanfang:Zende,1);
IstX2 = IstX1(Zanfang:Zende,1);
StellX2 = StellX1(Zanfang:Zende,1);
Time2 = Time2 - Time2(1,1);
IstX2 = IstX2 - IstX2(1,1);
IstX2 = single(IstX2);
StellX2 = single(StellX2);
%Erstellen neuer Messdatenvariablen
IstX3 = 0;
StellX3 = 0;
Time3 = 0;
Time3(1,1) = 0;
%Erstellen eines normierten Zeitvektors Time3 mit konstanten Zeitdeltas
```

```
%und entsprechender Länge
k = 1;
L1 = length(Time2);
while Time2(L1,1) > Time3(k,1) + 0.025
    k = k + 1;
    Time3(k,1) = Time3(k-1,1) + 0.025;
end
%Generieren der zu Time3 passenden Messwerte durch
%lineare Approximation
L2 = length(Time3);
IstX3(1,1) = IstX2(1,1);
StellX3(1,1) = StellX2(1,1);
for n = 2:1:L2
    k = 2;
    while Time3(n,1) > Time2(k,1)
        k = k + 1;
    end
    %lineare Approximation
    Steigung1 = (IstX2(k,1) - IstX2(k-1,1))/(Time2(k,1) - Time2(k-1,1));
    Achsenabschnitt1 = IstX2(k,1) - (Steigung1 * Time2(k,1));
    IstX3(n,1) = Steigung1 * Time3(n,1) + Achsenabschnitt1;
    Steigung2 = (StellX2(k,1) - StellX2(k-1,1))/(Time2(k,1) - Time2(k-
    1,1));
    Achsenabschnitt2 = StellX2(k,1) - (Steigung2 * Time2(k,1));
    StellX3(n,1) = Steigung2 * Time3(n,1) + Achsenabschnitt2;
end
%Erstellen einer passenden Variable mit Werten für die
%Verwendung mit pzMove
Sprungantwort(:,1) = Time3(:,1);
Sprungantwort(:,2) = StellX3(:,1);
Sprungantwort(:,3) = IstX3(:,1);
%Plotten der Ergebnisse
plot1 = plot(Time3, IstX3, 'b');
%Beschriftung und Konfigurierung des Plots
grid on;
axis ([0 inf 0 33]);
legend('Sprungantwort');
title('Sprungantwort X-Achse');
xlabel('Zeit in [s]');
ylabel('Kugelposition X in [cm]');
```

6.3.5 Weitere Bedienfunktionen

Das Programm enthält noch drei weitere Bedienfunktionen, die nachfolgend erläutert werden. Der entsprechende Abschnitt dieser Bedienmöglichkeiten im Programm ist in Abbildung 43 gezeigt, der sich in Stellwertstörung, Sprunggenerierung und Offsetkalibrierung aufteilt.

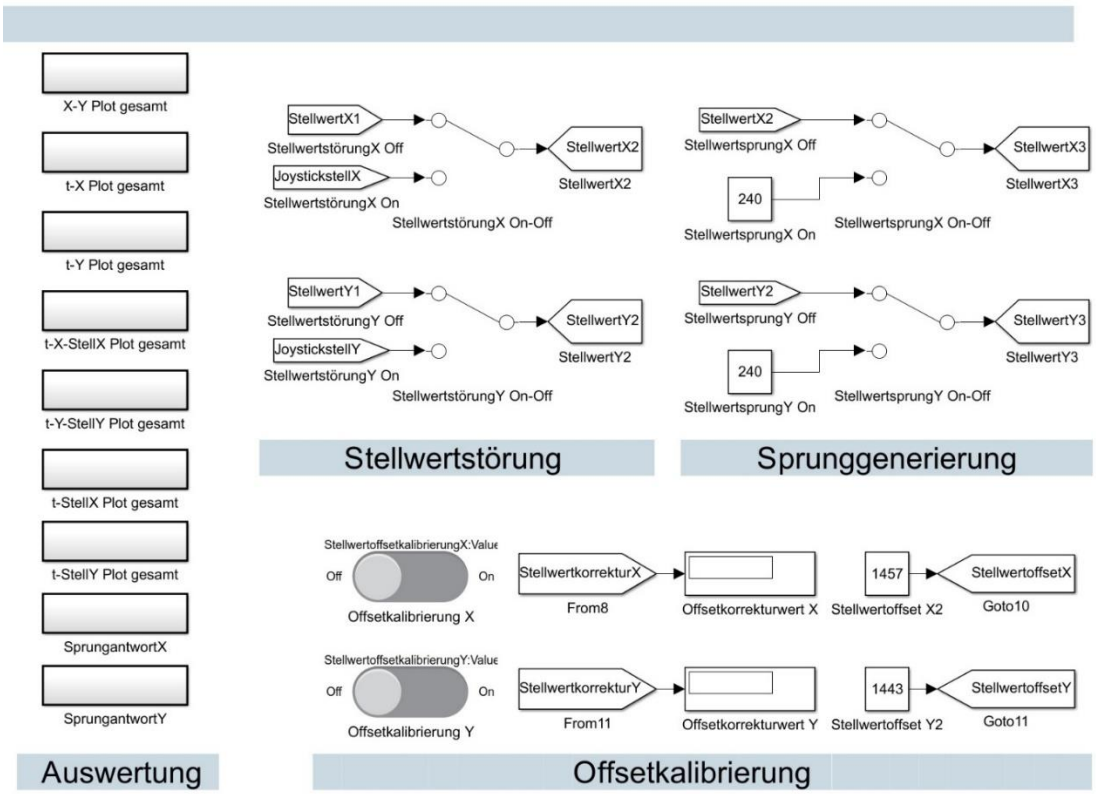


Abbildung 43: Rechte Seite der Bedienungsoberfläche

Die Funktion der Stellwertstörung ist, wie der Name schon aussagt, für das Testen des Regelungsverhaltens auf eine Stellwertstörung konzipiert, wobei die X-Achse und die Y-Achse getrennt einstellbar sind. Zum Aktivieren dieser Funktion muss lediglich der Signalfluss per Schalter auf das untere Signal umgeschaltet werden. Danach kann durch Benutzen des Joysticks ein flexibler Stellwertoffset in das ursprüngliche Stellwertsignal eingebaut werden.

Die Modifizierung des Stellwertsignals ist im Subsystem „Rechteck_Joystick“ untergebracht, welches bereits erläutert wurde. Durch die Unterbringung in diesem Subsystem konnte die Bedienoberfläche übersichtlicher gehalten werden und hat zudem den Vorteil, dass das aufbereitete Messsignal vom Joystick ohne großen Umstände auch für die Stellwertstörung mit verwendet werden konnte.

Eine weitere Funktion stellt der Abschnitt „Sprunggenerierung“ dar. Diese kann verwendet werden, um vom System Sprungantworten zu generieren, die anschließend ausgewertet werden können. Das Bedienen funktioniert in gleicher Weise wie bei der Stellwertstörung durch das Umschalten des Signalflusses per Schalter. Damit wird das Stellwertsignal auf eine einstellbare Kontante, die die Sprunghöhe darstellt, geschaltet, anstatt dass der vom Regler berechnete Stellwert für die Ausgabe an die Stellmotoren verwendet wird. Am besten ist es, wenn man zunächst mit dem Sollwert „Joystick“ die Kugel an einer gewünschten Position auf dem Touchscreen positioniert und anschließend per Doppelklick den definierten Stellwertsprung auslöst. Nach dem Stoppen des

Regelungsprogramms können die Messwerte direkt mit den Blöcken im Abschnitt „Auswertung“ visualisiert und für die Analyse mit pzMove vorbereitet werden.
Die letzte Bedienfunktion stellt der Programmabschnitt „Offsetkalibrierung“ dar, der für das Kalibrieren des Tabledance gedacht ist. Die Funktion ist bereits mit der S-Funktion „S-Funktion Servoansteuerung“ implementiert worden, die bereits in Kapitel 6.2.3 beschrieben wurde. Im Bereich der Offsetkalibrierung ist daher nur noch das nötigste zum Anwenden der Funktionalität untergebracht. Mit den Schiebeschaltern „Offsetkalibrierung X“ und „Offsetkalibrierung Y“ kann durch schieben auf „On“ die Suche des Stellwertkorrekturwerts bei laufendem Programm jeweils für die X- und Y-Achse getrennt gestartet werden. Bleibt der rechts daneben angezeigte Korrekturwert stabil, muss zum Kalibrieren lediglich der wiederrum rechts davon positionierte Stellwertoffset um den Korrekturwert geändert werden. Danach sollte die Funktion der Offsetkalibrierung wieder deaktiviert werden.

7. Analyse des Systems

7.1 Verifizierung des Simulationsmodells mit pzMove

Mit der Funktionalität der Sprungantwortgenerierung des Tabledance bietet es sich an, das theoretisch erstellte Simulationsmodell mit einem über die Sprungantwort approximiertem Modell zu vergleichen. Dafür kann das hochschulinterne Regelungstechniklernprogramm pzMove verwendet werden.

Zunächst musste die Sprungantwort für eine Achse generiert werden, wobei hier die X-Achse gewählt wurde. Dazu wurde mit dem Joystick die Kugel in der Mitte zwischen linkem Rand und der Mitte des Touchscreens platziert. Diese Stelle ist am repräsentativsten für das System. Je nachdem von wo aus die Sprungantworten gestartet werden, unterscheiden sie sich leicht, denn der Stellwertsprung lässt die Kugel durch die Zentrifugalkraft in Abhängigkeit zum Abstand zur Mitte nach außen bewegen. Mit Doppelklick auf dem Schalter für den Stellwertsprung wurde der Stellwertsprung in Höhe von 240 ausgeführt und aufgenommen. Danach konnte diese mit dem bereits beschriebenen Block „SprungantwortX“ automatisch für pzMove vorbereitet werden. Die erstellte Variable „SprungantwortX“ muss zum Verwenden mit pzMove jedoch noch abgespeichert und in den „temp“-Ordner von pzMove kopiert werden.

Nach Einstellung der Parameter in pzMove konnte die Modellgleichung durch Betätigen des „Update“-Buttons gebildet werden. Das leicht nachmodifizierte Ergebnis ist unterhalb in Gleichung (5) dargestellt, wohingegen das theoretisch berechnete Modell zu Vergleichszwecken darunter in Gleichung (6) dargestellt ist.

$$Gs(s) = \mathbf{0,295}\ \frac{\boldsymbol{cm}}{\boldsymbol{sec^2}} * \frac{\mathbf{1}}{\boldsymbol{s^2}} \quad (5)$$

$$Gs(s) = \mathbf{0,328}\ \frac{\boldsymbol{cm}}{\boldsymbol{sec^2}} * \frac{\mathbf{1}}{\boldsymbol{s^2}} \quad (6)$$

Die beiden Modelle sind bis auf den Wert der Konstanten, die sich leicht unterscheiden, vollkommen übereinstimmend. Dabei macht es sogar Sinn, dass das theoretische Modell, bei dem z.B. die Reibung und die Zentrifugalkraft vernachlässigt wurde, eine etwas größere Konstante aufweist, was bedeutet, dass die Kugel schneller beschleunigt. Das mit pzMove approximierte Modell hingegen stellt das reale Verhalten besser dar. Das kann daraus geschlossen werden, dass das Modell sehr gut mit der Sprungantwort, die in Abbildung 44 gezeigt sind, übereinstimmt. Die dazugehörige Abweichung des Modells zur Sprungantwort ist darunter in Abbildung 45 dargestellt. Erkennbar ist, dass die Abweichung ziemlich gering ist und keine Vorzugsrichtung aufweist. Dass das mit pzMove entwickelte Modell genauer ist, konnte auch bei laufender Regelung beobachtet und bestätigt werden. Hier sei noch zu erwähnen, dass die gegenseitigen Auswirkungen der X- zur Y-Achse und

dynamische Effekte nicht durch die Sprungantwort berücksichtigt werden können und daher auch nicht vom Modell abgedeckt sind.

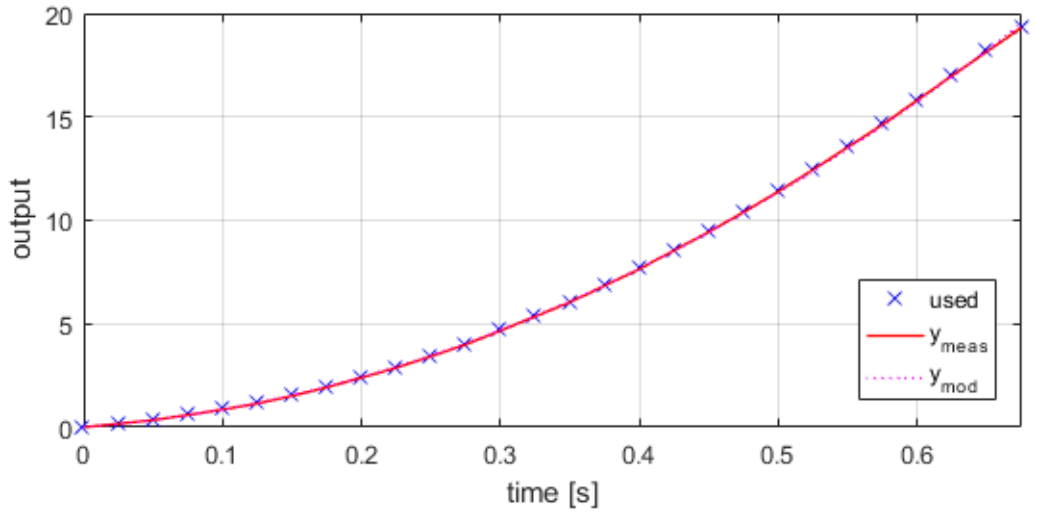


Abbildung 44: Gemessene Sprungantwort und approximiertes Modell zur Sprungantwort

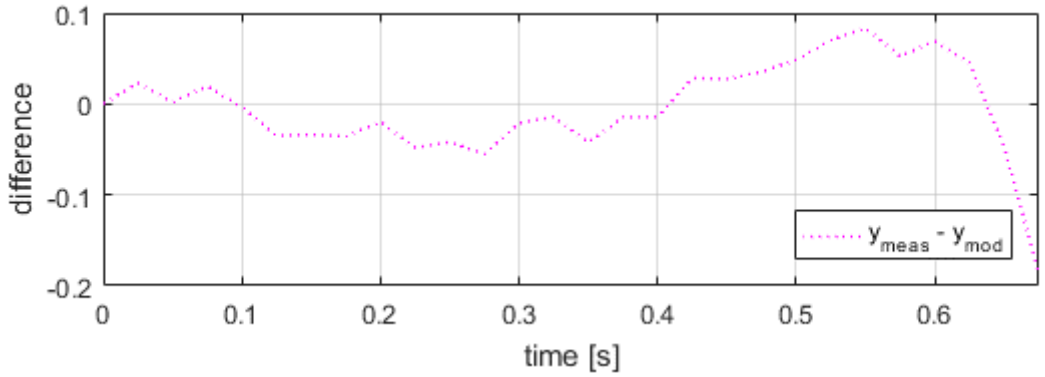


Abbildung 45: Abweichung von gemessener Sprungantwort zur approximierten Sprungantwort

7.2 Regelungsalgorithmus

7.2.1 PID-Regler

Als Regelungsalgorithmus ist im System ein PID-Regler mit einem Filter für den D-Anteil implementiert. Bereits ein PD-Regler ist für eine stabile Regelung ausreichend, denn die Regelungsstrecke hat selbst ein integrales Verhalten, weswegen ein I-Anteil im Regler überflüssig ist. Der P-Regler stellt dabei die Neigung des Touchscreens in Richtung des Sollwerts linear zum Abstand der Kugel zum Sollwert ein. Im theoretischen Fall entsteht mit einem alleinigen P-Regler eine sinusförmige Dauerschwingung mit einer Amplitude in der Höhe vom Anfangswert. Am

Vorgängermodell vom Tabledance ist diese Schwingung sehr stark aufklingend. Vor allem die in der Modellgleichung nicht berücksichtigte Zentrifugalkraft und Latenzen bei der Stellwertausführung tragen zum Aufschwingen bei. Die Abbildung 46 zeigt die Messwerte vom Versuch mit einem P-Regler am neuen System. Hierbei ist auch ein instabiles und aufschwingendes Verhalten zu beobachten, das aber deutlich schwächer ist als am Vorgängersystem.

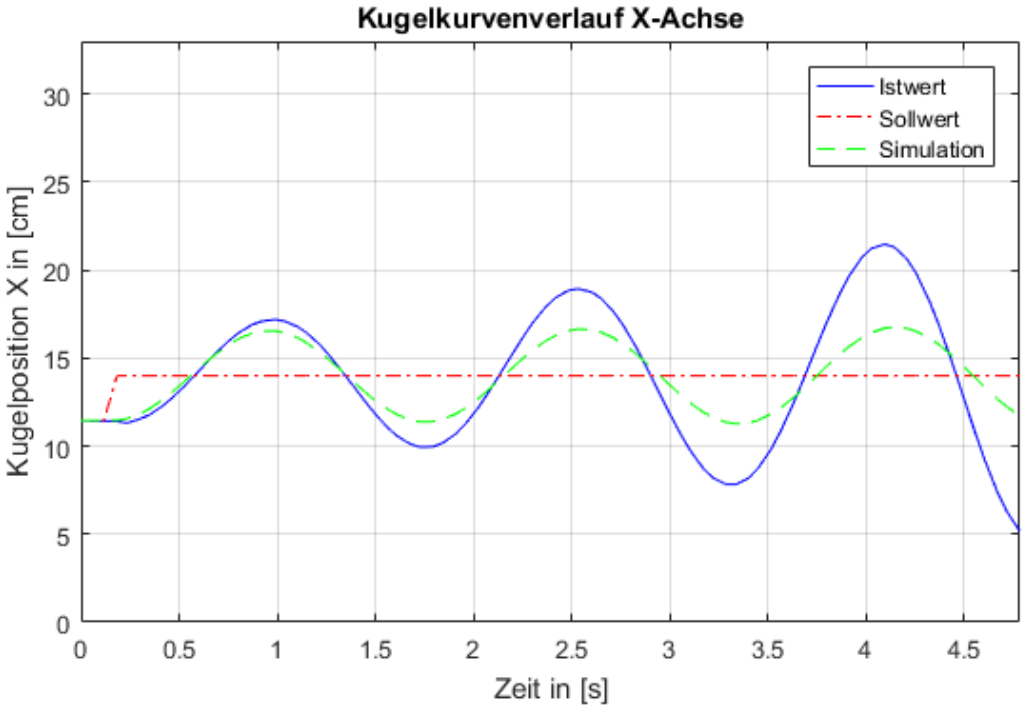


Abbildung 46: Kugelkurvenverlauf bei der Regelung mit einem P-Regler mit P = 50

Neben dem P-Anteil im Regler ist der D-Anteil für eine stabile Regelung der Kugel nötig. Dieser sorgt für die Dämpfung im System, indem er dem vom P-Anteil hervorgerufenen Stellwert bei der Bewegung der Kugel zum Sollwert entgegenwirkt bzw. den Stellwert verstärkt, sobald die Kugel sich vom Sollwert entfernt. Praktisch gesehen reagiert der D-Anteil im Regler auf die Geschwindigkeit der Kugel, was entsprechend auch die Ableitung der Kugelposition ist. Da die Messwerte bei hohen Sampleraten nicht weit auseinander liegen, ist es klar, dass Messrauschen den D-Anteil im Regler stark beeinflussen kann. Bei der erreichten 11,5 bit effektiven Messgenauigkeit bei 333 Hz ist ein Filterkoeffizient von um die 20 die richtige Wahl, um das Stellwertrauschen vom D-Anteil gering zu halten und dennoch keine zu starke Latenz zu erzeugen.
Der I-Anteil im Regler sollte für normale Regelungsaufgaben null sein, denn dieser verstärkt lediglich das integrale Verhalten der Strecke. In Ausnahmefällen kann dieser dennoch nützlich sein, insbesondere wenn eine bleibende Regelungsabweichung bei einer Eingangsgrößenstörung beseitigt werden soll. Dies wird im Kapitel 7.4 nochmals aufgegriffen.

7.2.2 Reglerparameterwerte

Nachfolgend sollen sinnvolle Regelungsparameterwerte für das Tabledance hergeleitet werden. Besonders wichtig ist dabei, das System und seine Grenzen, neben dem Simulationsmodell, mit einzubeziehen. Lässt man diese unberücksichtigt, wäre laut dem Modell ein PD-Regler umso besser je höher die Werte sind.
Die sinnvolle Höhe der Regelparameter ist durch verschiedene Grenzen des Tabledance selbst begrenzt. Zum einen ist der maximale Neigungswinkel des Touchscreens auf ca. 15° begrenzt. Zum anderem ist auch die Größe des Touchscreens selbst begrenzt. Werden jetzt zu hohe Parameterwerte genommen, werden die Neigungswinkelgrenzen zu oft überschritten, sodass keine konsistente Regelung zustande kommt und die Kugel mit dem Zaun kollidiert. Außerdem ändern sich die Stellwerte so schnell, dass die Kugel ohne die Stellwertänderungsgeschwindigkeitsbeschränkung abheben würde. Zu hohe Regelwerte führen somit unweigerlich zu einem instabilen Regelungsverhalten.
Sinnvolle Werte für den P-Regler können dennoch abgeschätzt werden, indem die Stellwertgrenze mit der Kugelpositionsgrenze verglichen wird. Ziel ist es, moderate Stellwerte zu generieren, ohne dass die Stellwertgrenzen zu schnell überschritten werden. Das ist am Tabledance z.B. der Fall, wenn die maximale Stellwertgrenze erst bei einer Regelabweichung von der Hälfte der Länge des Touchscreens erreicht wird. Die Gleichung (7) zeigt die Berechnung des P-Anteils anhand des aufgestellten Zusammenhangs.

$$Px_{schätz} = \frac{Stellbereich}{0{,}5 * Länge\ Touchscreen} = \frac{500}{0{,}5 * 33\ cm} = 30$$

Ausgehend vom abgeschätzten Wert des P-Reglers kann der dazugehörige D-Reglerparameterwert unter der Bedingung, dass ein bestimmtes Einschwingverhalten erreicht werden soll, z.B. mit der Methode der Polvorgabe berechnet werden. Da hier aber schon ein Simulationsmodell in Simulink erstellt wurde, konnte der entsprechende Wert des D-Anteils im Regler auch durch Ausprobieren an der Simulation gefunden werden. Ein anzustrebendes Regelungsverhalten ist in diesem Fall ein annähernd aperiodisches Einschwingverhalten mit beispielsweise 10 % Überschwingweite, was bedeutet, dass die Dämpfung des Systems etwas kleiner wie eins ist. Ein demensprechendes Regelungsverhalten wurde bei der Simulation mit einem D-Reglerwert von 12 erzielt. Das konnte auch am Gerät selbst bestätigt werden.
Tests haben gezeigt, dass das Tabledance auch mit höheren Regelungsparametern auskommt. Als gut haben sich ein P-Wert von 50 und ein D-Wert von 14 erwiesen. Setz man Verstärkungsfaktoren deutlich über 50 ein, kann das System bei bestimmten Sollwerten übermäßig zum Schwingen anfangen, weil die Stellwertgrenzen und die Stellwertänderungsgeschwindigkeit nicht mehr

ausreichend sind. Außerdem treten bestimmte Effekte verstärkt auf, die das Regelungsverhalten negativ beeinflussen, wodurch auch die Simulation ungenauer wird. Den größten Effekt ist der Zentrifugalkraft zuzuschreiben, die bei hohen Regelungsparametern verstärkte Auswirkungen hat. Diese wirkt bei schnellen Neigungswinkeländerungen eine signifikante Kraft gegen die Beschleunigungskraft aus, weshalb die Kugel zunächst entgegen dem Sollwert ausgelenkt oder langsamer beschleunigt als es vom Simulationsmodell suggeriert wird. Zur Folge ist der Neigungswinkel des Touchscreens größer als bei der Simulation, woraufhin die Kugel schneller beschleunigt und letztendlich eine höhere Geschwindigkeit erreicht, als es die Simulation zeigt. Dadurch kommt es zu einem deutlich geringer gedämpften Einschwingverhalten bis hin zur Instabilität. Die wird vor allem dadurch bedingt, dass durch den nochmals erhöhten Stellwinkeln die Stellwertgrenzen schneller nicht mehr ausreichend sind.
Die Abbildung 47 zeigt einen Plot der Positions- und Stellwerte, bei dem das erläuterte Verhalten nachvollzogen werden kann. Zum Einsatz kam dabei ein PD-Regler mit P = 80 und D = 18.

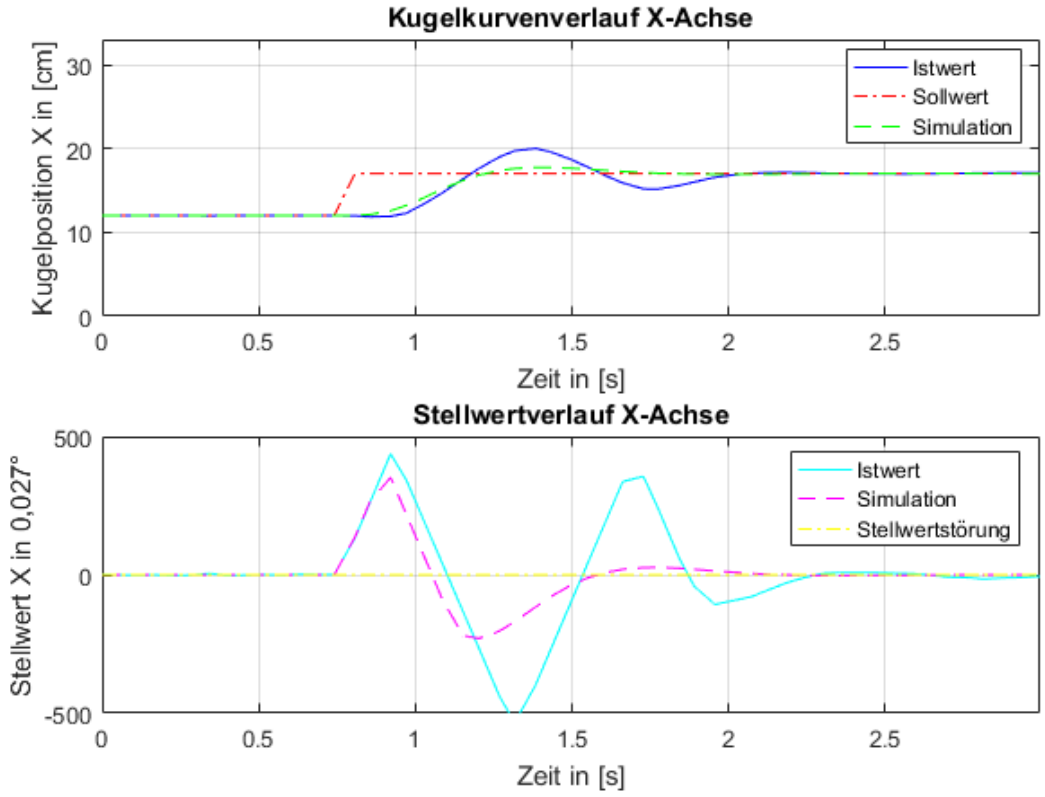


Abbildung 47: Kugelkurven- und Stellwertverlauf der X-Achse bei einer Regelung mit P = 80 und D = 18

Erkennbar ist, dass die Kugel wie beschrieben zunächst langsamer beschleunigt und anschließend deutlich stärker beschleunigt. Der reale Kugelverlauf erreicht sogar schneller als die Simulation den

Sollwert, schwing aber, aufgrund der höheren Geschwindigkeit, deutlich über. Das Simulationsmodell hingegen zeigt ein nur leicht überschwingendes Verhalten.
Anhand der Stellwerte ist leicht zu erkennen, dass der simulierte Stellwertverlauf deutlich kleiner ist, wodurch bestätigt wird, dass die Kugel stärker beschleunigt. Bezeichnend ist vor allem die erste Stellwertreaktion auf den Sollwertsprung, bei dem der reale Stellwert aufgrund der verzögernden Wirkung der Zentrifugalkraft einen deutlich höheren Wert als die Simulation erreicht. Sowohl der Stellwertverlauf der Simulation als auch der reale Stellwertverlauf zeigen auf, dass die Regelung bereits mit der maximalen Stellwertänderungsgeschwindigkeit arbeitet, was an den linearen Stellwertverläufen erkennbar ist.
Insgesamt wird die Regelung mit hohen Parametern zwar schnell, aber mit deutlich schlechterem Einschwingverhalten bis hin zur Instabilität. Zusätzlich wird das Simulationsmodell bei hohen Regelungsparametern deutlich schlechter, was zum Großteil der Zentrifugalkraft zuzuschreiben ist.

7.3 Regelungsverhalten

7.3.1 Sollwert Joystick

Mit dem angebauten Fingerjoystick kann der Sollwert proportional zur Joystickauslenkung an der X- und Y-Achse verändert werden. Damit sind beliebige Bewegungen bis zu einer bestimmten Geschwindigkeit manuell vorgebbar. Die Abbildung 48 zeigt einen aufgenommenen Kugelkurvenverlauf der X-Achse mitsamt den dazugehörigen Stellwerten, bei dem der Sollwert durch den Joystick vorgegeben wurde. Zu erkennen ist, dass hier der Istwert und der simulierte Kurvenverlauf recht gut übereinstimmen. Am geringsten ist die Regelungsabweichung bei Sollwertrampen mit konstanter Geschwindigkeit. Wie man an den Graphen erkennen kann, wird die Kugel zunächst auf die entsprechende Geschwindigkeit beschleunigt und die Regelungsabweichung anschließend im Idealfall zu Null. Auch die dazugehörigen Stellwerte stimmen relativ gut mit den simulierten Stellwerten überein. Auffällig ist aber, dass bei Stellwertsprüngen der reale Stellwert immer etwas größer wie der simulierte Stellwert ist. Vermutlich ist das der schon erwähnten Zentrifugalkraft zuzuschreiben, denn diese wirkt besonders bei Stellwertsprüngen und verstärkt die Stellwertreaktion ausgehend vom Differentialanteil des Reglers.

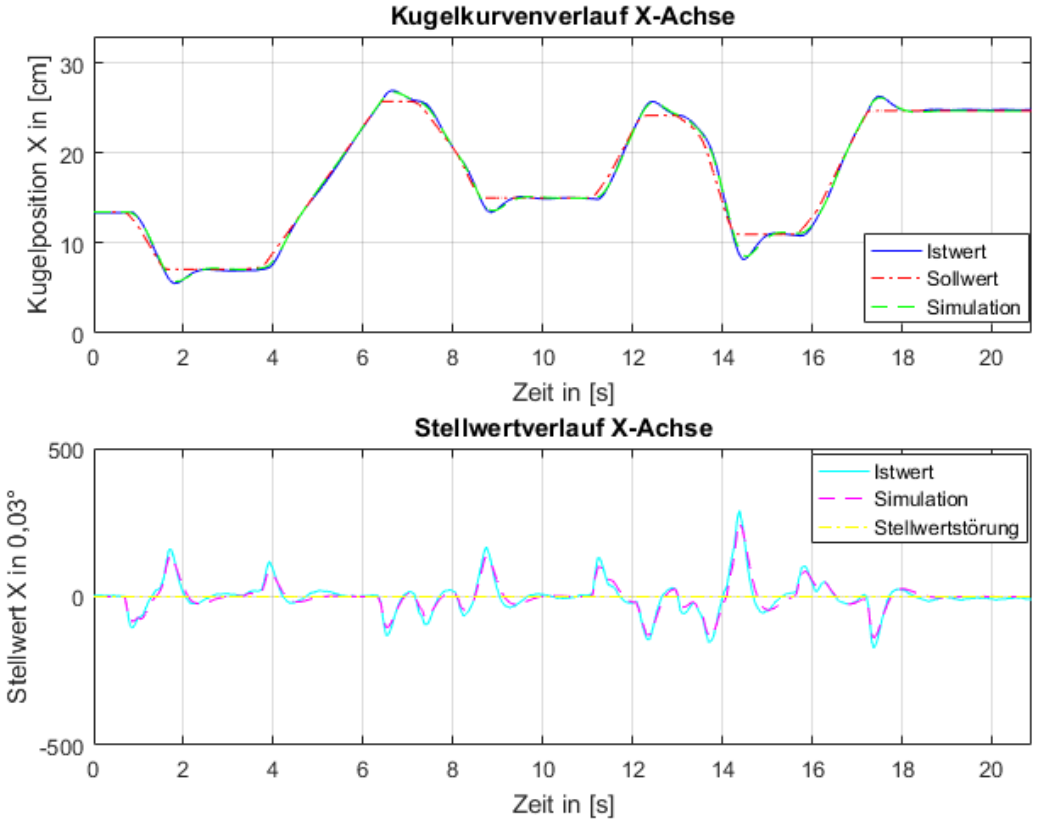


Abbildung 48: Kugelkurven- und Stellwertverlauf der X-Achse mit dem Sollwert „Joystick“ und P = 50, D = 14

Die unterhalb angebrachte Abbildung 49 zeigt eine andere durch den Fingerjoystick vorgegebene Kugellaufbahn in einem X-Y-Plot.

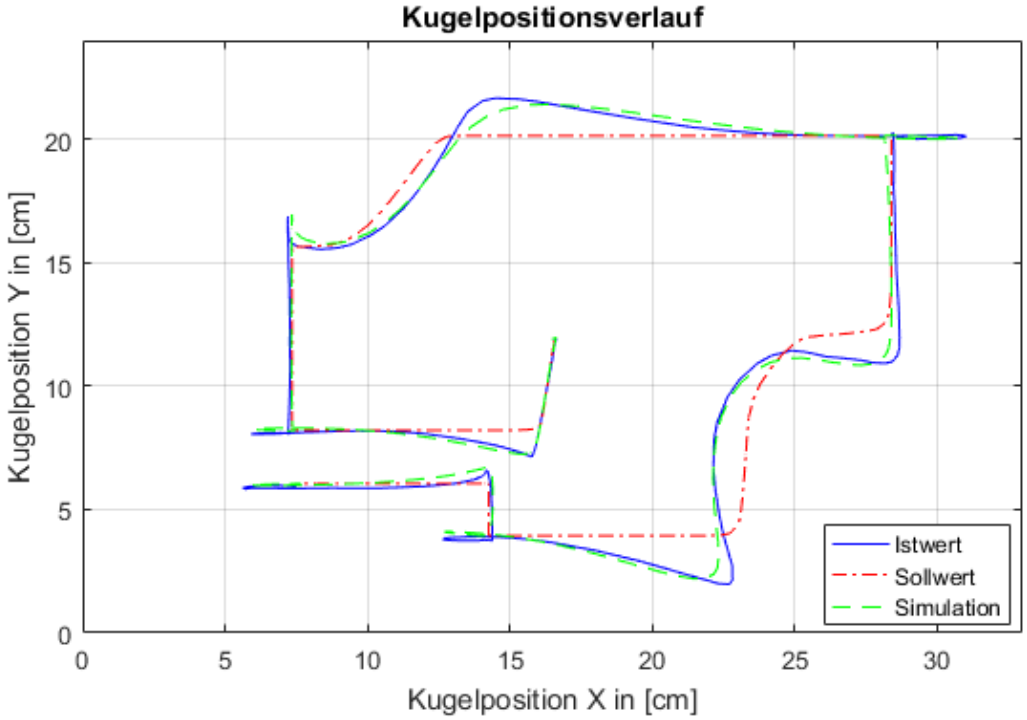


Abbildung 49: Kugelkurvenverlauf mit dem Sollwert „Joystick“ und P = 50, D = 14

Auch hier ist die Simulation bis auf einzelne Stellen recht gut mit den Istwerten übereinstimmend, obwohl es sich hier um eine zweidimensionale Bewegung handelt, deren Wechselwirkungen nicht im Modell berücksichtigt sind. Die stärksten Abweichungen der Simulation treten bei scharfen Kurven auf, weil gerade dort die Wechselwirkungen am stärksten sind.
Insgesamt ist die Regelung bei den hier verwendeten Regelungsparametern mit P = 50 und D = 14 und einer Sollwertvorgabe mit dem Joystick als vorzüglich zu bewerten, da das Regelungssystem recht nahe am ideal anzunehmenden Simulationsmodell arbeitet, obwohl nicht alle Faktoren in diesem Modell berücksichtigt sind. Das der Sollwert zum Istwert teilweise deutlich abweicht, ist der anspruchsvollen Regelungsstrecke geschuldet. Diese weist ein instabil beschleunigendes Verhalten auf, das sich verstärkend auf die Regelungsabweichung auswirkt. Noch dazu sind die Eingriffsmöglichkeiten der Regelung begrenzt, denn einerseits kann die Kugel theoretisch maximal mit der Erdbeschleunigung beschleunigen und andererseits kann das Stellsystem nicht beliebig schnell reagieren, weil ansonsten die Kugel von der Platte abheben würde.

7.3.2 Sollwert Kreis

Für eine Kreisbewegung muss die X- als auch die Y-Achse auf den Sinussollwert geschalten werden. Diese sind bereits mit einer entsprechenden Phasenverschiebung konfiguriert, sodass im zweidimensionalen ein Kreis entsteht. Während des Betriebs kann die Amplitude, die Frequenz als auch die Phasenverschiebung der Sinuskurven nach Belieben geändert werden.
Die Abbildung 50 zeigt den Positions- und Stellwertverlauf der X-Achse bei einer Kreisbewegung mit einem Radius von 4 cm und einer Frequenz von 3 rad/sec. Als Regelungsparameter wurden auch hier die etablierten Parameter mit P = 50 und D = 14 jeweils für die X- und Y-Achse verwendet.
Der Istwert vom Positionsverlauf der X-Achse ist wie der Sollwert auch sinusförmig, wobei eine Amplitudenüberhöhung von ca. 2 cm und eine deutliche Phasenverschiebung zum Sollwert zu erkennen ist. Somit weicht die Kreisbewegung deutlich vom Sollwert ab, verhält sich aber ziemlich genau, wie es vom Simulationsmodell simuliert wird. Lediglich der reale Stellwertverlauf hat eine erkennbare Phasenverschiebung zur Stellwertsimulation. Dies kann auf Effekte zurückgeführt werden, die bereits erläutert wurden, aber nicht im Simulationsmodell enthalten sind.

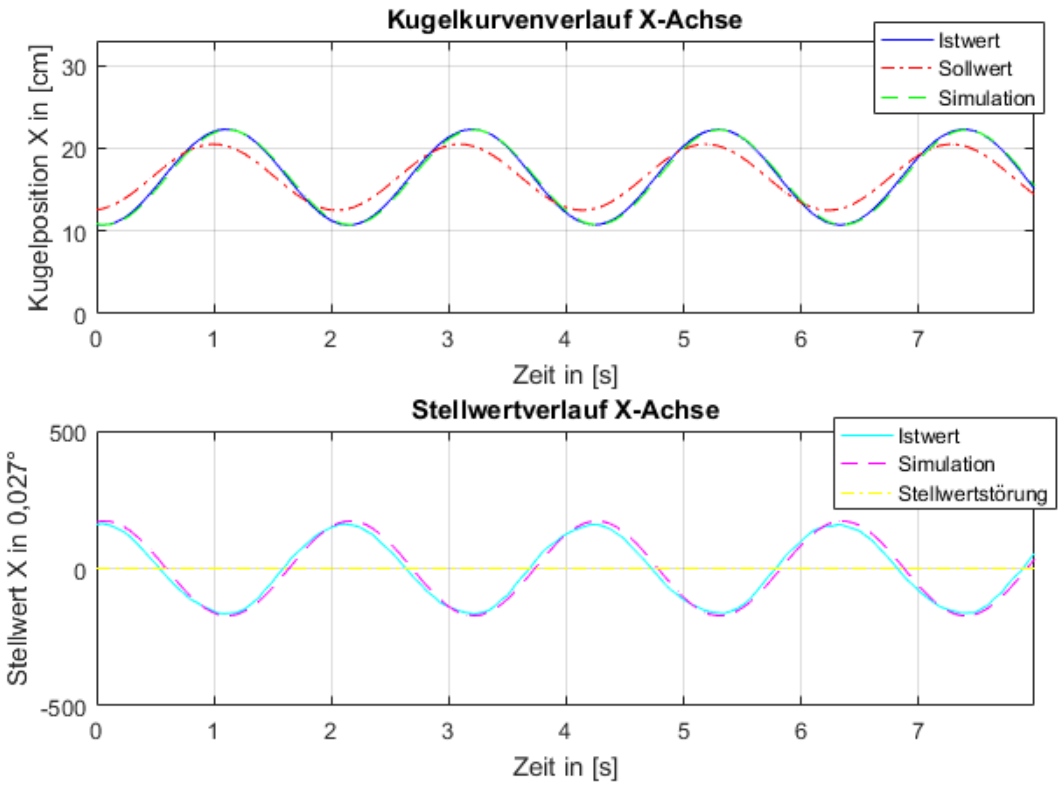


Abbildung 50: Kugelkurven- und Stellwertverlauf mit Sollwert „Kreis“ und P = 50, D = 14

Die nachfolgende Abbildung 51 zeigt ein Screenshot des selbst so genannten „X-Y-Laufzeitplots“ mit der gleichen Kreisbewegung wie davor. Mit diesem X-Y-Laufzeitplot kann die Kreisbewegung während der Laufzeit gut beobachtet und mit der Simulation verglichen werden, ohne dass dieser durch zu viele Linien unübersichtlich wird. Damit können auch unmittelbar Auswirkungen von Regelungsparameteränderungen beobachtet werden, wofür dieses Gerät unter anderem auch gedacht ist.

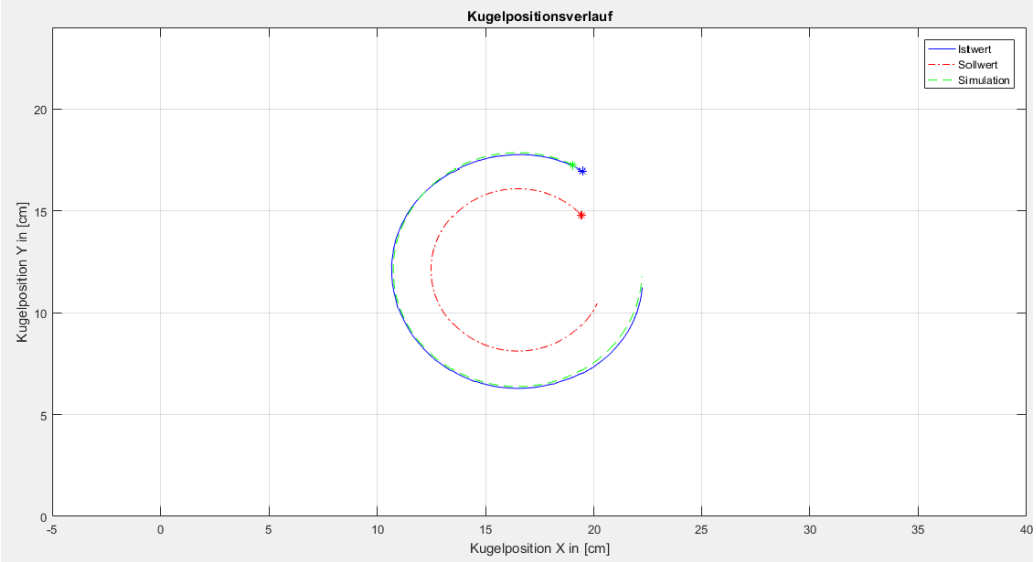


Abbildung 51: X-Y-Laufzeitplot

Sobald die Kugel sich auf der Kreisbahn befindet, stellt die Kreisbewegung einen relativ einfachen Sollwert für die Regelung dar, weil dieser gleichmäßig und ohne Sprünge ist. Dadurch können sogar höhere Regelparameter stabil betrieben werden, bei denen die Regelungsqualität bei sprungartigen Sollwerten bereits abnimmt. In diesem Zusammenhang wurde eine Messung mit stark erhöhten Parameterwerten bei gleichem Sollwert durchgeführt, dessen Ergebnis mit der Abbildung 52 gezeigt ist. Verwendet wurde hierbei ein Proportionalitätsfaktor von 100 und ein Differentialfaktor von 22. Vergleicht man die Messung mit der vorangegangenen Messung, steht fest, dass sowohl die Phasenverschiebung als auch die Amplitudenüberhöhung stark verringert sind und der Istwert somit ziemlich nahe am Sollwert liegt. Dabei konnte man beobachten, dass ein erhöhter Proportionalitätsfaktor maßgeblich die Phasenverschiebung zum sinusförmigen Sollwert verringert, während der Differentialfaktor die Phasenverschiebung wiederrum erhöht, aber die Amplitudenüberhöhung verringert.

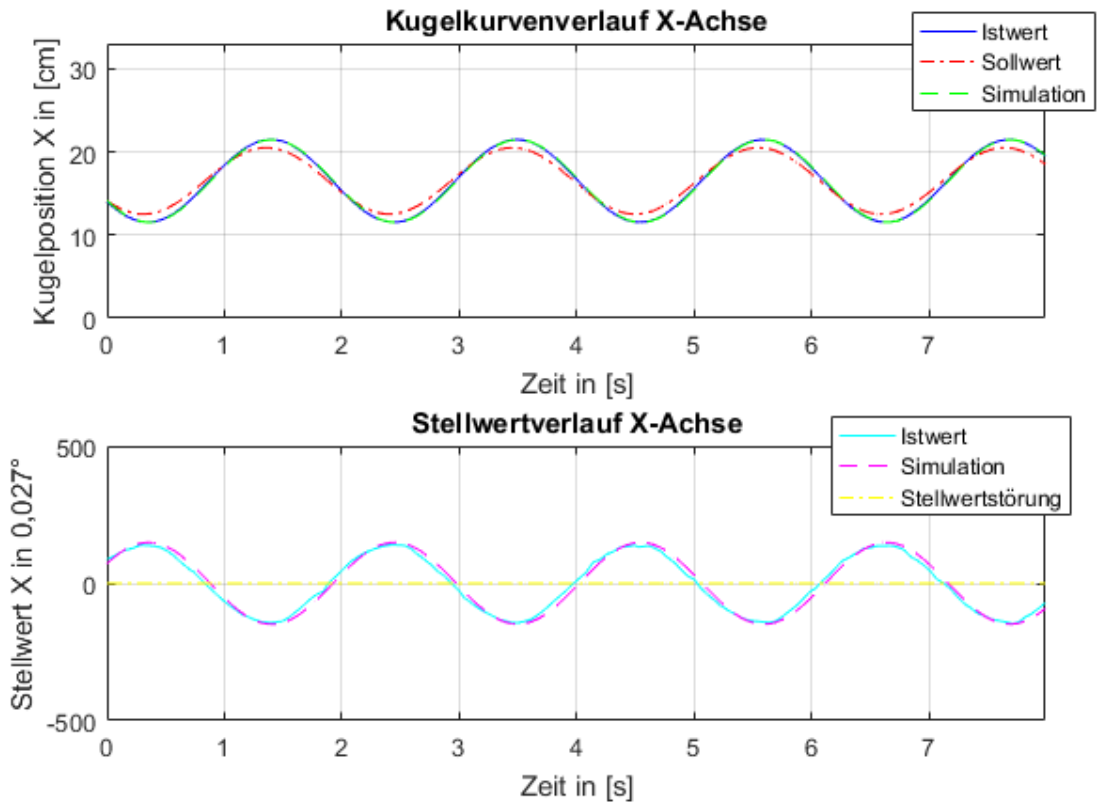


Abbildung 52: Kugelkurven- und Stellwertverlauf der X-Achse mit P = 100, D = 22

Auffallend ist, dass trotz der hohen Parameterwerte die Simulation den Istwert der Kugelposition und den des Stellwerts dennoch sehr gut wiederspiegelt. Wie bereits gezeigt, ist die Simulation bei so hohen Parameterwerten und einem sprungartigem Sollwert nicht mehr zielführend, während bei sinusförmigen Sollwerten die Simulation gut mit den Istwerten übereinstimmt. Das liegt daran, dass

die im Simulationsmodell nicht berücksichtigten Effekte bei einem sinusförmigen Sollwert nicht so stark auftreten. Bezeichnend dazu ist der Stellwert dieser Sinusbewegung zu interpretieren. Dieser ist sehr ähnlich zur vorangegangenen Messung, wonach das Stellwerk nicht stärker belastet wird als davor und dennoch eine deutlich geringere Regelungsabweichung erreicht wird.

7.3.3 Sollwert Schriftzug

Für das Tabledance wurde auch gefordert, eine beliebige Kurve über den Touchscreen aufnehmen und als Sollwertkurve vorgeben zu können. Damit lassen sich z.B. Schriftzüge realisieren, die die Kugel dann nachfährt.
Um dies zu realisieren wurden sogenannte „From Workspace“-Blöcke eingesetzt, die die Werte von Variablen im Workspace vom Matlab beziehen. Dabei muss die Variable in einer von Matlab bekannten Form gespeichert sein, sodass die Positionswerte und Zeitwerte von Matlab korrekt interpretiert werden können. Aufgenommen werden die Werte mit den „To Workspace“-Block und dem Aufnahmeformat „Structure with time“. Andere Formate wurden auch getestet, funktionieren jedoch nicht richtig. Für das Aufnehmen wurde ein gesondertes Simulinkprogramm mit dem Namen „Arduino12bit_Kurvenaufnahme“ erstellt, dessen Aufbau weitgehend gleich zum Normalprogramm ist und daher nicht weiter erläutert wird. Nach starten des Programms kann mit der Kugel oder einem stumpfen Stift eine beliebige Bahn aufgezeichnet werden. Beim Beenden des Programms werden zwei Variablen namens „KurveX12bit“ und „KurveY12bit“ im Workspace von Matlab erstellt. Diese können gespeichert werden, um sie bei Bedarf wieder in den Workspace laden zu können. Zum Verwenden eines aufgenommenen Sollwertverlaufs muss dieser zunächst im Workspace geladen werden. Im Normalzustand sind die entsprechenden „From Workspace“-Blöcke im Hauptprogramm auskommentiert, da sie beim Starten eine Fehlermeldung verursachen würden, wenn keine entsprechend benannten Variablen im Workspace vorhanden sind. Diese müssen mit Rechtsklick=>Uncomment erst wieder aktiviert werden. Außerdem müssen noch die manuellen Schalter auf das richtige Signal umgestellt werden, damit der richtige Sollwert verwendet wird. Simulink überträgt beim Starten des Simulinkprogramms alle Werte der Sollwertkurven auf den Mikrocontroller, wodurch viel Speicherplatz verbraucht wird. Das Programm ohne den externen Sollwertvariablen benötigt nur ca. 10 % des 512 kB großen Flash-Speichers. Der restliche Speicher kann für Sollwertvariablen verwendet werden. Je nachdem wie hoch die Samplerate der Sollwertkurvenaufnahme ist, kann diese um die 4 Minuten lang sein. Sind die Sollwertvariablen zu groß, überläuft der Speicher beim Starten des Programms und es gibt eine Fehlermeldung.

Unter anderem wurde eine Sollwertkurve mit dem Schriftzug „Das ist Regelungstechnik“ aufgenommen. Das Ergebnis der Regelung mit dem Sollwert und Standartregelungsparametern ist unterhalb in Abbildung 53 als X-Y-Plot dargestellt. Auch hier weicht der Istwert teils deutlich vom Sollwert ab, ist aber auch hier bis auf wenige Stellen gut mit der Simulation übereinstimmend. Will man geringere Regelungsabweichungen bekommen, können die Regelungsparameter entsprechend für diese Aufgabe optimiert werden.

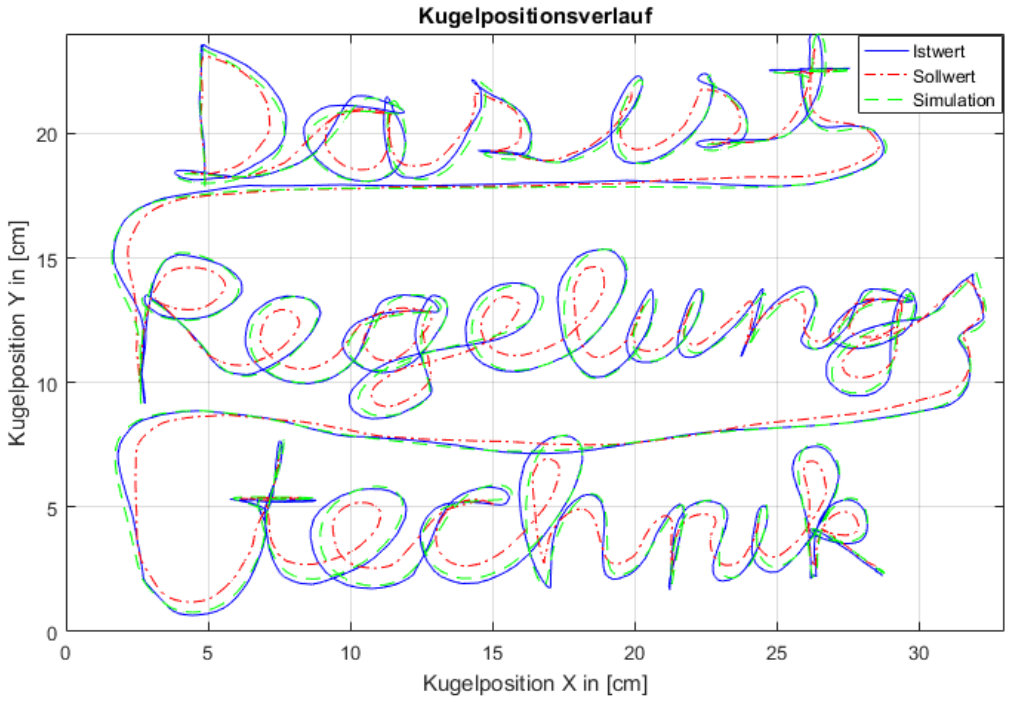


Abbildung 53: Kugelkurvenverlauf beim Schriftzug „Das ist Regelungstechnik“ mit P = 50, D = 14

7.4 Eingangsgrößenstörung

Das Tabledance ist auch geeignet um die Auswirkungen von Eingangsgrößenstörungen auf das Regelungsverhalten zu analysieren.
Als Eingangsgrößenstörung wird eine Störung am Stellwert bezeichnet. Im erstellten Regelungsprogramm ist speziell dafür die Funktion „Stellwertstörung“ integriert, mit der sich eine Stellwertstörung per Fingerjoystick aufschalten lässt. Zum Aktivieren der Funktion muss lediglich der manuelle Schalter „Stellwertstörung On Off“, den es für beide Achsen gibt, auf „On“ gesetzt werden. Mit den Fingerjoystick lässt sich danach eine Stellwertstörung proportional zur Joystickauslenkung in den Regelkreis einbauen. Der Wert der Stellwertstörung wird mit aufgezeichnet, sodass die Auswirkung dieser Störung bei der Auswertung auch quantitativ nachvollzogen werden kann. In

gewisser Weise stellt das Getriebespiel des Aufbaus auch bereits eine kleine Eingangsgrößenstörung dar, die anders als an der absichtlich aufaddierten Stellwertstörung nicht beziffert werden kann.
In der Abbildung 54 ist das Regelungsverhalten der X-Achse mit einem konstantem Sollwert und einer variablen Stellwertstörung aufgezeigt. Man erkennt, dass eine Stellwertstörung an diesem System mit implementierten PD-Regler eine dauerhafte Regelungsabweichung erzeugt, die proportional zur Stellwertstörung ist. Im statischen Fall wirkt nur der P-Anteil im Regler der Stellwertstörung aufgrund der Regelungsabweichung entgegen, sodass gilt, dass die statische Regelungsabweichung so groß wird, bis der generierte Stellwert vom P-Regler entgegengesetzt gleich groß wie die Stellwertstörung ist. Daraus folgt, dass mit höheren P-Regelwert die bleibende Regelungsabweichung sinkt. Bei dynamischer Änderung der Stellwertstörung verhält sich die Regelung so, als ob der Sollwert proportional zur Stellwertstörung verschoben wird.
Wie zu erkennen ist, liefert die Simulation auch bei einer zugeschalteten Stellwertstörung solide Werte, die besonders bei den Positionswerten gut übereinstimmen.

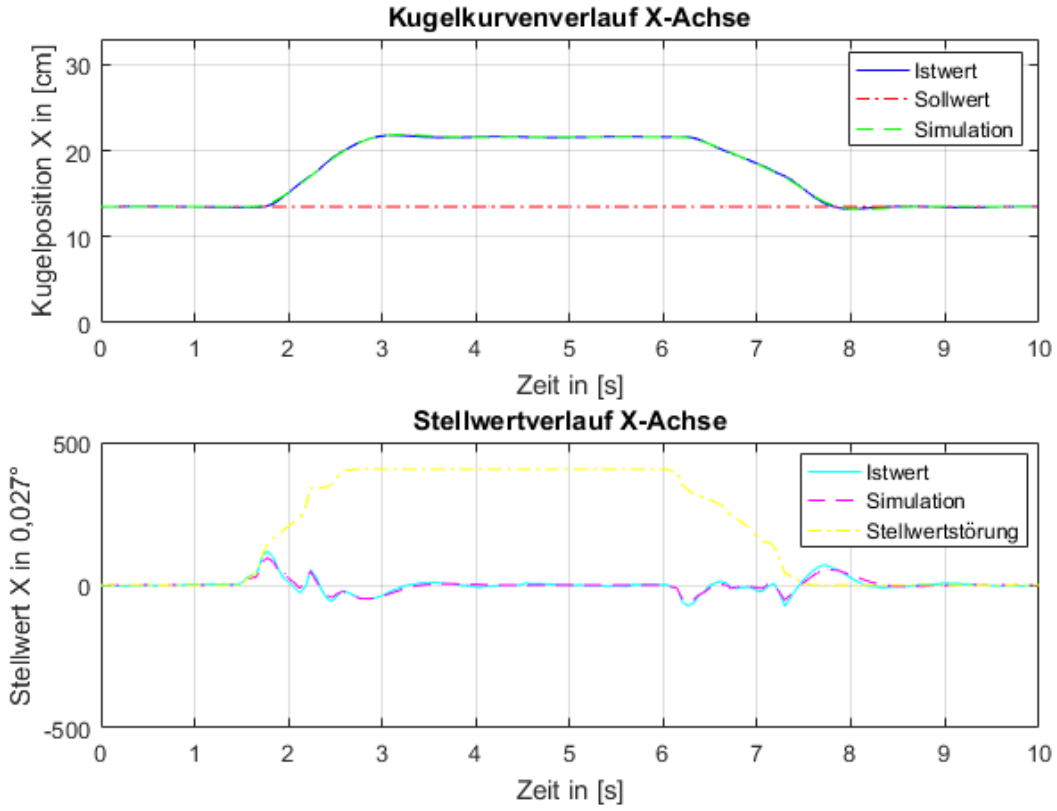


Abbildung 54: Kugelkurven- und Stellwertverlauf unter Einwirkung einer Stellwertstörung und P = 50, D = 14

Im Falle einer Stellwertstörung ist es eventuell sinnvoll einen PID-Regler zu verwenden, weil der integrale Anteil im Regler bleibende Regelungsabweichungen eleminieren kann. Wie bereits beschrieben, verschlechtert der I-Anteil im Regler potentiell das dynamische Regelungsverhalten, weil es die integrale Regelungsstrecke verstärkt. Je nach Aufgabe kann die Eleminierung der

bleibenden Regelungsabweichung wichtiger sein, sodass ein PID-Regler verwendet werden sollte. In diesem Zusammenhang ist unterhalb in der Abbildung 55 das Regelungsverhalten des Systems mit einem PID-Regler aufgezeigt. Vergleicht man diese mit der vorigen Abbildung ist schnell zu erkennen, dass der integrale Anteil im Regler der bleibenden Regelungsabweichung recht stark entgegenwirkt. Wie schnell die Stellwertstörung unterdrückt wird, hängt dabei von der Höhe des I-Regelwerts ab. Bei dieser Messung wurde ein I-Faktor von 50 verwendet. Ist die Stellwertstörung konstant, kann auch ein geringerer I-Regelwert für den Regler verwendet werden, weil die Stellwertstörung während der Regelungszeit nur einmal kompensiert werden muss. Damit könnte ein nahezu gleichbleibendes dynamisches Regelungsverhalten mit einer Kompensation von Stellwertstörungen kombiniert werden.

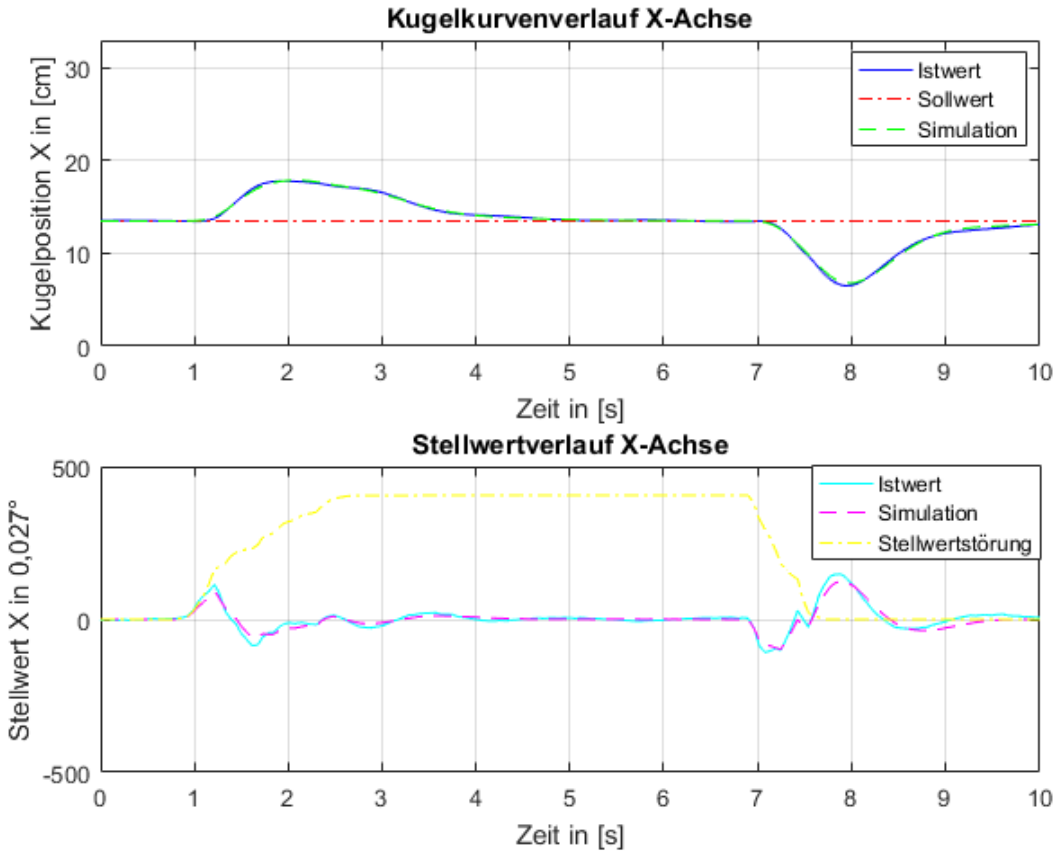


Abbildung 55: Kugelkurven- und Stellwertverlauf bei einer Stellwertstörung und P = 50, D = 14, I = 50

7.5 Ausgangsgrößenstörung

Eine Ausgangsgrößenstörung bezeichnet eine Störung der Regelgröße, die die Ausgangsgröße der Regelungsstrecke darstellt. Die Regelgröße vom Tabledance ist die Kugelposition, bei der z.B. mit der

Hand eine Störung eingebracht werden kann. Im Gegensatz zur Eingangsgrößenstörung kann hier die Ausgangsgrößenstörung nicht über das Regelungsprogramm erzeugt und größentechnisch erfasst werden, weshalb keine gleichzeitige Simulation des Regelungsverhaltens auf eine Ausgangsgrößenstörung realisiert werden konnte. Nichtsdestotrotz kann das Regelungsverhalten des Systems auf eine Störung der Kugelposition begutachtet werden.
Die Abbildung 56 zeigt dazu einen Graphen, bei dem die Kugel auf dem Touchscreen zunächst vom konstanten Sollwert weg verschoben und anschließend los gelassen wurde. Die Kugel rollt verhältnismäßig schnell wieder zum Sollwert zurück und zeigt dabei noch ein annähernd aperiodisches Einschwingverhalten auf. Das Einschwingverhalten ist in einem solchen Fall deutlich besser als es bei einer sprunghaften Sollwertvorgabe wäre, was daran liegt, dass der Touchscreen beim Loslassen der Kugel bereits entsprechend dem P-Regelanteil geneigt ist und die Kugel nur entsprechend abbremsen muss. Damit wirkt die Regelung bei einer Störung der Kugelposition im statischen Fall eine Kraft entsprechend der Beschleunigungskraft, hervorgerufen durch die Neigung des Touchscreens, der Störung entgegen.

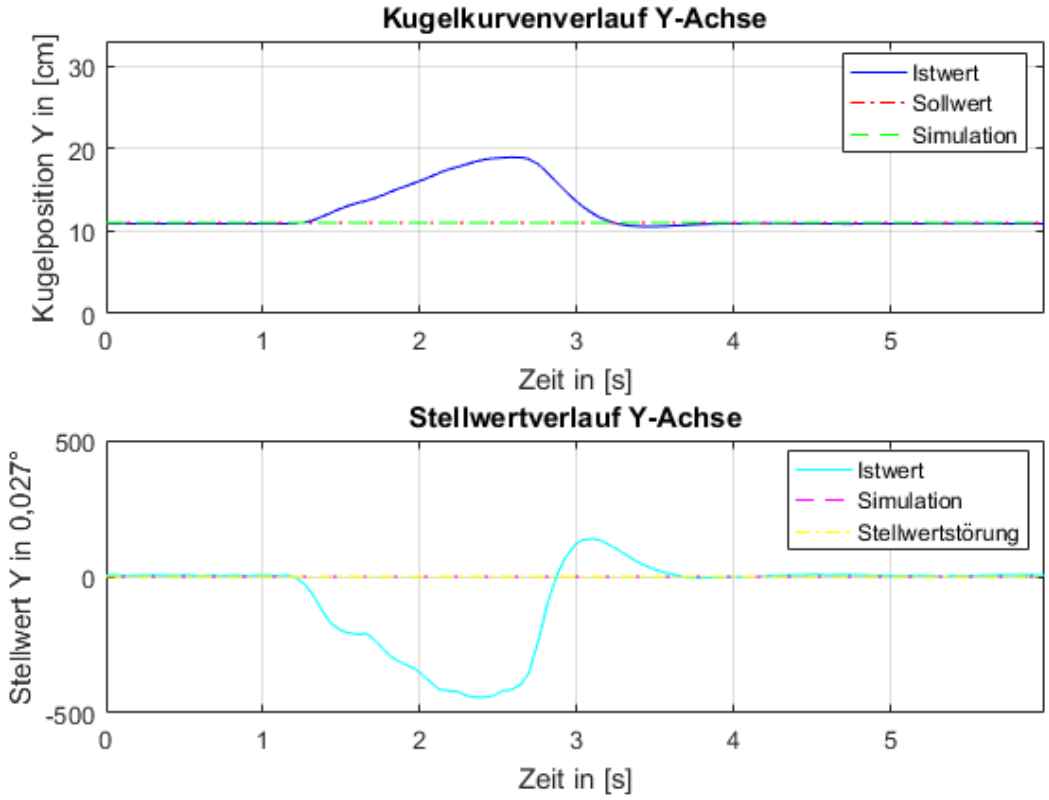


Abbildung 56: Kugelkurven- und Stellwertverlauf bei einer Ausgangsgrößenstörung und P = 50, D = 14

8. Messsystemverbesserungen

Mit den integrierten A/D-Wandlern konnte eine Kugelpositionserfassung mit einer Genauigkeit von ca. 11,5 bit bei 333 Hz erreicht werden. Diese ist dennoch nicht ausreichend, um das System bei akzeptablen Stellwertrauschen ohne Filterkoeffizienten für den differentiellen Anteil im Regler betreiben zu können. Der Filterkoeffizient verringert effektiv das Stellwertrauschen, fügt der Regelung aber funktionsbedingt eine Verzögerung zwischen der tatsächlichen Kugelposition und der verwendeten gefilterten Kugelposition ein. Um dieses Problem zu umgehen und eine noch verzögerungsärmere Regelung verwirklichen zu können, wurde die effektive Auflösung der Kugelpositionsmessung verbessert.

8.1 Analyse des Messsystems

Bei der Verbesserung der effektiven Positionsauflösung stellte sich die Frage, an welchen Stellen des Sensorsystems die Messung signifikant verbessert werden kann. Die Auflösung der Kugelposition ist nämlich nicht nur von der Messgenauigkeit der A/D-Wandler abhängig, sondern auch von der Signalqualität des Touchscreensensors. Diese wiederrum wird maßgeblich von der Qualität der Versorgungsspannungen beeinflusst. Messungen an der Spannungsausgabe der digitalen Pins zeigte auf, dass diese ein gewisses Spannungsrauschen aufweisen. Verglichen mit dem Spannungsrauschen an der Sensorleitung des Touchscreens ist diese aber deutlich geringer. Demzufolge verschlechtert der Touchscreen die Spannungsstabilität der Sensorleitung gegenüber der Versorgungsspannung. Mögliche Gründe dafür könnten z.B. elektromagnetische Einflüsse auf die Touchscreenflächen sein, wodurch Spannungsrauschen induziert werden kann.
Ausgehend von diesen Erkenntnissen hat sich ergeben, dass eine verbesserte Kugelpositionserfassung entweder durch einer besseren A/D-Wandlung, besserer Versorgungsspannung oder Verringerung der Störeinflüsse auf das Touchscreen erzielt werden kann. Die Störeffekte auf die Sensorspannung vom Touchscreen konnte leicht verringert werden, indem das Tabledance geerdet wurde. Dazu wurde ein Kabel an einem Aluminiumprofil des Tabledance befestigt und an einem Masseanschluss am Arduino angeschlossen. Anzumerken sein, dass schon die Annäherung einer Hand zum Touchscreen eine leichte Verschlechterung der Signalqualität zur Folge hat. Allein dieser Effekt zeigt auf, dass es sich beim resistiven Touchscreen um ein sensibles Sensorsystem handelt, sobald hochgenaue Messwerte gebraucht werden.
Es wurde auch versucht, die Versorgungsspannung des Touchscreensensors zu verbessern. Dazu wurde eine selbst entworfene Schaltung ausprobiert, bei der zwei CMOS-Invertierer, aufgebaut mit

jeweils einen P-Kanal und N-Kanal MOS-Feldeffekttransitor, eine von einer Referenzspannungsquelle bereitgestellte Spannung durchschalten. Die Transistoren wurden für das Schalten mit dem Mikrocontroller angesteuert. Letztendlich wurde damit nur die gleiche Versorgungsspannungsqualität wie mit dem Arduino selbst erreicht, weswegen diese Schaltung wieder verworfen wurde und nicht genauer darauf eingegangen wird.
Eine deutlich höhere Positionsauflösung konnte letztendlich mit einem externen A/D-Wandler erreicht werden. Zuerst wurde ein 16 bit A/D-Wandler von Adafruit in das System integriert, der mit einer effektiven Auflösung von 11,5 bit bei 333 Hz nur gleich gut wie der mit Überabtastung betriebene 12 bit A/D-Wandler des Arduino Due war und deshalb anschließend mit einem 24 bit A/D-Wandler ausgetauscht wurde. Darum soll nur kurz auf den A/D-Wandler von Adafruit eingegangen werden. Bei diesem handelt es sich um eine betriebsfertige Platine, die über den I²C-Bus mit dem Mikrocontroller kommunizieren kann und speziell auch für den Einsatz an Arduino Mikrocontrollern gedacht ist. Hauptschwierigkeit bei der Integration war die zugeschnittene Programmierung der Kommunikation über den I²C-Bus über spezielle Bibliotheken, die auf die eigenen Anforderungen abgeändert werden mussten. Davon abgesehen musste die Platine, die bei der Abbildung 57 gezeigt ist, nur mit den zwei Leitungen des Bussystems und an einer Versorgungsspannung angeschlossen werden. Der I²C-Bus ist ein bidirektionaler Bus mit relativ geringer Datenbandbreite, weswegen abgesehen vom A/D-Wandlerchip selbst nur geringe Messfrequenzen möglich sind. In diesem Fall ist die Messfrequenz auf maximal 800 Hz begrenzt, weshalb im Anwendungsfall auch keine Überabtastung möglich ist, womit die effektive Auflösung hätte verbessert werden können.

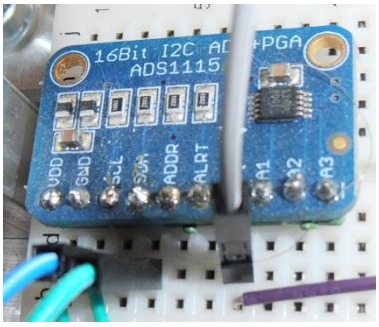


Abbildung 57: 16 bit A/D-Wandler

8.2 Aufbau und Anwendung des optionalen 24 bit A/D-Wandlers

Auf den Aufbau des A/D-Wandlers soll nur verhältnismäßig kurz eingegangen werden, weil es sich um eine Alternative zu den integrierten A/D-Wandlern des Arduino handelt und nicht für die Funktion des Regelungssystems essentiell ist.

8.2.1 Aufbau des 24 bit A/D-Wandlers

Nach umfänglichen Recherchen und Überlegungen wurde eine eigene und hochauflösende A/D-Wandlerschaltung realisiert, weil hochwertige A/D-Wandler nur als einzelne Chipbausteine erhältlich

sind. Dieser A/D-Wandlerbaustein benötigt einige passende Zusatzkomponenten. In der Industrie ist es ohnehin üblich, den Mikrocontroller selbst und alle Beschaltungselemente auf einer Leiterplatte zu integrieren. Stattdessen wurde hier ein Steckboard verwendet, mit dem alle Komponenten flexibel verbunden werden können. Dazu mussten die Halbleiterchips auf entsprechende Adapterplatinen gelötet werden, die wiederrum kompatibel zu den Steckplätzen des Steckboards sind.
Ausgesucht aus einer Vielzahl von Halbleiterchips wurde der A/D-Wandler ADS1252U von Texas Instruments mit einer nominellen Auflösung von 24 bit, der eine effektive Auflösung bis zu 18 bit bei bis zu 40 kHz erreichen kann. Dieser kann mit einer speziell angepassten SPI-Buskommunikation mit dem Mikrocontroller kommunizieren. Dabei stellt der SPI-Bus einen recht unspezifizierten Bus mit grundsätzlich vier unidirektional arbeitenden Drähten dar. Diese werden MISO (Master In Slave Out), MOSI (Master Out Slave In), CS (ChipSelect) und SCKL (Serial Clock) genannt. Als Master ist in diesem Fall der Mikrocontroller gemeint, weil der den Bus bereitstellt, während Chips, die am Bussystem angeschlossen sind, als Slaves bezeichnet werden. Weil der A/D-Wandler selbst keine Daten empfängt und kein CS-Eingang für die Auswahl des aktiven Chips aufweist, sind nur die zwei Drähte MISO und SCKL für die Buskommunikation nötig. Mit dem MISO empfängt der Mikrocontroller die Daten vom A/D-Wandler, während die Übertragung mit der SCKL-Leitung synchronisiert wird.
Insgesamt hat der A/D-Wandler acht Anschlüsse und ist in der Bauform SO-8 gefertigt. Neben der Betriebsspannung, die auf 5 V festgelegt ist, benötigt der A/D-Wandler eine Referenzspannung und ein Taktsignal. Die aus dem Datenblatt entnommene Pinkonfiguration ist in Abbildung 58 gezeigt. Der A/D-Wandler arbeitet mit der CMOS-Logik-Spannung von 5 V, wohingegen der Arduino Due eine Spannung von 3,3 V verwendet, womit auch die Bussysteme des Arduino arbeiten. Das hat zur Folge, dass für fehlerfreie SPI-Buskommunikation ein Pegelwandler verwendet werden muss, der 5 V zu 3,3 V und umgekehrt konvertiert. Implementiert wurde der Pegelwandler TXB0104 von Adafruit, der auch für hohe Frequenzen geeignet ist. Das ist nötig, weil der verwendete SPI-Bus mit bis zu 21 MHz betrieben werden kann.

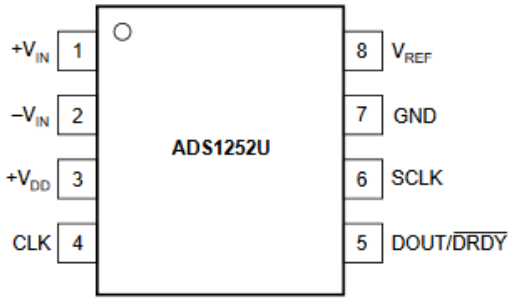


Abbildung 58: Pinkonfiguration vom ADS1252U Quelle [5]

Am Pin 4 des A/D-Wandlers ist der Anschluss eines CMOS-kompatiblen Taktsignals vorgesehen. Dieser ist nicht nur für den Betrieb nötig, sondern legt gleichzeitig in Abhängigkeit von der verwendeten Taktfrequenz die Messfrequenz fest. Dabei benötig der A/D-Wandler insgesamt 384 Takte für die Messung der Spannung. Die Übertragung des Messwerts über den SPI-Bus ist vom Taktsignal unabhängig. Angewendet wurde ein CMOS-kompatibler Taktoszillator mit einer Frequenz von 7,3728 MHz, womit sich eine Messfrequenz von 19,2 kHz ergibt, die sich gut für die vorliegende Messaufgabe eignet. Zu berücksichtigen ist dabei, dass der A/D-Wandler selbst einen digitalen Filter

integriert hat, der die letzten fünf erfassten Messwerte mit einbezieht. Deshalb sind mindestens fünf A/D-Konvertierungen nötig, bis sich der Messwert komplett auf eine veränderte Spannung einstellt. Als Referenzspannungsquelle wurde ein hochwertiger Chip ausgewählt, weil diese maßgeblich die Qualität der Messung mit beeinflusst. Ausgesucht wurde eine Referenzspannungsquelle von Analog Devices mit der Bezeichnung ADR443. Diese ist dank der neuartigen XFET-Technologie besonders rauscharm und benötigt, bezogen auf die ausgegebene Referenzspannung, eine nur gering höhere Versorgungsspannung. Damit konnte die vom Arduino bereitgestellte 5 V Spannungsversorgung als Versorgungsspannung der Referenzspannungsquelle verwendet werden. Ausgabespannung der Referenzspannungsquelle beläuft sich auf 3,00 V, wodurch auch der Messbereich des A/D-Wandlers auf 0 – 3 V festgelegt ist. Laut Datenblatt können für den A/D-Wandler Referenzspannungen von 0,5 – 5 V verwendet werden. 3 V als Referenzspannung ist diesbezüglich am sinnvollsten, weil der gesamte Messbereich vom Sensor gut ausgenutzt und dadurch die effektive Positionsauflösung erhöht wird.

Neben der Versorgungsspannung muss die Referenzspannungsquelle mit mehreren Kondensatoren beschaltet werden. Im Datenblatt sind dazu nur grobe Richtwerte für die Kapazität der Kondensatoren gegeben, weil die besten Werte bei jeder Schaltung anders sind und durch Ausprobieren herausgefunden werden müssen.

Den gesamten Aufbau des A/D-Wandlers, der in einem schließbaren Aluminiumschutzkasten eingebaut wurde, ist mit der Abbildung 59 gezeigt. Außer den Kondensatoren sind alle Bauelemente bei der Schaltung beschriftet.

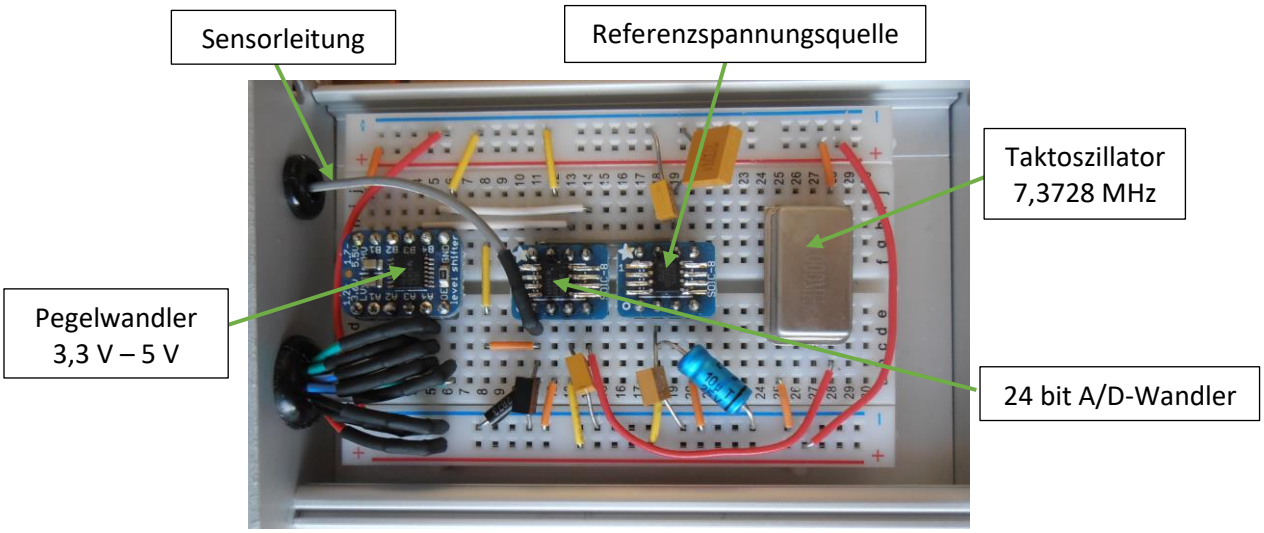


Abbildung 59: 24 bit A/D-Wandlerschaltung

Als letztes musste die Kommunikation über den SPI-Bus zwischen dem A/D-Wandler und dem Arduino Due in Simulink programmiert werden. Die spezifizierte SPI-Buskommunikation ist im

Datenblatt des A/D-Wandlers beschrieben. Die Abbildung 60 zeigt diesbezüglich die Zustände des A/D-Wandlers über die 384 Takte eines Messzyklus.

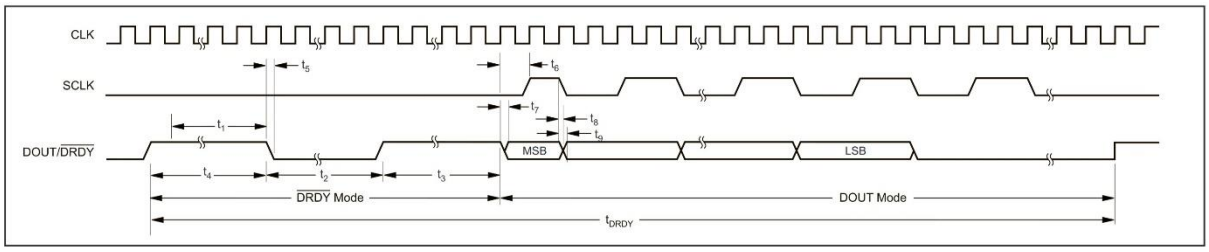


Abbildung 60: Taktdiagramm für die SPI-Kommunikation des ADS1252U, Quelle [5]

Der aktuelle Messwert kann nur während des „DOUT Mode“ übertragen werden, während die Takte davor für die Bereitstellung des neuen Messwerts verwendet werden. Der Mikrocontroller musste so programmiert werden, dass er die Übertragung des 24 bit großen Messwerts innerhalb der Zeit des „DOUT Mode“ bewerkstelligt, weil ansonsten fehlerhafte Werte übertragen werden. Um den Zeitpunkt des „DOUT Mode“ zu erkennen, musste das „DOUT/DRDY“-Signal mit einem Pin am Arduino digital ausgewertet werden. Dabei misst der Mikrocontroller innerhalb einer while-Schleife bis das „DOUT/DRDY“-Signal auf logisch 1 schaltet. Dadurch kann man feststellen, dass der A/D-Wandler sich in den im Diagramm benannten Zeitpunkten t_4 oder t_3 befindet. Durch warten von wenigen Mikrosekunden ist der A/D-Wandler im „DOUT Mode“ und der Messwert kann über den SPI-Bus übertragen werden. Dabei werden drei Byte übertragen, die von der „SCLK“-Leitung gesteuert werden. Die Übertragung benötig nur einen Bruchteil der Zeit des „DOUT Mode“, ist aber abhängig von der in Simulink eingestellten SPI-Frequenz. Nach der Übertragung werden die drei Bytes innerhalb einer S-Funktion zum endgültigen 24 bit großen Messwert zusammengefügt. Das Regelungsprogramm abseits der Positionserfassung ist gleich zum Programm für die integrierten 12 bit A/D-Wandler aufgebaut.

Die Realisierung des Programms gestaltete sich insgesamt als aufwendig, weil das zuerst verwendete Matlab2015a keine Blöcke für den SPI-Bus bereitstellt. Die SPI-Übertragung konnte aus nicht nachvollziehbaren Gründen auch nicht erfolgreich mit einer S-Funktion implementiert werden, was jedoch am Arduino Mega möglich war. Die Nachfolgeversionen Matlab2015b und Matlab2016a haben einen Simulinkblock für die Kommunikation per SPI, der für das Regelungsprogramm verwendet werden konnte. Mit diesen Versionen gab es jedoch regelmäßig Verbindungsabbrüche zum Mikrocontroller. Nach umfangreicher Analyse konnte herausgefunden werden, dass es sich nicht um ein Fehler im eigenen Regelungsprogramm handelt, sondern um ein Bug in Simulink. Dieser betrifft nur den Arduino Due im Modus „External“ und tretet nur bei speziellen und komplexeren Simulinkmodellen häufiger auf, auch wenn der SPI-Bus nicht verwendet wird. Der Bug wurde dem

Kundendienst von Matlab gemeldet, die zunächst Workarounds erarbeiteten, die aber nicht zuverlässig funktionierten. Erst bei der Nachfolgeversion Matlab2016b konnte der Fehler behoben werden, der laut Matlab an einer fehlerhaften Implementierung der SPI-Funktionalität zurückzuführen war. Wegen dem und anderen Verbesserungen wurde letztlich für alle erstellten Programme die Matlabversion Matlab2016b verwendet.

8.2.2 Anwendung des A/D-Wandlers

Mit dem gebauten A/D-Wandler konnte eine effektive Positionsauflösung von bis zu 14 bit erreicht werden, was unter der Rücksichtnahme des ungenauen Sensorsignals gut ist. Diese Auflösung von 14 bit ist ausreichend, um das System auch ohne Filterkoeffizienten betreiben zu können.
Entgegen der Vermutung verschlechtert sich das Regelungsverhalten ohne Verwendung des Filterkoeffizienten erheblich, was auch die Abbildung 61 zeigt, bei der das Regelungssystem mit normalen Regelungsparametern ohne Filterkoeffizienten betrieben wurde.

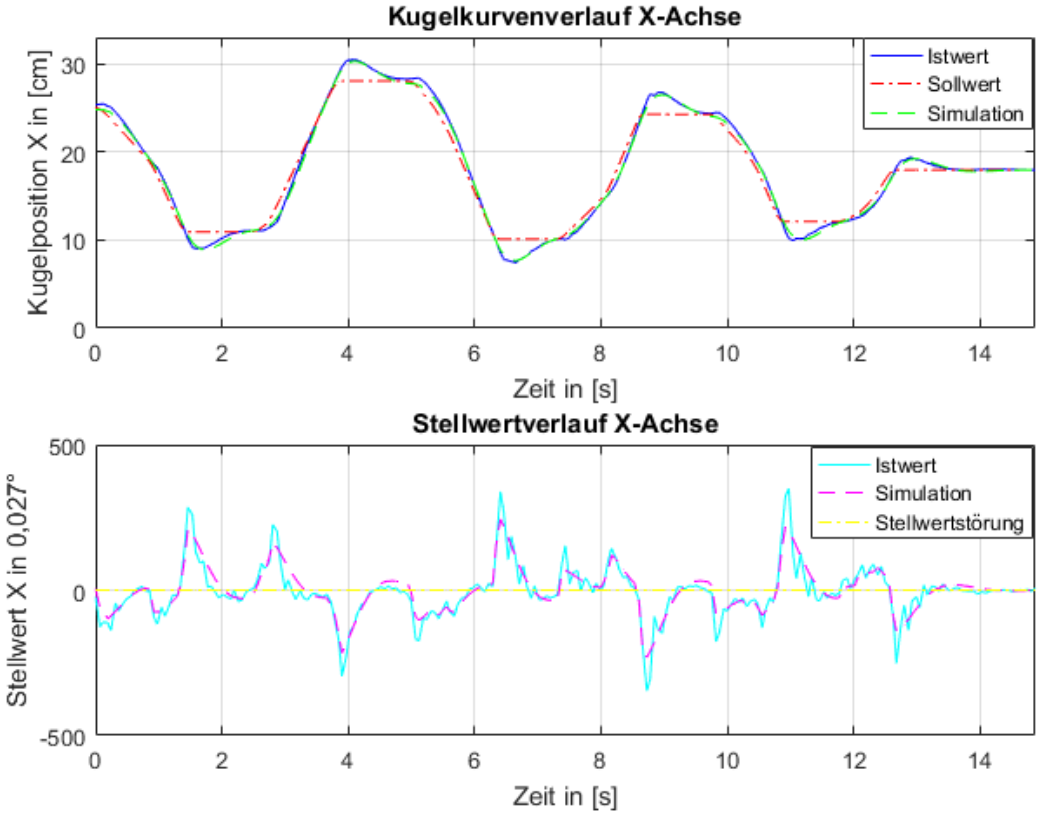


Abbildung 61: Kugelkurven- und Stellwertverlauf der X-Achse bei P = 50, D = 14 ohne Filterkoeffizient

Besonders auffallend ist das zittrige Stellwertverhalten, das sich zwar nicht so stark auf den Kurvenverlauf der Kugel auswirkt, die Regelung aber fast instabil werden lässt.
Anhand von Beobachtungen konnte auch der Effekt nachvollzogen werden, der dieses ungewöhnliche Verhalten erklärt.
Ursache ist wieder die Zentrifugalkraft, die die Kugel vor allem bei schnellen Neigungswinkeländerungen entgegen dem Sollwert verschiebt. Auf diese abrupte Verschiebung der Kugelposition entgegen dem Sollwert wirkt der D-Anteil verstärkend zum P-Anteil im Regler, wodurch der Effekt sich nochmals selbst verstärkt. Die Stellwertänderungen aufgrund des differentiellen Anteils im Regler haben selbst wieder weitere Reaktionen des D-Reglers zur Folge, weil sich die Geschwindigkeit der Kugel wegen der Zentrifugalkraft oft abrupt ändert. Auch Elastizitäten im Antrieb tragen durch viele schnelle Stellwertänderungen zum Aufschwingen bei. In diesem Sinne verursacht der D-Anteil im Regler bei dieser Konfiguration ein überreagierendes Schwingen, obwohl dieser Parameter eigentlich für die Dämpfung sorgt. Neben dem Stellwertrauschen wird auch dieser Effekt vom Filterkoeffizienten verringert. Grund dafür ist, dass die durch die Zentrifugalkraft hervorgerufenen abrupten Geschwindigkeitsänderungen der Kugel durch die Mittelung mehrerer Messwerte über den Filterkoeffizienten abgeschwächt werden, wodurch auch die Stellwertreaktion des differentiellen Anteils im Regler verringert wird. Daraus resultierend entsteht ein trägeres Reaktionsverhalten der Regelung mit einer gleichmäßigeren Bewegung der Kugel. Der Effekt mit der Zentrifugalkraft bedingt also, dass eine latenzarme Regelung ohne Filterkoeffizienten entgegen der Vermutung zu einem schlechterem Regelungsverhalten führt als wenn ein Filterkoeffizient in geeigneter Höhe eingesetzt wird. Neben dem Filterkoeffizienten kann die Auswirkung der Zentrifugalkraft auch mit der Stellwertänderungsgeschwindigkeitsbegrenzung beeinflusst werden. Eine stärkere Begrenzung verringert die auf die Kugel wirkende Zentrifugalkraft, wodurch die Kugel langsamer oder gar nicht mehr entgegen dem Sollwert ausgelenkt wird. Gleichzeitig kann das Stellglied dem Stellwert des Reglers zu langsam folgen, wodurch das Regelungsverhalten wiederrum erheblich verschlechtert wird. Deshalb kann dieser Parameter nicht sinnvoll gegen den Störeffekt verwendet werden.
Die Stärke der auf die Kugel wirkenden Zentrifugalkraft wird abseits der Neigungsgeschwindigkeit des Touchscreens auch vom Abstand der Kugel zur Touchscreenmitte beeinflusst. Mit zunehmenden Abstand zur Mitte des Touchscreens vergrößert sich auch entsprechend die Zentrifugalkraft. Demnach begünstigt der große Touchscreen den Störeffekt.
Im Simulationsmodell sind die Auswirkungen der Zentrifugalkraft nicht berücksichtigt, weswegen die Simulation besonders bei den Stellwerten nicht mehr gut mit dem Istwert übereinstimmt, während die Simulation bei gleichen Regelungsparametern mit einem entsprechenden Filterkoeffizienten sehr gute Ergebnisse liefert.

Für eine optimale Regelung ist es wichtig, den richtigen Wert für den Filterkoeffizienten zu wählen. Einerseits soll der Filterkoeffizient zur Reduktion des Störeffekts so klein wie möglich sein (starke Filterung), anderseits soll dieser so groß wie möglich sein (schwache Filterung), damit die Regelung nicht träge und „schwammig“ wird. Die vom Filterkoeffizienten hervorgerufene Latenz bedingt, dass der D-Regler beim Einschwingen der Kugel zu spät abbremst, wodurch die Dämpfung im Einschwingverhalten abnimmt. Dieser Effekt ist vor allem bei Filterkoeffizienten von 12 und darunter stark bemerkbar. Je nach Einsatzzweck der Regelung und Wertebereich der Regelparameter kann ein anderer Filterkoeffizient sinnvoll sein, wobei ein Filterkoeffizient von 20 als Kompromiss zwischen den beiden Störeffekten gute Regelungsergebnisse liefert, die ähnlich zu denen in den vorigen Kapiteln sind. Bei einem D-Regelwert von 5 oder darunter ist der Störeffekt aufgrund der Zentrifugalkraft auch nicht mehr ausschlaggebend, wodurch hier die Regelung auch ohne Filterkoeffizienten betrieben werden kann.
Wegen dem Umstand, dass das beste Regelungsverhalten unter Verwendung des Filterkoeffizienten erreicht wird, bietet der gebaute externe A/D-Wandler keine direkten Vorteile. Da die internen A/D-Wandler des Arduino robuster gegen Störeffekte sind, werden diese in der Grundkonfiguration verwendet. Soll der externe 24 bit A/D-Wandler verwendet werden, muss das entsprechende Programm für den 24 bit A/D-Wandler verwendet und einige Kabel eingesteckt werden. Diese sind ausgesteckt, weil offensichtlich die 5 V Spannungsversorgung des externen A/D-Wandlers die Messgenauigkeit der integrierten A/D-Wandler des Arduino stört. Zum Umstecken wie folgt vorgehen: weiss -> 3,3 V; rot -> 5 V; 2 x schwarz -> GND; gelbe Sensorleitung am Touchscreen mit grauer Sensorleitung austauschen.
Nachfolgend sind die Programm- und Dateinahmen der wichtigsten Programme und Dateien zum Betreiben des Tabledance aufgelistet:

- Name des Programms für integrierten 12 bit A/D-Wandler: Arduino12bit_Tabledance.slx
- Name des Programms für externen 24 bit A/D-Wandler: Arduino24bit_optional.slx
- Name des Programms für die Sollwertkurvenaufnahme: Arduino12bit_Kurvenaufnahme.slx
- Name der aufgenommenen Sollwertkurve: Sollwertkurve_DasistRegelungstechnik.mat

9 Zusammenfassung

Im Rahmen dieser Masterarbeit konnte erfolgreich ein verbessertes Nachfolgesystem für die Regelung einer Kugel auf vorgebbaren Bahnen realisiert werden. Dieses zeichnet sich durch ein sehr gutes Regelungsverhalten und einer flexiblen Bedienung mit Simulink aus. Die anschauliche Regelungsstrecke und die Regelung und Visualisierung über Simulink sind dabei Elemente, die das System besonders gut als regelungstechnischen Demonstrationsversuch eignen lassen. Auch ist es möglich, die Regelungsparameter während der Laufzeit zu ändern, sodass die Auswirkungen direkt beobachtet werden können.

Im Regelungsprogramm ist auch ein Simulationsmodell integriert, sodass die Positions- und Stellwerte während der Laufzeit des Programms verglichen werden können. Die Analyse des Regelungssystems hat gezeigt, dass das Simulationsmodell bei den meisten Regelungsparametern gute Ergebnisse liefert, obwohl physikalische Störeffekte, wie z.B. die Zentrifugalkraft, im Modell nicht berücksichtigt sind.

Das System besteht im Wesentlichen aus einem resistivem Touchscreen, dem Mikrocontroller Arduino Due und zwei Modellbauservomotoren mit Zahnriemengetriebe. Der Mikrocontroller erfasst beim Betrieb die Position der Kugel über das resistive Touchscreen und gibt entsprechende Stellwerte an die Modellbauservomotoren aus, die den Neigungswinkel des Touchscreen einstellen.

10 Quellenverzeichnis

[1] www.weltderphysik.de

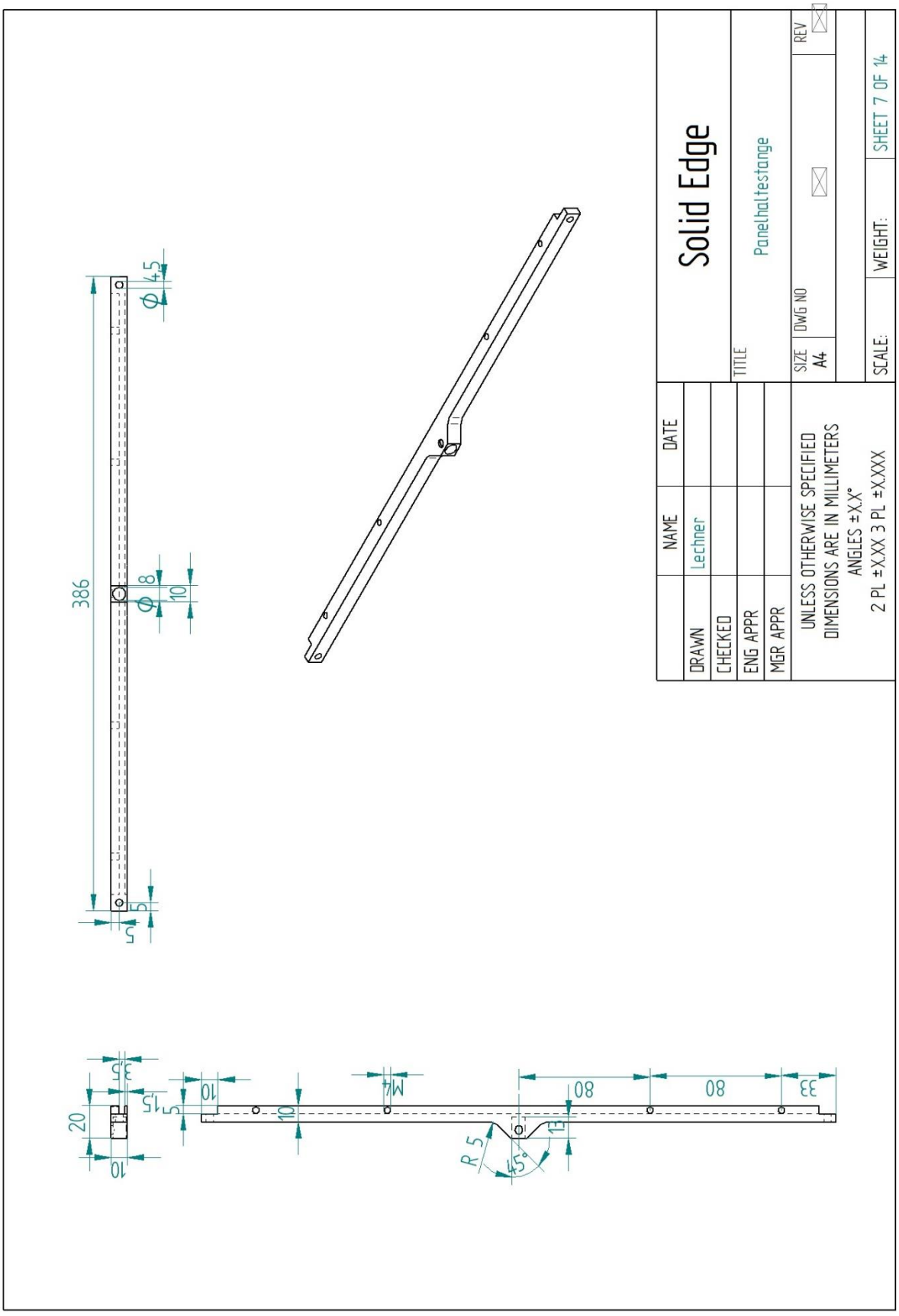
[2] www.conrad.de

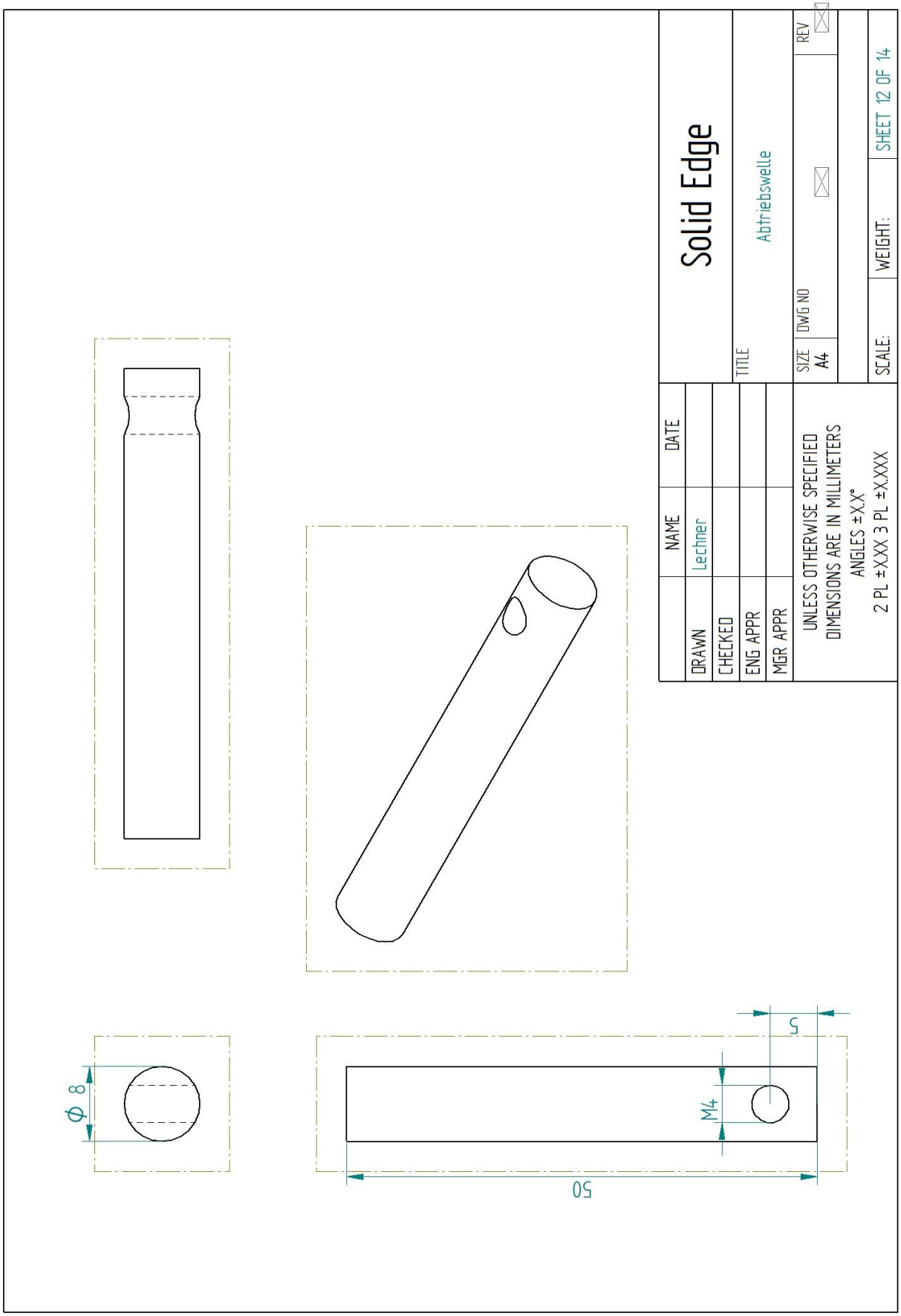
[3] https://de.wikipedia.org/wiki/Touchscreen

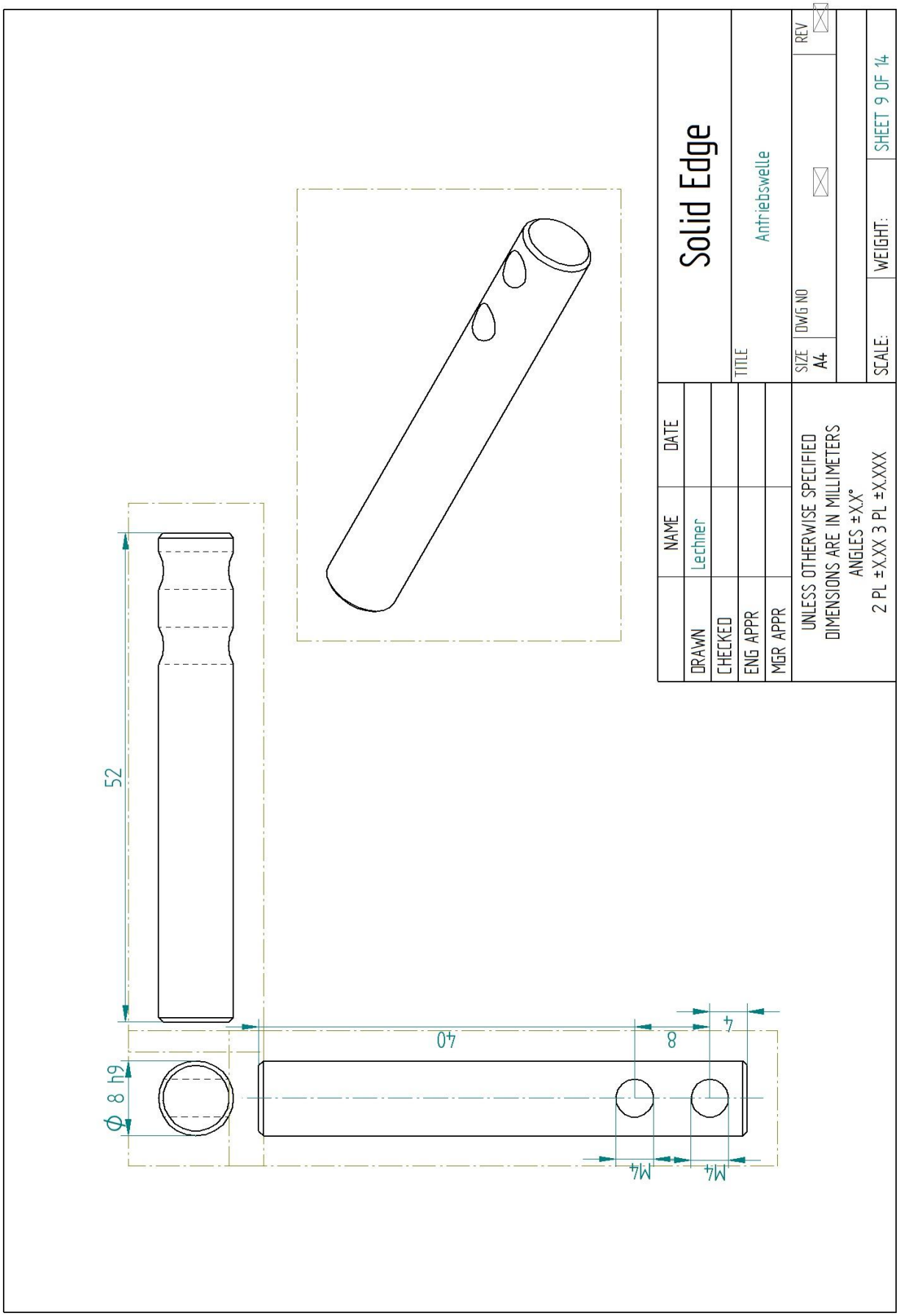
[4] www.mädler-shop.de

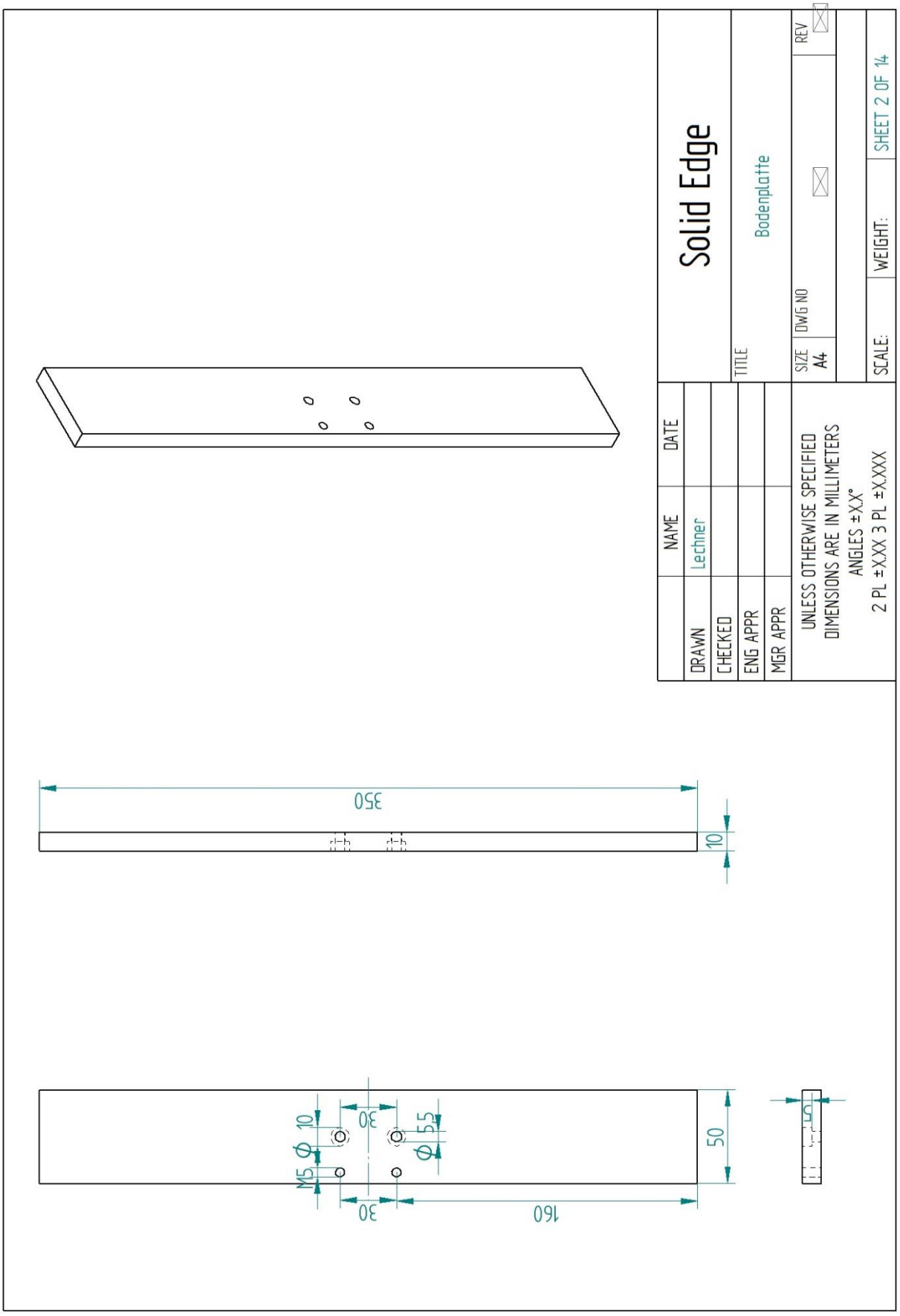
[5] Datenblatt Texas Instruments ADS1252U

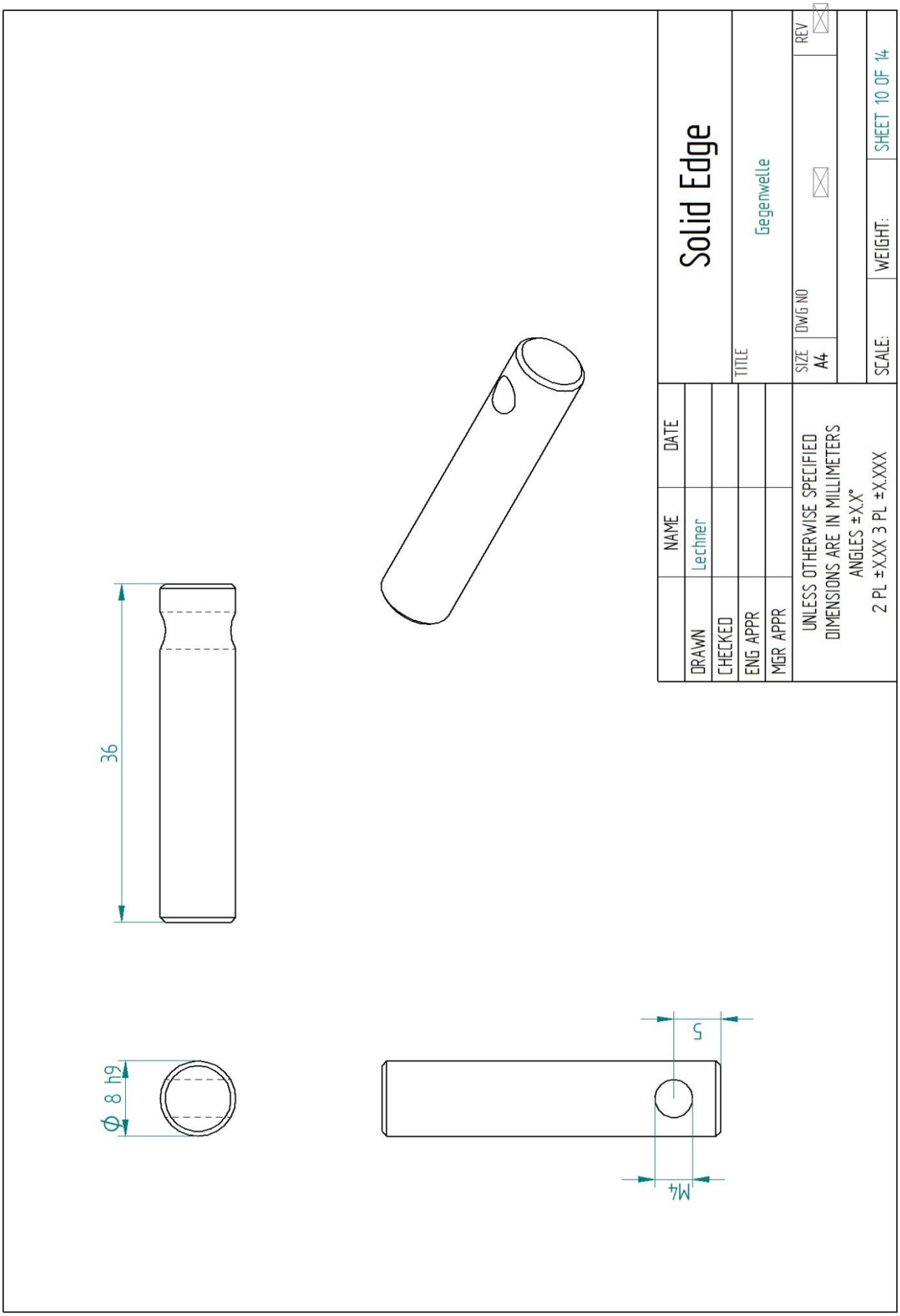
11 Anhang

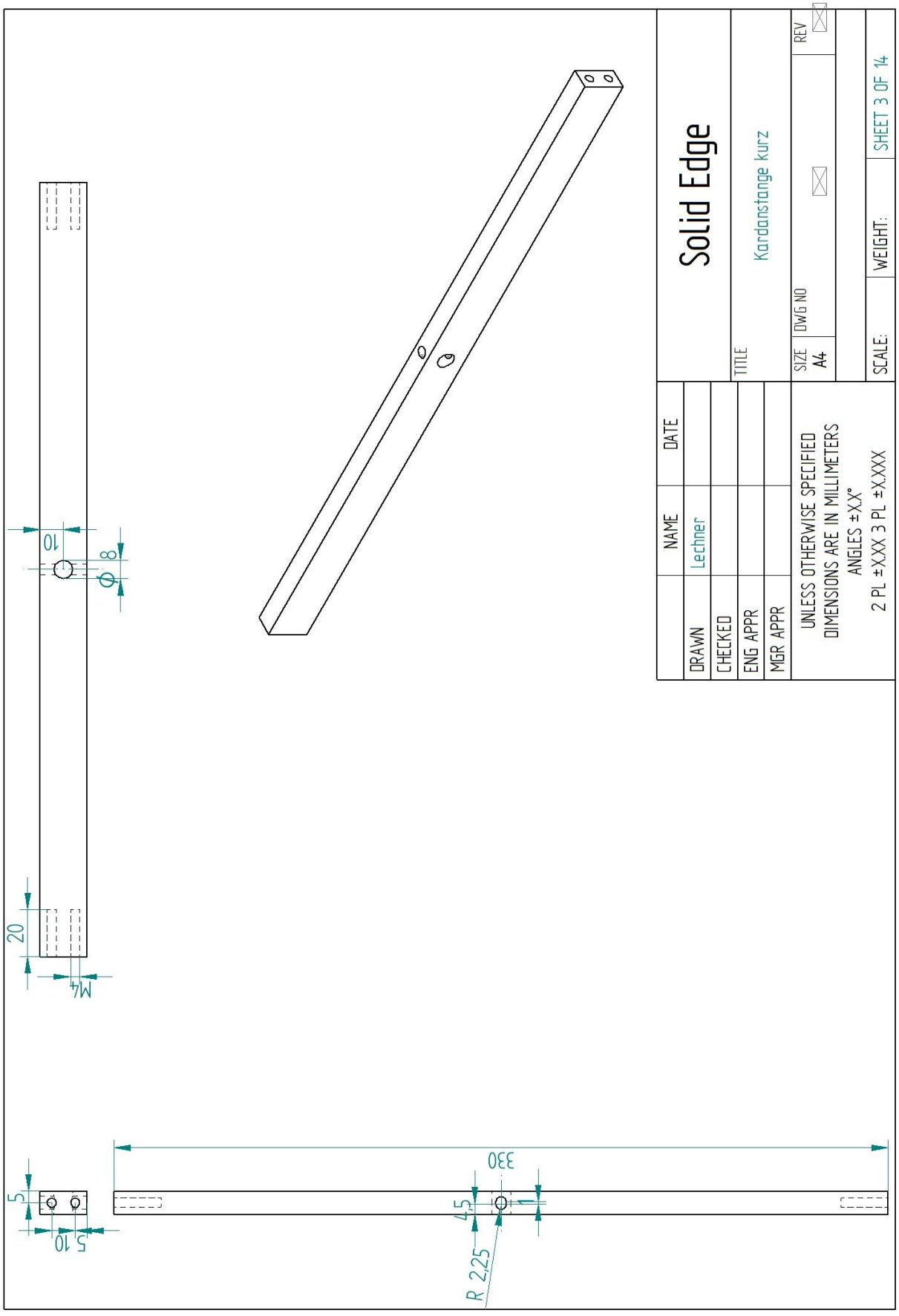

Solid Edge
Panelhaltestange
TITLE
SIZE A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 7 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
386
Φ4.5
Φ8
10
5
5
35
15
5
20
10
10
10
M4
80
80
33
13
R 5
45°

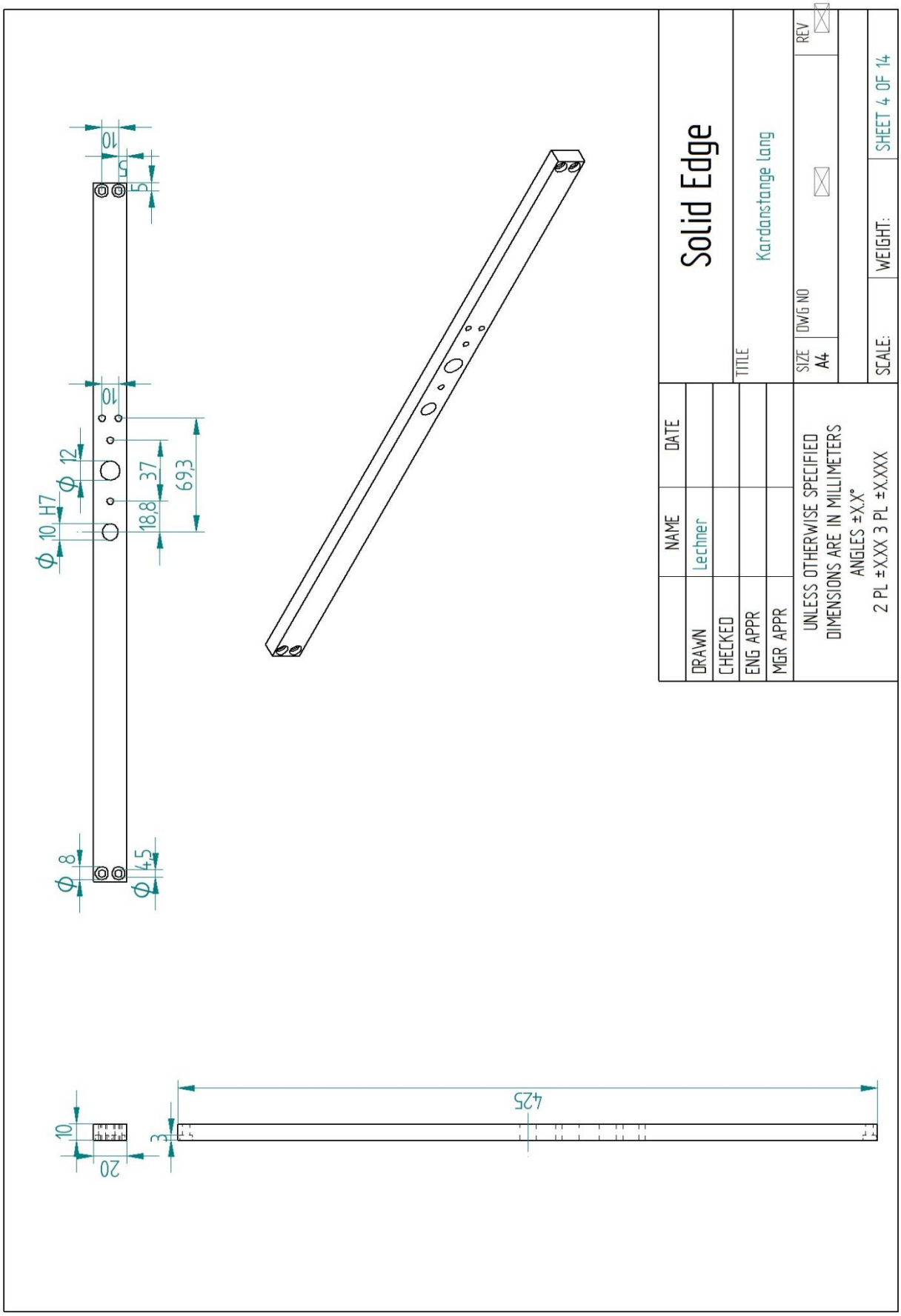

Solid Edge
TITLE
Abtriebswelle
SIZE A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 12 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
ϕ 8
M14
5
50

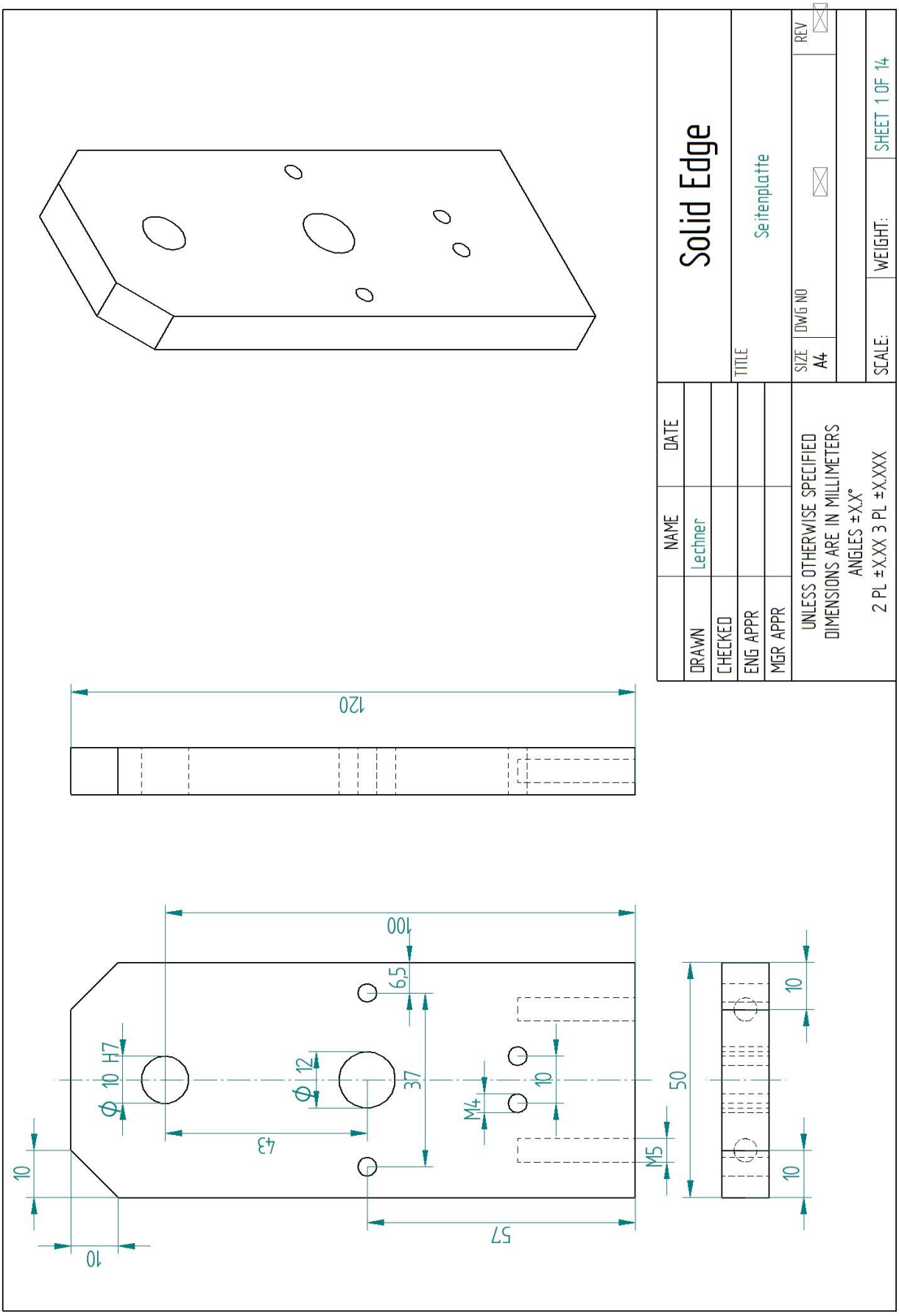

Solid Edge
Antriebswelle
TITLE
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 9 OF 14
52
40
8
4
ϕ 8 h9
M4
M4

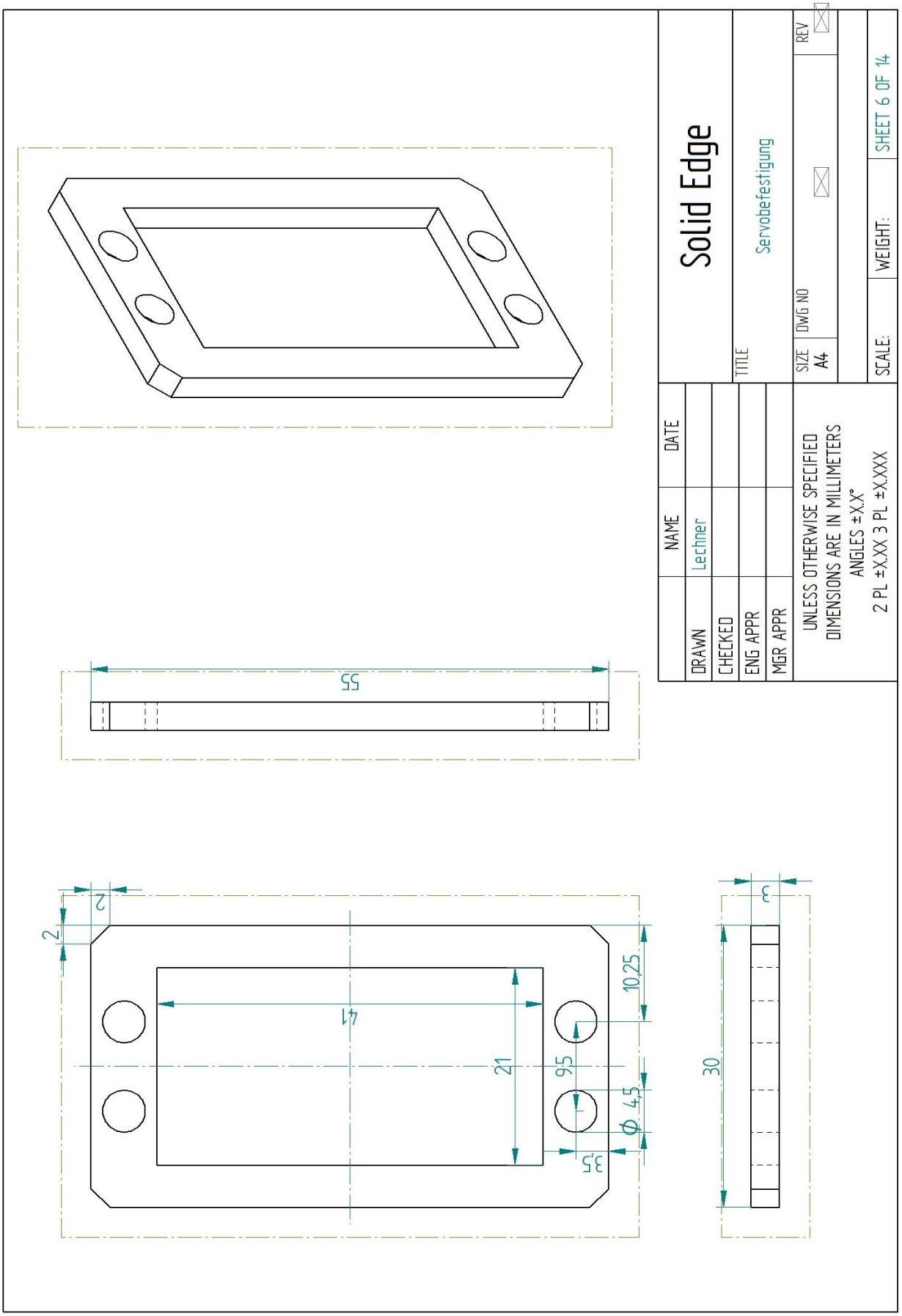

Solid Edge
Bodenplatte
TITLE
SIZE A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 2 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
350
10
50
30
160
10
30
5,5
M5

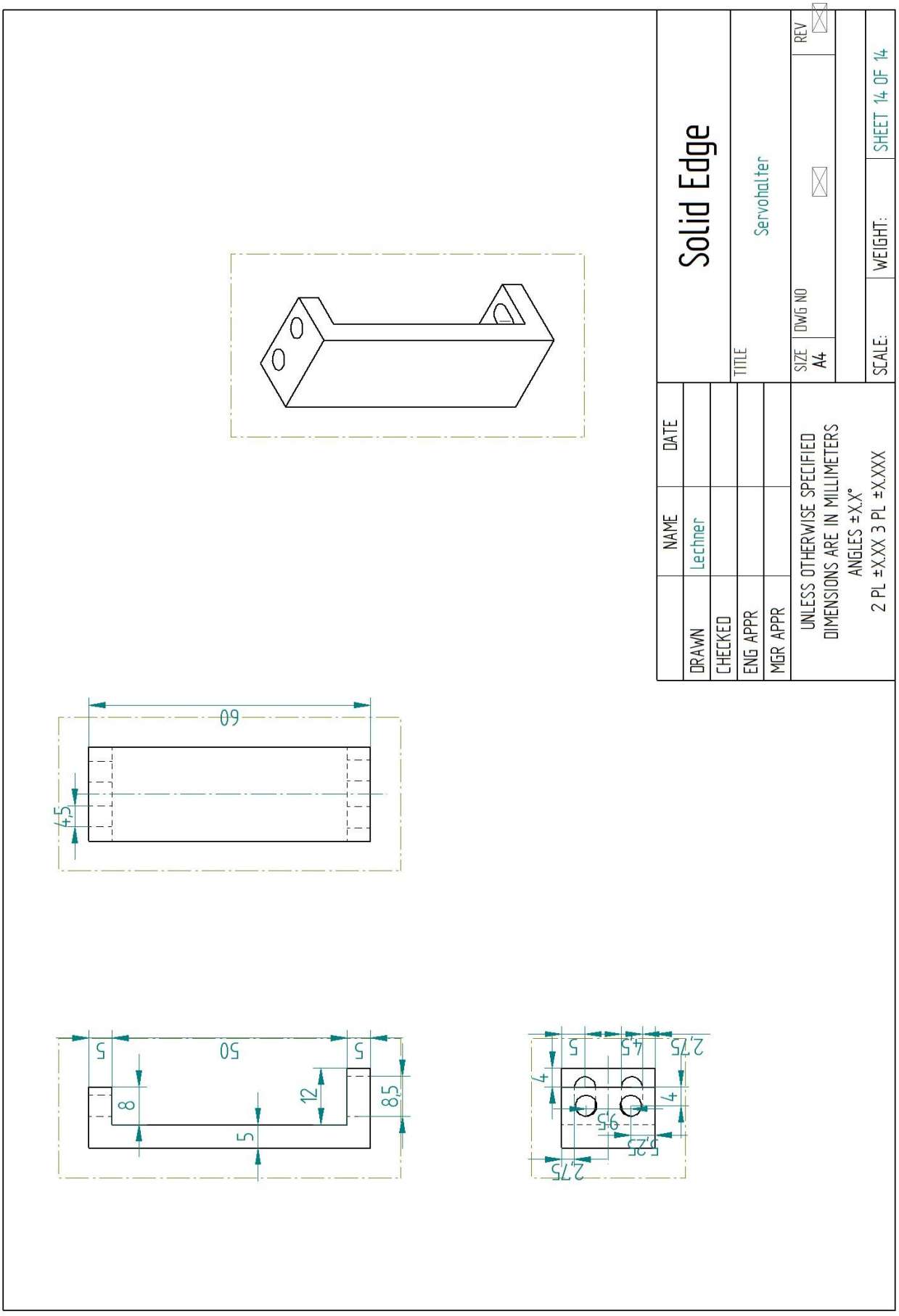

Solid Edge
TITLE
Gegenwelle
SIZE A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 10 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
36
Ø 8 h9
5
M4

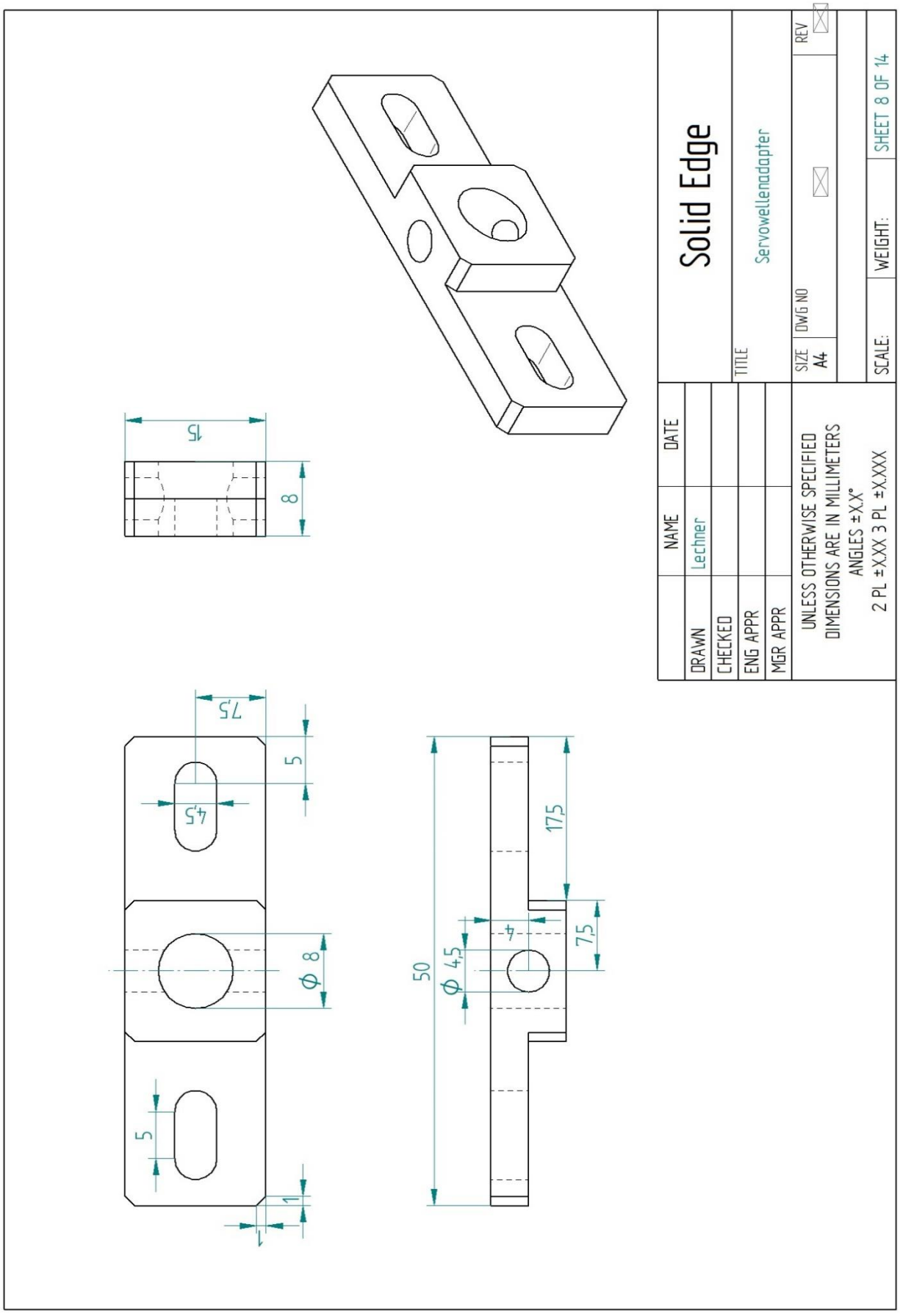

Solid Edge
TITLE
Kardanstange kurz
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 3 OF 14
10
Ø8
20
M4
330
45
1
R 225
5
5
10

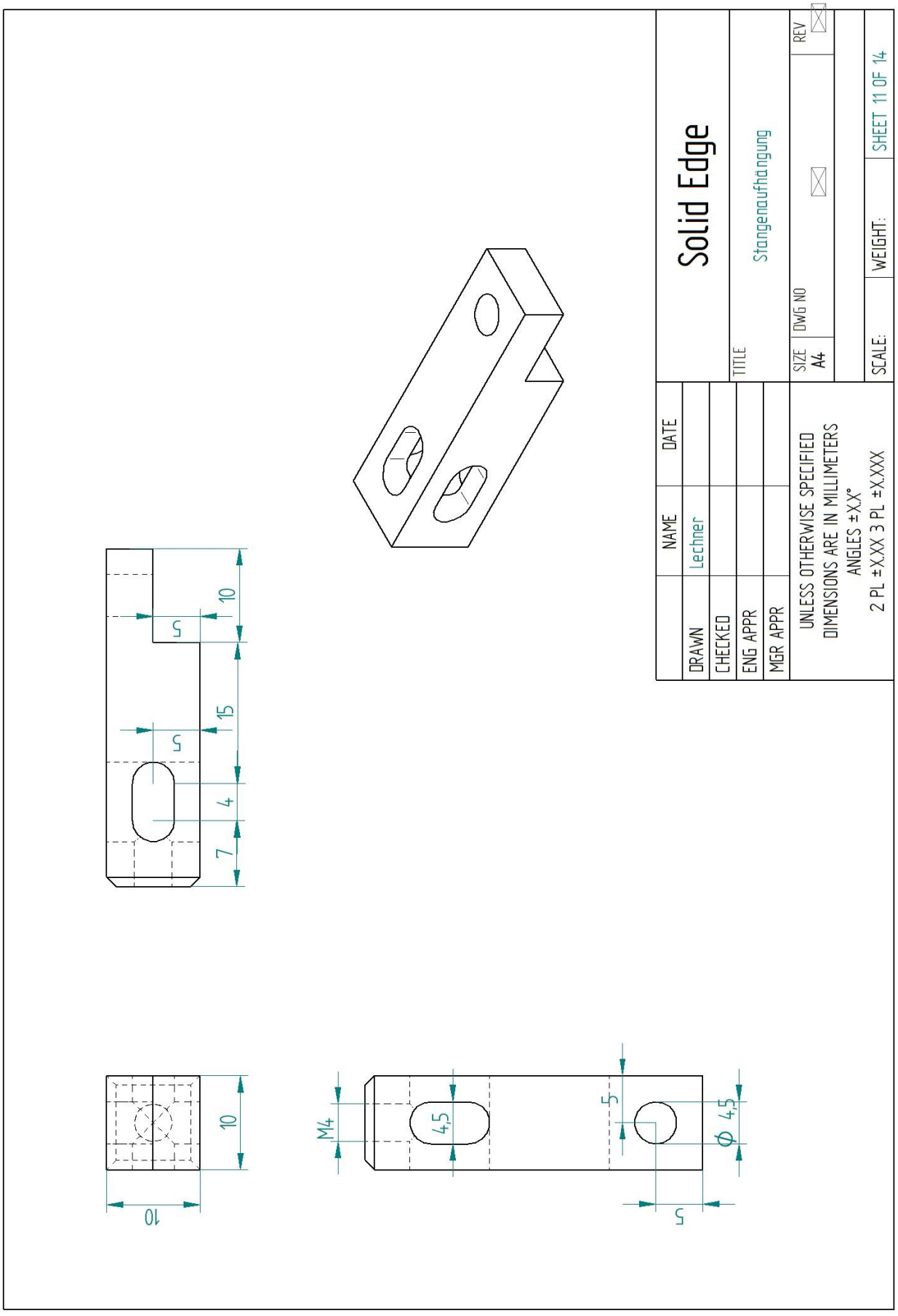

Solid Edge
TITLE
Kardanstange lang
SIZE A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 4 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
10
5
10
Φ 12
Φ 10 H7
37
69,3
18,8
Φ 8
Φ 4,5
425
10
20
3

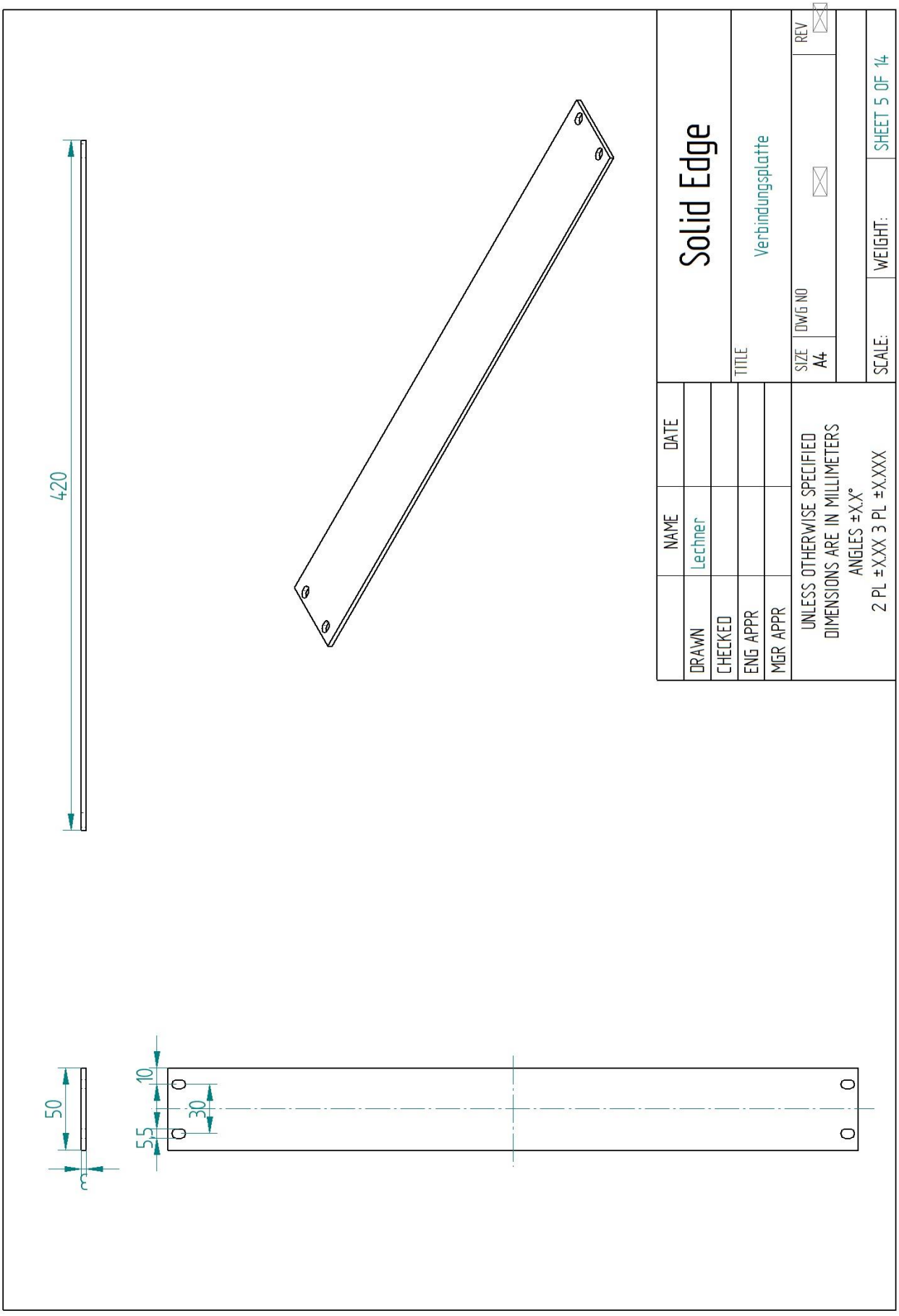

Solid Edge
TITLE
Seitenplatte
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 1 OF 14
120
100
6,5
Φ 10 H7
Φ 12
37
43
M4
10
M5
50
57
10
10
10
10


Solid Edge
Servobefestigung
TITLE
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 6 OF 14
55
2
2
41
21
10,25
9,5
Ø 4,5
5,5
3
30


Solid Edge
Servohalter
TITLE
SIZE
A4
DWG NO
REV
SCALE:
WEIGHT:
SHEET 14 OF 14
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
60
4,5
5
50
5
8
12
5
8,5
5
4,5
2,75
4
4
9,5
5,25
2,75


Solid Edge
TITLE
Servowellenadapter
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 8 OF 14
15
8
7,5
5
4,5
ϕ 8
5
1
50
17,5
7,5
4
ϕ 4,5


Solid Edge
TITLE
Stangenaufhängung
SIZE A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 11 OF 14
10
5
15
5
4
7
10
10
M4
4,5
5
Ø 4,5
5


420
50
10
30
5,5
Solid Edge
Verbindungsplatte
TITLE
SIZE
A4
DWG NO
REV
NAME
DATE
DRAWN
Lechner
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SCALE:
WEIGHT:
SHEET 5 OF 14