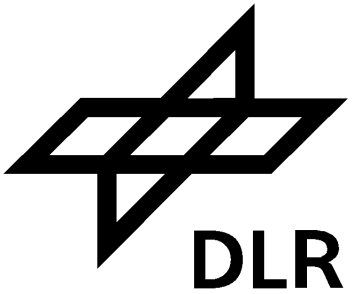
Analyse, Modellierung und Programmierung des Geige-spielens an einem Robotersystem

Praktikumsbericht

Björn Wendt*
Deutsches Zentrum für Luft- und Raumfahrt
Institut für Robotik und Mechatronik
Oberpfaffenhofen
82234 Wessling



von der
Fachhochschule für Technik und Wirtschaft zu Berlin
Fachbereich Ingenieurwissenschaften I
Studiengang Elektrotechnik

FHTW

01. Oktober 2008

Zug: D6ET
Semester: SS 2008

*Tel.: 0171/40 80 341; E-Post: triterium@yahoo.de

Inhaltsverzeichnis

Teil I.

Geschichte, Forschung und Entwicklung

1. Über das Deutsche Zentrum für Luft- und Raumfahrt[1]

Das Deutsche Zentrum für Luft- und Raumfahrt e.V. (DLR) ist ein institutielles Forschungszentrum der Bundesrepublik Deutschland für Luftfahrt, Raumfahrt, Energie und Verkehr. Es ist sowohl an nationalen und internationalen Kooperationen mit eingebunden. Darüber hinaus ist es im Auftrag der Bundesregierung und in Kooperation mit der europäischen Weltraumorganisation ESA (European Space Agency) für die Planung, Umsetzung und den Betrieb deutscher und europäischer Raumfahrtaktivitäten zuständig.

1.1. Geschichte

Göttingen 1907:

Ludwig Prandtl (Abbildung 1.1) gründete eine Modellversuchsanstalt der Motorluftschiff-Studiengesellschaft, aus der später die Aerodynamische Versuchsanstalt (AVA) hervorging. Heute wird diese als die älteste Vorgängerorganisation des DLR angesehen. 1969 entstand aus diesen Vorgängern und durch den Zusammenschluss mehrerer Forschungseinrichtungen wie der Deutschen Forschungsanstalt für Luftfahrt (DFL) die Deutsche Forschungs- und Versuchsanstalt für Luft und Raumfahrt (DFVLR). 1972 trat schließlich noch die Gesellschaft für Weltraumforschung (GfW) der DFVLR mit bei. 1989 wurde

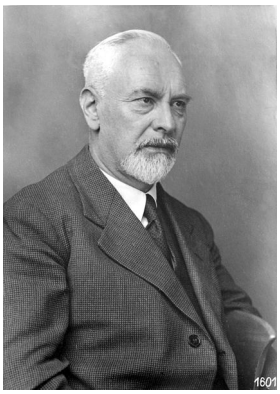


Abbildung 1.1.: Ludwig Prandtl 1937 [2]

die DFVLR in Deutsche Forschungsanstalt für Luft- und Raumfahrt (DLR) umbenannt

[1] [1]

und schließlich, durch die Fusion mit der Deutschen Agentur für Raumfahrtangelegenheiten (DARA) am 1. Oktober 1997 in Deutsches Zentrum für Luft- und Raumfahrt umbenannt.

1.2. Überblick

Das DLR ist eine institutielle Organisation mit 28 Instituten und Einrichtungen mit Hauptsitz in Köln (Abbildung 1.2). Die 13 Standorte sind deutschlandweit verteilt: Berlin, Bonn, Braunschweig, Bremen, Göttingen, Hamburg, Köln, Lampoldshausen, Neustrelitz, Oberpfaffenhofen, Stuttgart, Trauen und Weilheim. Darüber hinaus gibt es noch zahlreiche Außenstützpunkte, wie z. B. die Solarforschungseinrichtung in der südostspanischen Provinz Almeria. Insgesamt werden beim DLR circa 5.600 Mitarbeiter dauerhaft beschäftigt. Damit stellt sie eine der größten Forschungsanstalten Europas dar. Der jährliche Etat des DLR für die eigenen Forschungs- und Entwicklungsarbeiten sowie für Betriebsaufgaben liegt bei circa 670 Millionen Euro. Etwa ein Drittel davon sind im Wettbewerb eingeworbene Drittmittel. Weitere 550 Millionen Euro an deutschen Fördermitteln verwaltet das DLR für die Europäische Weltraumorganisation (ESA).

Abbildung 1.2.: DLR-Hauptsitz in Köln [3]

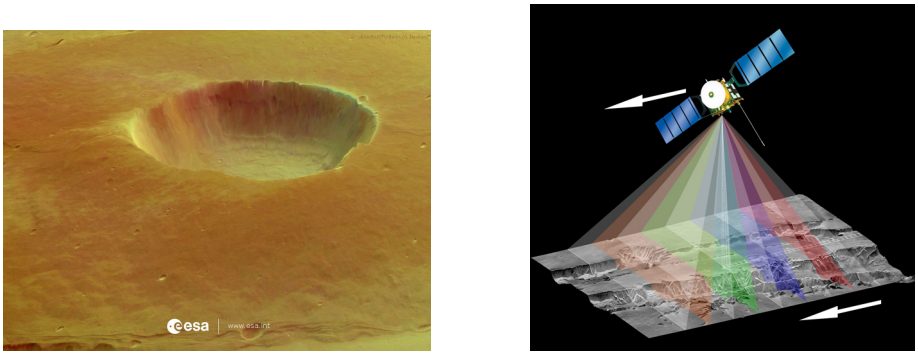
1.3. Forschung

Die Forschungsaktivitäten des DLR sind außerordentlich zahlreich. Ursprünglich nur auf die Luftfahrt bezogen, umfasst sie heute die Erforschung von Erde und Sonnensystemen, bis hin zu extrasolaren Planeten, Asteroiden und Kometen, die Forschung für den Erhalt der Umwelt und umweltverträgliche Technologien, die Weiterentwicklung neuer hochleistungsfähiger Materialien sowie neue Erkenntnisse in der Strömungslehre, Thermodynamik, Steuerungs- und Regelungstechnik bis hin zu kybernetischen Organismen.

Hier ein paar ausgewählte Beispiele:

1.3.1. HRSC auf Mars Express

Ein sehr erfolgreiches und prestigeträchtiges Projekt ist die hochauflösende Stereokamera HRSC, welche den deutschen Beitrag zur europäischen Mission Mars Express darstellt. Sie ist die erste digitale Stereokamera, die zusätzlich multispektrale Informationen liefert und über ein sehr hochauflösendes Objektiv verfügt. Entwickelt am Institut für Planetenforschung stellt sie eine wichtige Grundlage für zahlreiche Untersuchungen der Marsoberfläche dar.



(a) Aufnahme vom Mars (b) Funktionsprinzip der HRSC

Abbildung 1.3.: Die Stereokamera HRSC [4, 5]

1.3.2. Fernerkundung der Erde

Satelliten zur Fernerkundung liefern wichtige Daten zur Beobachtung und Untersuchung der Atmosphäre, der Land- und Ozeanflächen sowie der Eisflächen (Abbildung 1.4) der Erde. Die Untersuchungen werden im Fernerkundungsdatenzentrum (DFD) am Standort Oberpfaffenhofen durchgeführt. Mit ihnen ist es möglich, Veränderungen der Biosphäre der Erde zu erkennen und zu studieren. Darüber hinaus liefern sie unschätzbar wichtige Daten für die Katastrophenhilfe in betroffen Gebieten, wie z. B. beim schweren Tsunami Dezember 2004 im indischen Ozean, bei welchem durch die Erdbeobachtungssatelliten aktuelle detaillierte Karten über dem Gebiet erstellt werden konnten und den Hilfsor-

ganisationen zur Einschätzung der Lage sowie zur Orientierung dienten. Terra-SAR-X stellt hierbei den neuesten deutschen Erdbeobachtungssatelliten dar.

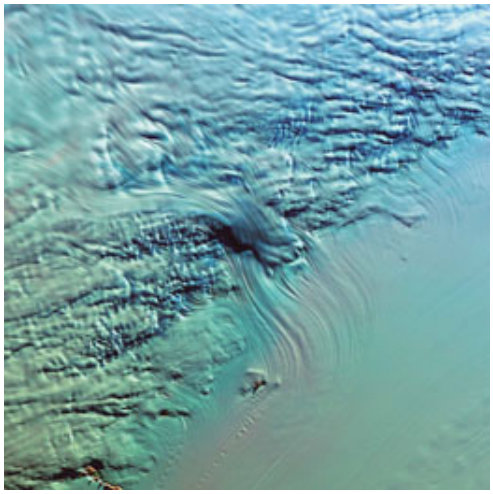


Abbildung 1.4.: Antarktis - Lambert-Gletscher [6]

1.3.3. Emissionsforschung

Ein wichtiges Aufgabengebiet in der Luftfahrtforschung stellt die Kohlendioxid- und Lärmemission von Flugobjekten dar. Trotz des rasch steigenden Luftverkehrs soll mithilfe neugewonnener Erkenntnisse u. a. auf dem Gebiet der Aerodynamik und Triebwerkstechnik ein Ansteigen des Schadstoffausstoßes als auch eine Zunahme der Lärmemission vermieden werden. Weitreichende Modellrechnungen zielen bereits darauf ab, den weltweiten Luftverkehr auf Wasserstoffantrieb umzurüsten.

1.3.4. Energieträger Wasserstoff

Ein sehr vielverspechendes Projekt stellt HYDROSOL dar. Mir diesem ist es den DLR-Wissenschaftlern erstmals gelungen, mithilfe von Solarenergie Wasser in seine zwei Komponenten Wasserstoff und Sauerstoff aufzuspalten, ohne jegliche Freisetzung von Kohlenstoffdioxid. Im Jahre 2007 wurde die Arbeitsgruppe dafür mit dem Descartes-Preis von der Europäischen Kommission ausgezeichnet.

1.3.5. Columbus

Im Jahre 2008 wurde das europäische Weltraumlabor Columbus (Abbildung 1.5) vom Weltraumbahnhof Cape Canaveral in den Orbit geschossen und an der Internationalen Raumstation ISS angedockt. Diese Ereignis stellte einen Meilenstein in der europäischen Raumfahrtgeschichte dar. Das in Bremen gebaute hochmoderne Forschungslabor eröffnet den Wissenschaftlern vielfältige Möglichkeiten der Forschung unter den Bedingungen der

Schwerelosigkeit. Das in Oberpfaffenhofen ansässige Kontrollzentrum des DLR ist für die Koordination, die wissenschaftlichen Arbeiten sowie für den Systembetrieb und die Lebenserhaltung zuständig. Alle wichtigen Daten fließen hier zusammen und werden von dort an die verschiedensten Institutionen und Einrichtungen weiterversandt.

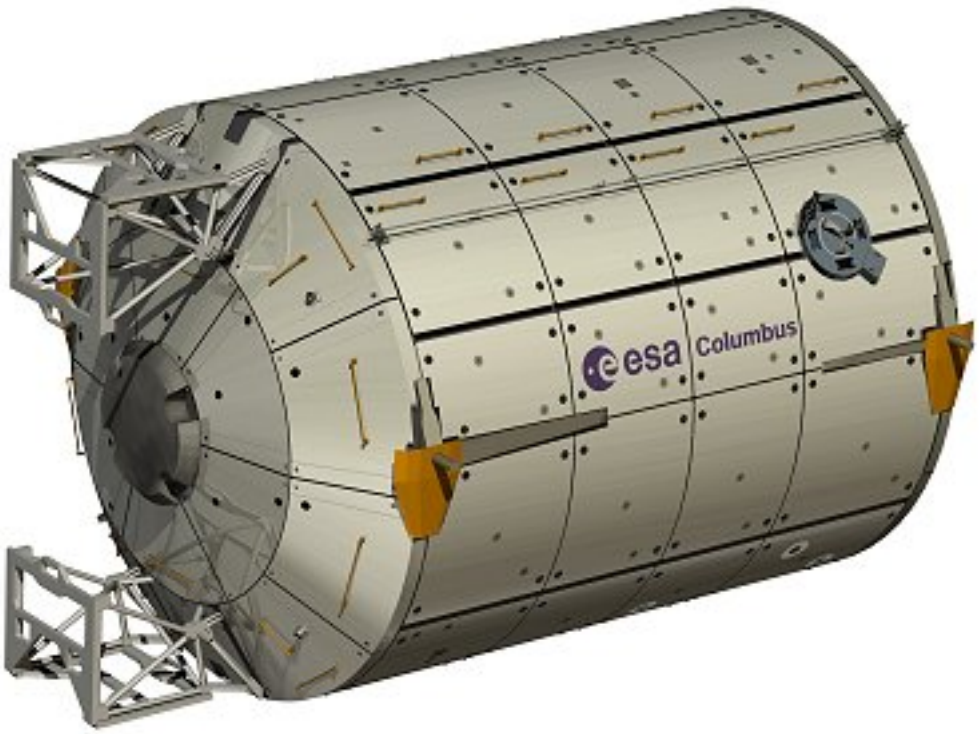
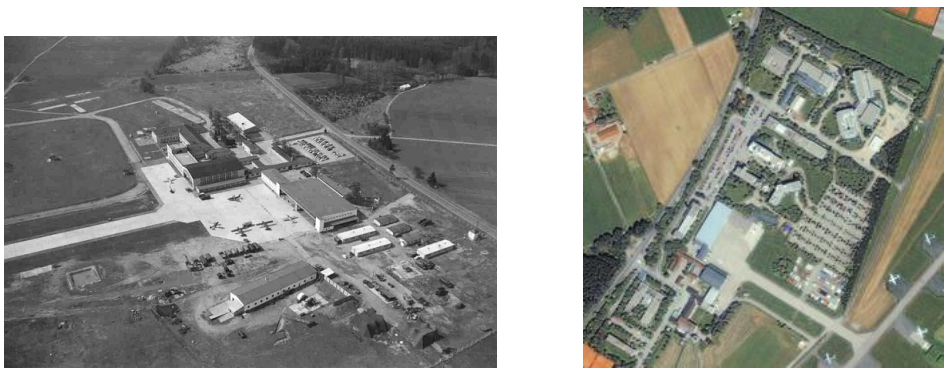


Abbildung 1.5.: Das Columbus-Modul [7]

2. Der Standort Oberpfaffenhofen[1]

Neben dem Standort Köln zählt Oberpfaffenhofen mit über 1.500 Mitarbeitern und acht Institutionen zu einem der größten Forschungszentren Deutschlands. Forschungsschwerpunkte sind unter anderem die Beteiligung an Weltraummissionen, die Klimaforschung, Forschung und Entwicklung zur Erdbeobachtung, der Ausbau von Navigationssystemen sowie die Weiterentwicklung der Robotik und Mechatronik. Der Standort, malerisch gelegen im bayrischen Voralpenland, profitiert zudem von einer starken Infrastruktur als auch von dem nebenan gelegenen Forschungsflughafen. Besondere Aufmerksamkeit wurde dem Standort zuletzt durch das europäische Weltraumlabor Columbus zuteil, welches vom neu gebauten DLR-Kontrollzentrum in Oberpfaffenhofen koordiniert wird. Man könnte somit vielleicht sagen, dass Oberpfaffenhofen das europäische Gegenstück zum amerikanischen Huston geworden ist.



(a) Die Anfänge (b) Der Standort heute

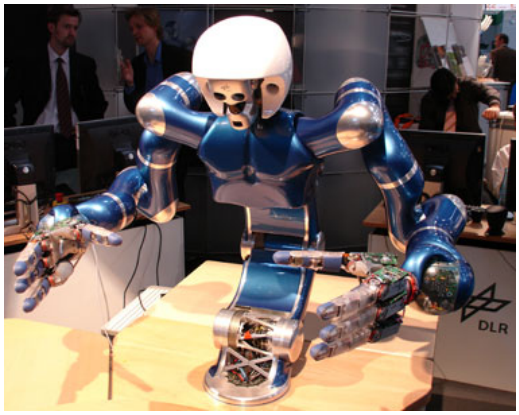
Abbildung 2.1.: Oberpfaffenhofen [8, 9]

[1] [1]

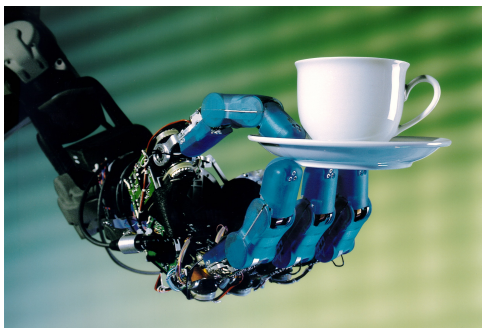
3. Das Institut für Robotik und Mechatronik

Das Institut für Robotik und Mechatronik ist eines von acht institutiellen Einrichtungen am DLR-Standort Oberpfaffenhofen (Abbildung 2.1 b unten links). Das von Prof. Gerd Hirzinger geleitete Institut zählt auf seinem Gebiet zu einem der führendsten in der Welt. Geforscht wird hier an zahlreichen Projekten über eine Spanne, die von der Fertigung der Werkstoffe bis hin zur Entwicklung künstlicher Intelligenzen reicht.

Eines der großen Schwerpunkte der Entwicklung liegt in der Konstruktion neuer Generationen von Leichtbaurobotern. Prestigeobjekt und gleichzeitiger Publikumsmagnet stellt der Roboter-Torso *Justin* dar (Abbildung 3.1), ein Leichtbauroboter der dritten Generation, an welchem die ebenfalls sehr bekannte vierfingrige DLR-Hand in zweifacher Ausführung angebracht wurde (siehe Kapitel 4 auf Seite 15).



(a) Der Roboter-Torso "JUSTIN", ein Leichtbauroboter der 3. Generation

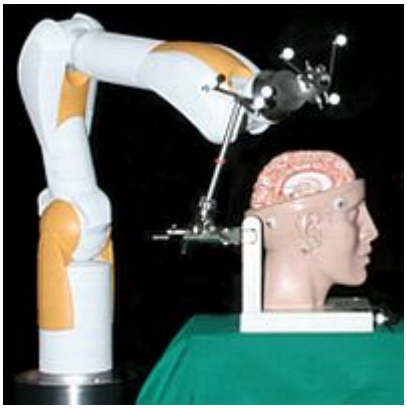


(b) Die DLR-Hand der 2. Generation

Abbildung 3.1.: Leichtbauroboter & DLR-Hand [10, 11]

Ein weiteres Augenmerk liegt auf der Entwicklung medizinischer Roboter (Abbildung 3.2 a), mit deren Hilfe in Zukunft einmal Operationen durchgeführt werden sollen, die heute undenkbar erscheinen und an Präzision alle heutigen Maßstäbe übertreffen. Eine preisgekrönte Entwicklung aus dem Bereich der Medizin stellt das DLR-Herz dar (Abbildung 3.2 b). 2003 wurde es bereits mit dem europäischen Innovationspreis für künstliche Organe ausgezeichnet.

Ein großer Durchbruch in der Robotik gelang dem Institut 1993 bei der D-2-Shuttle-Mission. Bei dieser Mission gelang es den Wissenschaftlern ihren Roboterarm ROTEX



(a) Medizinischer Roboter des DLR

(b) Das DLR-Herz

Abbildung 3.2.: Medizinische Forschungsgebiete [12, 13]

von der Erde aus so genau zu steuern, dass sie damit im schwerelosen Raum sogar frei schwebende Objekte einfangen konnten. Weitere Forschungsbereiche findet man in der Modellierung, Steuerung, Regelung von komplexen Systemen jeglicher Art, als auch das Erstellen virtueller Echtzeit-Umgebungen mit Kräfte- und Drehmomentenrückopplung für die Steuerung entfernter Robotersysteme (siehe ROTEX).

In der Arbeitsgruppe *Bionic*, welcher ich selber angehöre, werden u. a. biologische Systeme untersucht und mithilfe der so gewonnenen Erkenntnisse neue mechatronische Systeme entwickelt, welche sich immer mehr an biologische Gegebenheiten orientieren. Auch dynamische und intuitive Fähigkeiten werden untersucht nach natürlichem Vorbild in ein mechatronisches System integriert. Projekte sind z. B.:

- Das Fuß-Projekt
- künstliche Muskeln
- intuitives Greifen
- Das Violine-Projekt
- verschiedenste Sensorentwicklungen, u.v.m.

4. Die DLR-Leichtbauroboter der 3. Generation

4.1. Eine kleine Vorbetrachtung

Eine der großen Errungenschaften des Instituts für Robotik und Mechatronik sind die dort entwickelten Leichtbauroboter. Leichtbauroboter unterscheiden sich von herkömmlichen Industrierobotern vor allem durch ihr gutes Kräfte-Gewicht-Verhältnis. Die am Institut entwickelten Leichtbauroboter weisen ein Kräfte-Gewicht-Verhältnis von 1:1 auf. Anders ausgedrückt bedeutet das, dass sie ihr eigenes Gewicht heben könnten. Industrieroboter besitzen dagegen nur ein Verhältnis von 1:100 bis maximal 1:10. Das gute Kräfte-Gewicht-Verhältnis wird vor allem durch eine extrem leichte Bauweise mit leichten aber stabilen Materialien (z. B. Kohlestofffaser statt Stahl) sowie durch Motoren mit hoher Leistungsdichte erzielt. In den folgenden Zeilen dieses Abschnittes möchte ich nun einen kleinen Gedankenausflug ins Reich der Bewegungsabläufe von Maschine und Mensch unternehmen. Die Schlussfolgerungen dieses Textes sind rein spekulativ und entsprechen meinen persönlichen Vorstellungen.

Wie schon erwähnt, wird im Institut für Robotik und Mechatronik u. a. eine extreme Leichtbauweise der entwickelten Systeme bei gleichzeitiger Gewährleistung hoher Winkelgeschwindigkeiten angestrebt. Als Folge der verwendeten leichteren Materialien sowie der speziellen Antriebselemente büßt man jedoch auch einen gewissen Grad an Steifigkeit ein. Kohlestofffasern oder ähnliche Materialien besitzen zwar eine hohe Festigkeit, sind aber im Vergleich zu Stahl wesentlich flexibler, d. h. unter Belastung dehnbarer. Weitere Elastizitätseigenschaften lassen sich auf das "Harmonic drive-getriebe" zurückführen. Diese Tatsachen erscheinen im ersten Moment weniger problematisch. Um diese Problematik verstehen zu können, muss man sich die kinematischen Aspekte eines solchen Systems etwas näher anschauen:

Für die Planung eines Industrie-Roboters ist es von besonderer Bedeutung, dass er einen hohen Grad an Bewegungsgenauigkeit besitzt, d. h. möglichst kleine Abweichungen vom geplanten Bewegungsablauf aufweist. Ein bewährtes Verfahren zur Bestimmung der Genauigkeit ist die Analyse der Wiederholgenauigkeit. Man lässt den Roboter fortwährend ein und dieselbe Bewegung ausführen und ermittelt, wie sehr jede Bahnbewegung von der vorhergehenden abweicht. Eine große Wiederholgenauigkeit ist vor allem für präzise Fertigungsprozesse von hoher Wichtigkeit. Um diesen Grad an hoher Genauigkeit zu erhalten, setzen die Entwickler solcher Roboter auf eine sehr hohe Steifigkeit des gesamten System (Glieder, Gelenke, etc.). Eine solch hohe Steifigkeit ist vor allem für die Positionsbestimmung des TCP[1] von Bedeutung. Eine Positionsbestimmung des

[1] Der *TCP (Tool Center Point)* ist der Punkt am Manipulator, an dem ein Endeffektor (oder eine Hand)

TCP erfolgt folgendermaßen: In allen Gelenken des Roboters sind an den Wellen der Motoren Sensoren angebracht, mit welchen man die Winkel, entweder absolut durch eine Code-Scheibe oder relativ durch "Zählen" der Umdrehungen, jedes Gelenkes relativ zu einer Ausgangssituation bestimmen kann. Sind nun alle Winkel bekannt und ist darüber hinaus auch die komplette Geometrie des Roboters, so kann mithilfe der räumlichen Geometrie mathematisch die exakte Lage des TCP im Raum bestimmt werden. Diesen Vorgang nennt man in der Robotik "Vorwärts-Kinematik" (näheres dazu und zu der Umkehrung siehe Abschnitt 7.3). Jetzt wird das Problem langsam erkennbar: Man wird mit respektabler Genauigkeit behaupten können, dass die Geometrie eines sehr steifen Systems annähernd konstant bleiben wird, was eine Voraussetzung für die Berechnung mit der Vorwärtskinematik ist. Wie verhält sich nun aber die Geometrie eines wesentlich flexibleren Systems? Richtig, sie verändert sich. Abhängig von der Last biegen sich die Materialien, während die gemessenen Gelenkwinkel konstant bleiben. Die Vorwärtskinematik würde immer noch genau dieselbe Lage des TCP berechnen, obwohl sich diese aufgrund der Belastung verschoben hat.

Um dieses Problem zu umgehen gibt es verschiedene Ansätze: Eine besonders sinnvolle und auch natürliche Variante ist eine überlagerte Positionsregelung nicht über Gelenkwinkel, sondern über externe Sensoren. Die in den Kopf des Robotertorsos *Justin* (vgl. Abschnitt 4.3) eingebaute Stereokamera bietet eine hervorragende Möglichkeit für eine externe absolute Positionsbestimmung des TCP, ähnlich, wie es ein Mensch auch zu pflegen tut. Trotz der hoch entwickelten Stereokamera erweisen sich auch bei der externen Positionsbestimmung Abweichungen von bis zu einigen Millimetern als unzulässig hoch. Eine weitere extern überlagerte Regelung, das "Tasten", erweist sich ebenfalls als eine unabdingbare Fähigkeit. Obwohl in den Fingern der DLR-Hand FT-Sensoren integriert sind, wird der Roboter (noch) nicht über diese Sensoren Tasteindrücke wahrnehmen. Vielmehr findet eine Drehmomentenregelung Verwendung, welche mithilfe von Drehmonentsensoren den sich in den Gelenken aufbauenden Druck misst und entspechend verarbeitet. Somit wird der Roboter, auch ohne externe Drucksensoren, quasi "gefühlvoll" und kann auf seine Umwelt reagieren. Die ist besonders wichtig für eine sichere Interaktion zwischen Mensch und Maschine.

Eine weitere Möglichkeit zur Positionsbestimmung bietet eine vorherige Modellierung der Massenverteilungen im System, abhängig von der zu tragenden Last und der Gelenkwinkelpositionen. Es ist leicht nachzuvollziehen, dass bei einem gestreckten Arm die Hebelwirkung ungünstiger ist, als bei einem angezogenen, und sich somit die Materialien aufgrund der Eigenmasse und der evtl. zu tragenden Last stärker biegen. Durch eine vorherige Bestimmung der Massenverteilung des Systems aufgrund der Lage des Armes lässt sich diese Abweichung relativ genau vorherberechnen und es können entsprechende Gegenmaßnahmen in Form einer Vorsteuerung programmiert werden. Letztendlich wird also der auftretende Fehler "vorhergesehen" und die Gelenkwinkel daraufhin entsprechend angesteuert, um die gewünschte Lage wieder herstellen zu können. Um dieses Vorgehen zu bewerkstelligen, ist allerdings eine sehr genaue Kenntnis über die Systemdynamik von Nöten. Des Weiteren erscheint diese Vorgehensweise als sehr unnatürlich im Bezug auf ein

zur Bewältigung verschiedenster Aufgaben angebracht wird.

biologisches Wesen. Bei einer etwas genaueren Betrachtung wird aber erkennbar, dass das so nicht ganz zutreffend ist. Nehmen wir ein Beispiel: Ein Mensch möchte durch eine Tür gehen. Um durch die Tür zu kommen, muss er sie vorher öffnen. Nehmen wir einmal an, dass er eine massive Stahltür vor sich sieht, welche durch Drücken zu öffnen ist. Er wird nun also auf diese Tür zugehen und noch bevor er diese berühren kann, laufen in seinem Kopf bereits Berechnungen über die zu erwartende Masse und der somit benötigten Kraft ab, um diese zu öffnen. Es wird quasi eine Vorsteuerung berechnet. Und, wie der Roboter, braucht auch der Mensch vorab eine ziemlich genaue Kenntnis über die Systemdynamik seines Körpers sowie der zu erwartenden Last. Diese wird ihm aber nicht einprogrammiert, er nimmt sie vielmehr aus seiner Erfahrung. Dass dies nicht von Geburt an so ist, können wir schon daran erkennen, dass wir schließlich auch erstmal laufen lernen mussten[2]. Es ist natürlich ein interessantes Phänomen, das die "menschliche Vorsteuerung" unseren Bewegungsablauf dominiert, und die externe Regelung nachfolgend kleinere Abweichungen, welche durch fehlerhafte Berechnungen z. B. aus fehlerhaften oder fehlenden Erfahrungen stammten, korrigiert. Das dies wirklich der Fall ist, lässt sich wieder sehr gut am Tür-Beispiel nachvollziehen: Der Mensch, der sich auf die Tür zubewegt, sieht nun also eine schwere Stahltür. Da diese voraussichtlich schwer zu öffnen sein wird, berechnet er (unbewusst) einen hohen Kraftaufwand voraus und stellt die Muskeln bereits darauf ein. Nehmen wir einmal an, es handele sich bei der Stahltür um eine Fehleinschätzung. In Wirklichkeit handelt es sich hierbei um eine ähnlich aussehende aber sehr viel leichtere hohle Aluminium-Tür. Was wird passieren? Der Mensch wird die Tür aufgrund der zu hoch angesetzten Kraft höchstwahrscheinlich gegen die Wand schleudern, noch ehe eine nachträgliche Regelung der Kraft aktiv werden konnte. Im umgekehrten Fall würde das heißen, er berechnet für eine schwere Tür zu wenig Kraft und schafft es in Folge dessen nicht diese zu öffnen. Nun wird die Regelung aktiv und erhöht den Druck auf die Tür, bis diese sich schließlich öffnet. Die Vorsteuerung durch unsere Erfahrung und angelernter Bewegungsalgorithmen spielt in unseren Bewegungen also eine dominierende Rolle.

Somit ist eine Steuerung mithilfe modellierter Massenverteilung ein gar nicht mal so unnatürlicher Ansatz, wie man zunächst glauben mag. Der große Unterschied besteht natürlich darin, dass diese Vorsteuerung dem Roboter aufgrund von Messdaten einprogrammiert wird, während ein Mensch sie erlernt. Letztendlich stellt dies auch u. a. ein großen Ziel für die Entwickler solcher Systeme dar.

4.2. Der Ballfang-Roboter

Nach dieser kleinen Vorbetrachtung möchte ich nun einige der am Institut entwickelten Leichtbauroboter kurz vorstellen:

Der Ballfangroboter besteht aus einem auf einer Plattform montierten Arm, in der Regel mit der aufgesetzten DLR-Hand am TCP. Dieser Arm kann mithilfe von zwei Kameras ihm zugeworfene Bälle sicher fangen und halten. Für diese Leistung wurde das Institut weltweit berühmt.

[2] Es gibt allerdings auch Untersuchen die zeigen, dass der Mensch bereits von Geburt an ein fertiges "Laufprogramm" besitzt.

Abbildung 4.1.: Der Ballfang-Roboter vom Typ LBRIII [14]

Bei der Entwicklung des Leichtbauroboter-Arm-Hand-Systems konnten die Ingenieure bereits auf die Entwicklung des Leichtbauroboters der ersten und zweiten Generation sowie auf die DLR-Hand I zurückgreifen. Die extrem leuchte Bauweise wurde hauptsächlich durch die Verwendung von Carbon-Verbundwerkstoffen in einzelnen Armsegmenten erzielt, sowie durch Motoren extremer Leistungsdichte, sehr kleine Haltebremsen, u.v.m. Das entwickelte Gesamtsystem weißt somit folgende Daten auf:

- Masse: ~ 14kg
- Traglast: ~ 14kg
- Freiheitsgrade des Armes: 7 (äquivalent zu dem des Menschen)
- Freiheitsgrade der Hand: 13 (menschl. Hand: ca. 21)
- max. Winkelgeschwindigkeit: Im Basisgelenk des Armes 120°/s, In der Hand > 500°/s

Dieser Aufbau lässt somit nahezu menschlich aussehende Bewegungen zu, bei beachtlichen dynamischen Fähigkeiten, nicht zuletzt bedingt durch die geringe Massenträgheit des Gesamtsystems. Auf eine hohe Steifigkeit und einer damit einhergehenden hohen Positioniergenauigkeit musste dabei allerdings verzichtet werden, was eine große Herausforderung für die Programmierer und Regelungstechniker darstellt. Dafür wurde dem

System durch den Einbau von Drehmomentensensoren an jedem regelbaren Gelenk eine gewisse “Feinfühligkeit” verliehen. Zusätzliche Sensoren an Fingerspitzen erlauben des Weiteren eine direkte Wahrnehmung der Umgebung. [15, 16, 17]

4.3. Der Roboter-Torso *JUSTIN*

Quasi eine Folgeentwicklung des Hand-Arm-Systems stellt der Robotertorso *Justin* dar (siehe Abbildung 4.2), angeblich so benannt, weil er *just in* dem Moment fertig geworden ist, als er für die Automatica-Messe gebraucht wurde. Arme und Hände wurden vom Ballfangroboter integriert. Hinzu kamen ein Torso und ein Kopf, welcher multisensorielle Fähigkeiten aufweist. Erst vor kurzen wurde “ihm” dann auch noch ein fahr- und schwenkbarer Untersatz angebaut[3].

Ziel war es, ein möglichst menschähnlichen Oberkörper zu konstruieren, mit dessen Hilfe neue Programmier- und Regelungs-Konzepte für die beidhändige Manipulation erforscht und erprobt werden können. Folgende Daten weist der Torso auf:

- Freiheitsgrade Torso: 3
- Freiheitsgrade Kopf: 2
- Sensoraustattung des Kopfes: Stereokamera, Laserscanner und Lichtschnittprojektor

Somit besitzt das gesamte System 43 regelbare Gelenke was höchste Ansprüche an die Steuerungs- und Regelungstechnik stellt. Um trotz der äußert hohen Komplexität des Gesamtsystems verschiedene Ansätze testen zu können, wurde am Institut eine neue Softwarearchtitektur, das aRD-Konzept ("agile Robot Development") entwickelt. Mit diesem Konzept ist es möglich, klassische Steuerungen in mehreren über ein Netzwerk verbundene Module aufzutrennen, wobei jedes Modul unter harten Echtzeitbedingungen auf verschieden Prozessoren integriert werden kann.

Nicht zuletzt durch solche Entwicklungen ist *Justin* bereits jetzt dazu in der Lage, mit drei Bällen gleichzeitig zu hantieren oder einen *Krümel-Tee* zuzubereiten. Trotz alledem erweist sich Manipulationsfähigkeit des Systems noch als recht bescheiden, was die überaus hohe Komplexität, solcher für den Menschen scheinbar einfachsten Aufgaben, untermauert. Neue Konzepte der Steuerungs- und Regelungstechnik sind hier somit fast an der Tagesordnung. [16, 18]

[3] Der fahrbare Untersatz wurde vor ein paar Wochen im Eiltempo angefertigt, um rechtzeitig auf der Automatica 2008 präsentiert werden zu können. Das Ziel wurde erreicht, auch wenn nach eigener Beobachtung noch nicht alles reibungslos funktonierte. (Stand: Juli 2008)

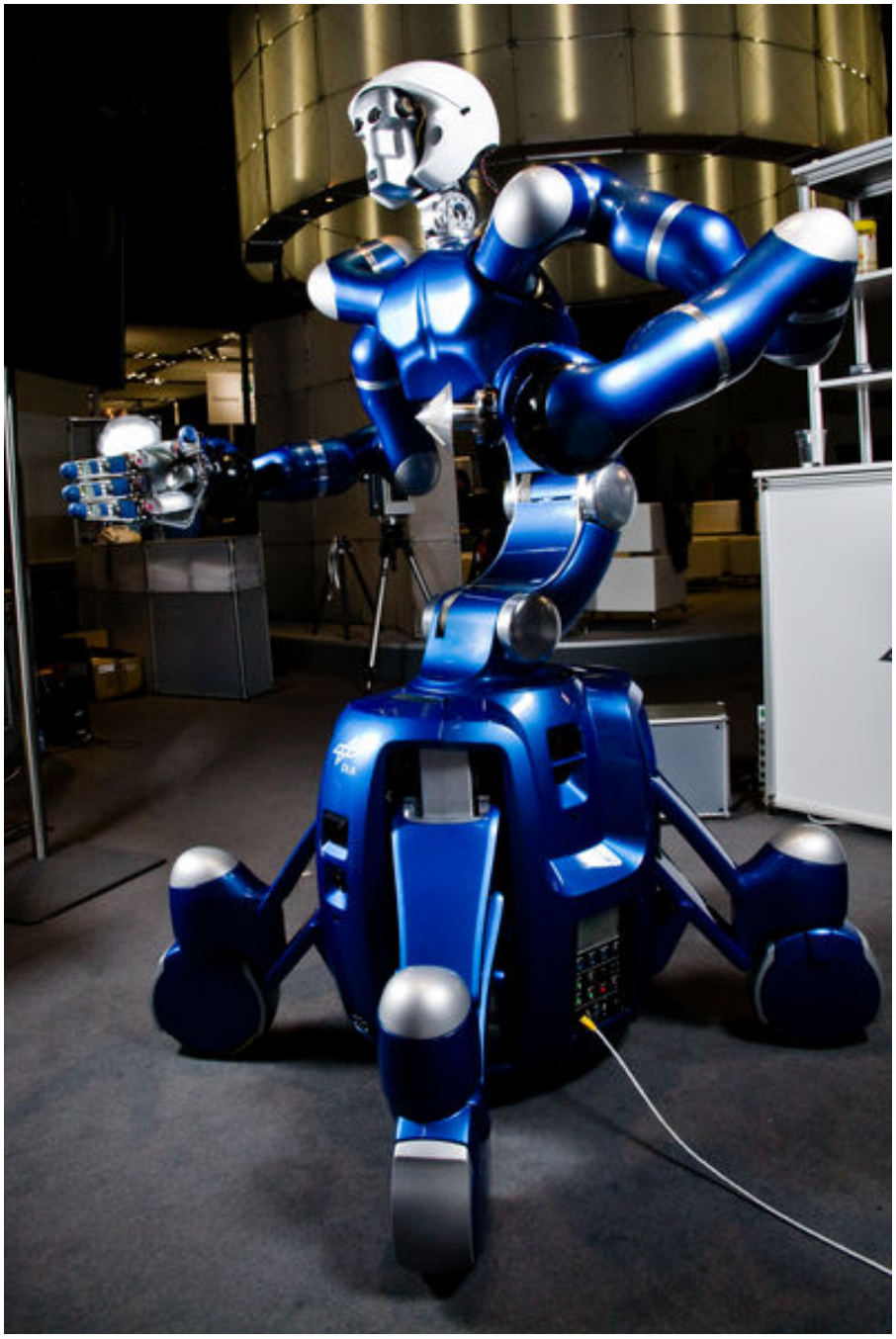


Abbildung 4.2.: Der Roboter *Justin*

Teil II.

Grundlagen der Robotik

Dieser Teil des Praktikumsberichtes soll einen kleinen grundlegenden Einblick in das überaus große und vielfältige Gebiet der Roboter-Technologien geben, mit Schwerpunkt auf die Steuerung. Es sei jedoch angemerkt, dass es nicht Ziel dieses Abschnittes ist, einen tieferen Einblick in die einzelnen wissenschaftlichen Gebiete zu geben. Das würde den Rahmen dieses Berichtes wohl gänzlich sprengen. Wer näheres über einzelne Gebiete erfahren möchte, den verweise ich hiermit auf entsprechende Literaturangaben.

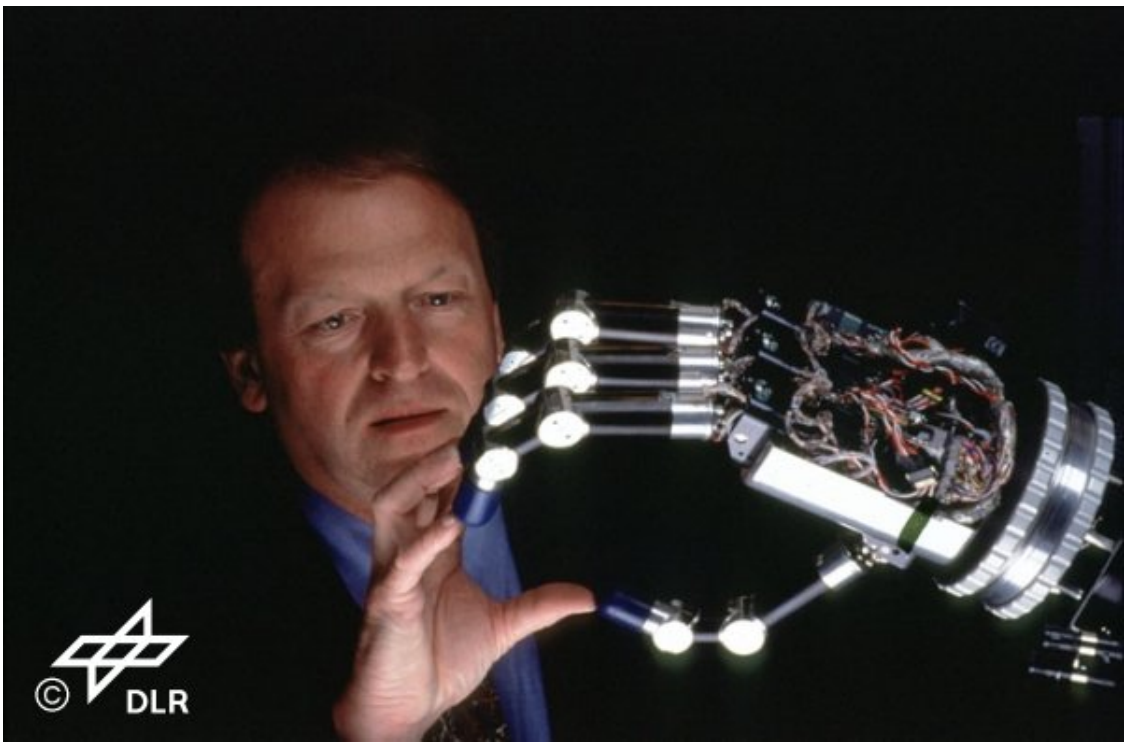


Abbildung 4.3.: Prof.Dr. Gerhard Hirzinger an der DLR-Hand I [19]

5. Entwicklung und Definition

Erste Ideen für den Bau von Maschinen, die selbstständig und automatisch Tätigkeiten ausführen sollten, sind schon sehr alt. Bereits in der Antike beschäftigten sich u. a. die alten Griechen auf diesem Gebiet. In der sogenannten *Schule von Alexandria* forschten und lehrten hochrangige Naturwissenschaftler und Philosophen, unter ihnen Heron, Pythagoras, Euklid und Archimedes. Diese *alexandrinischen Erfinder* waren allesamt Meister in der Kombination einfachster Mechanismen zu komplexeren Maschinen. Für die Schaffung komplexer Bewegungsabläufe dienten ihnen lediglich mechanische Komponenten wie Schrauben, Keile und Hebel sowie Wasser, Vakuum oder Luftdruck als Antriebskraft. Einer der bekanntesten Erfinder von ihnen war sicherlich Heron von Alexandria. In seinem Werk *Automata* erklärt er u. a. die Funktionsweise von Tempeltüren, welche sich beim entzünden eines Feuers wie von Geisterhand öffneten, Musikmaschinen, Münzautomaten und vieles mehr.

Natürlich kann man solche Gebilde wie sie die alten Griechen oder andere Kulturen entwickelt haben noch längst nicht als Roboter bezeichnen. Und doch zeigen sie erste Schritte auf dem Weg dahin auf. In der griechischen Mythologie findet man bereits viele solcher Gebilde, die einem "mechatronischen" System durchaus ähneln, z. B. künstliche Vögel, sprechende Statuen oder künstliche Diener. Nach dem Untergang des römischen

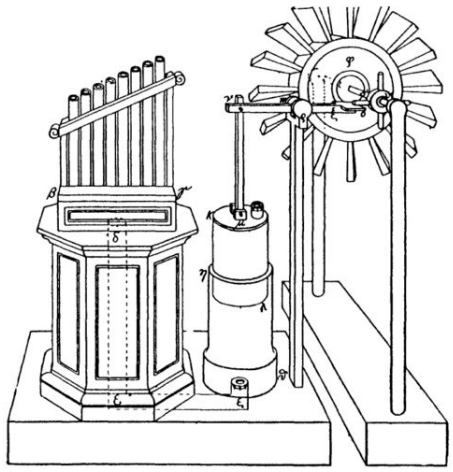


Abbildung 5.1.: Herons windangetriebene Orgel (Rekonstruktion) [20]

Imperiums entwickelte sich diese Art von Technologie überwiegend im arabischen Raum

weiter, da in Europa die meisten Schriften aus der Antike entweder verloren gingen oder nicht von Interesse waren. Dies änderte sich nahezu schlagartig mit dem Zeitalter der Renaissance. In dieser Zeit wurden scheinbar verschollene Schriften und Zeichnungen, teilweise auch von Heron, wieder aufgegriffen und im großen Stil nachgebaut. Die dabei entwickelten Regelungskonzepte erwiesen sich bereits als erstaunlich hoch entwickelt und wurden mehrmals unabhängig voneinander immer wieder aufs Neue wiedererfunden. Im 18. Jahrhundert erreichte der Automatenbau mit den Konstruktionen von *Jasques de Vaucanson* einen vorläufigen Höhepunkt. Nach seinem Studium der Anatomie konstruierte er unter anderem einen menschengroßen Flötenspieler, der sich gleichzeitig auf einem Tambourin[1] begleitete. Obwohl dieser bereits Lippen, Mund und Zunge besaß, bestand sein eigentlicher Antrieb wie üblich auch aus Uhrwerken und Blasebälgen. Weitere Visionen von ihm beinhalteten den Bau künstlicher Enten, die sich watschelnd fortbewegen aber auch fressen und verdauen sollten. (siehe Abbildung 5.2). Eine weitere sehr berühmte

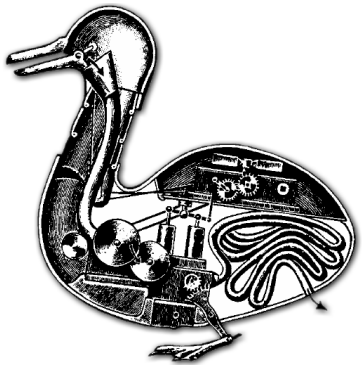


Abbildung 5.2.: Die mechanische Ente (1738) [20]

Konstruktion ist der *Schreiber*, erbaut von Pierre Jaquet-Droz und seinem Sohn Henri-Louis Jaquet-Droz um 1770. Diese etwa 70cm große Konstruktion war in der Lage, mit einer Gänsefeder in der Hand jeden beliebigen Text mit einer Länge von 40 Buchstaben zu schreiben. Kodiert wurde der Text auf einem Rad, von wo aus die Buchstaben einer nach dem anderen ausgelesen wurden. Somit kann man diese Erfindung schon als kleinen Vorläufer des Computers sehen, da sie über einen Speicher verfügte und programmiert werden konnte.

Nunmehr, seit über eintausend Jahren, dienten solche Entwicklungen mehr dem amüsanten Zeitvertreib, als das sie nutzbringend waren. Erst langsam entstanden daraus auch handwerklich oder industriell nutzbare Maschinen. Ab wann genau man nun eine solche Konstruktion als Roboter bezeichnen kann, ist sehr schwierig, allein schon deswegen, weil es keine weltweit einheitliche Festlegung dafür gibt. Der Ursprung des Wortes *Roboter* liegt im slawischen Wort *Robota*, welches man mit Arbeit, Fronarbeit oder Zwangsarbeit übersetzen kann. 1921 bekam das Wort durch Karel Capeks satirisches Stück *Rossum's Universal Robot,* in dem künstliche menschenähnliche Kreaturen Dienstleistungen vollrichten, seine heutige Bedeutung, wenngleich man seine Gebilde von damals heute eher

[1] Ein Tambourin ist ein Schlaginstrument, auch bekannt als Handpauke oder Schellentrommel

als Androiden bezeichnen würde. Die heute in Deutschland gängige Definition eines Roboters lautet folgendermaßen:

> *Ein Roboter ist ein frei und wieder programmierbarer, multifunktionaler Manipulator mit mindestens drei unabhängigen Achsen, um Materialien, Teile, Werkzeuge oder spezielle Geräte auf programmierten, variablen Bahnen zu bewegen zur Erfüllung der verschiedensten Aufgaben.*

Nach der Entwicklung einiger manuell gesteuerter Teleoperatoren und servoelektrisch betriebener NC-Maschinen wurde der erste eigentliche Roboter, ein einfaches "Pick-And-Place-Gerät, 1960 von Unimate entwickelt, mit welchem man manuell angefahrene Bahnkoordinaten abspeichern und erneut anfahren konnte (Teach-In-Verfahren). Dieses Verfahren wird auch heute noch sehr oft angewandt, wenn gleich die Bahnsteuerung durch Programmierung immer mehr Einzug findet. [28, 27, 21, 22]

6. Aufbau

6.1. Einteilung

Die große Bandbreite der Handhabungsgeräte, unter denen auch Roboter fallen, unterteilt man in die zwei großen Klassen: *manuell gesteuert* und *programmgesteuert*.

Bei den manuell gesteuerten Geräten handelt es sich um Teleoperatoren und Manipulatoren. Sie werden vom Bediener direkt ferngesteuert und vergrößern somit Kraft, Präzision, Schnelligkeit und Arbeitsbereich des Menschen. Zum Einsatz kommen diese ersten Robotertypen vor allem in Bereichen, die für den Menschen sehr oder gänzlig unzulänglich oder zu gefährlich bzw. kraftaufwändig sind, wie in Kernreaktoren, Meeresuntersuchungen oder dem Bergbau. Bei den programmgesteuerten Geräten wird nochmal in *festprogrammierten* und *freiprogrammierbaren* Robotern unterschieden. Wie schon der Name sagt, lässt sich die Programmierung eines festprogrammierten Gerätes nicht mehr ändern. Durch ihre geringen Anschaffungskosten und der modularen Aufbauweise finden diese Typen immer noch in vielen Bereichen Verwendung, wie z. B. beim Fräsen und Pressen. Freiprogrammierbare Roboter sind im Prinzip die universellsten Geräte, da, abhängig von ihrem Aufbau, jede denkbare Bewegung und jede Bewegungsabfolge möglich ist. Auch hierbei unterscheidet man mehrere Verfahren, welche im Kapitel 7 näher erläutert werden. [27]

6.2. Roboterkinematiken

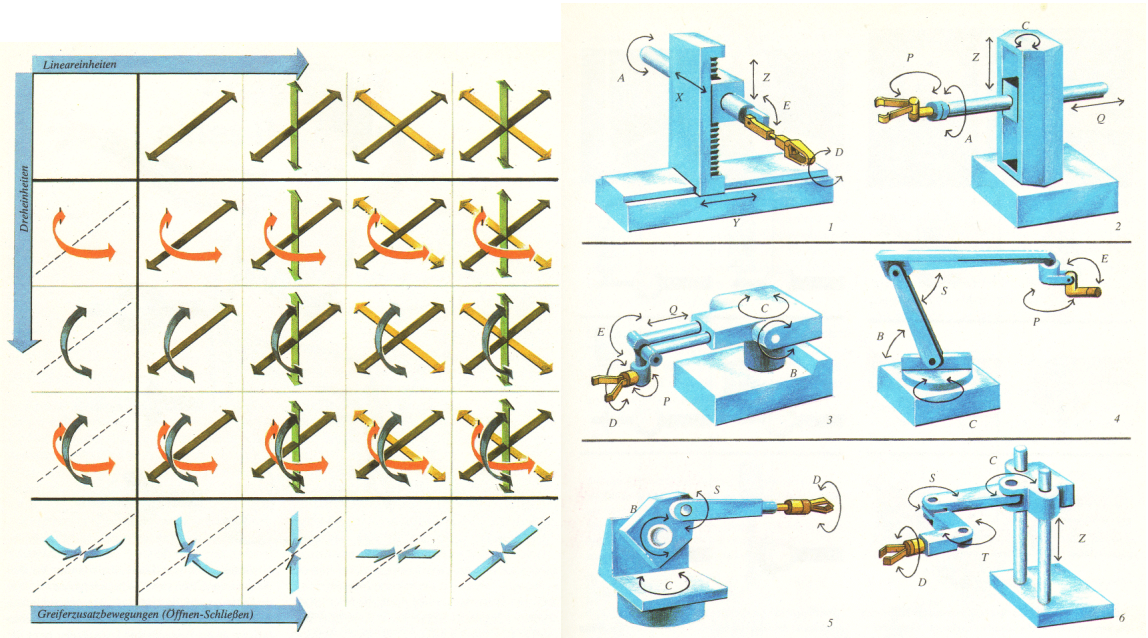
6.2.1. Freiheitsgrade

Roboter sind mehrgliedrige Mechanismen, die laut Definition mindestens drei Freiheitsgrade aufweisen, während die maximale Anzahl der Freiheitsgrade nach oben offen ist. Freiheitsgrade werden in den Gliedern der Maschine durch Schub- oder Drehgelenke realisiert. Der menschliche Arm z. B. besitzt sieben Freiheitsgrade, alle durch Drehgelenke realisiert: Jeweils drei im Schulter- und Handgelenk (Kugelgelenke) und noch einen zusätzlichen im Ellenbogengelenk (Scharniergelenk). Die meisten üblichen Industrieroboter besitzen hingegen sechs Freiheitsgrade, oftmals in einer Kombination aus Dreh- und Schubgelenken. Sie fragen sich jetzt vielleicht warum ausgerecht sechs? Nun, dies ist die minimale Zahl an Freiheitsgraden, die ein solches System aufweisen muss, damit der TCP jede Position und jede Orientierung an diesem im durch die Geometrie bestimmten Arbeitsraum erreichen bzw. einnehmen kann. Wie man sich leicht vorstellen kann, benötigt man mindestens drei Freiheitsgrade um in einem belieben Raum jeden Punkt anfahren zu können, da eine Bewegung in allen drei Dimensionen des Raumes von Nöten ist. Nun ist es aber so, das am TCP meist noch ein Werkzeug oder etwas ähnliches

starr angebracht ist. Damit dieses Werkzeug seine Arbeit vollrichten kann, bedarf es aber nicht nur der richtigen Position im Raum, sondern auch der richtigen Orientierung. Das Werkzeug muss also in die richtige Richtung geneigt bzw. gedreht werden. Da es sich hierbei wieder in allen drei Dimensionen bewegen können muss, sind drei zusätzliche Freiheitsgrade von Nöten, wobei diese nicht zwangsläufig direkt oder in der Nähe des TCP realisiert sein müssen. Wichtig ist nur, dass das Gesamtsystem sechs Freiheitsgrade aufweist. Der menschliche Arm besitzt also einen "überflüssigen" Freiheitsgrad, wobei dadurch mit Sicherheit fließendere und unkompliziertere Bewegungen möglich sind.

6.2.2. Arbeitsraum

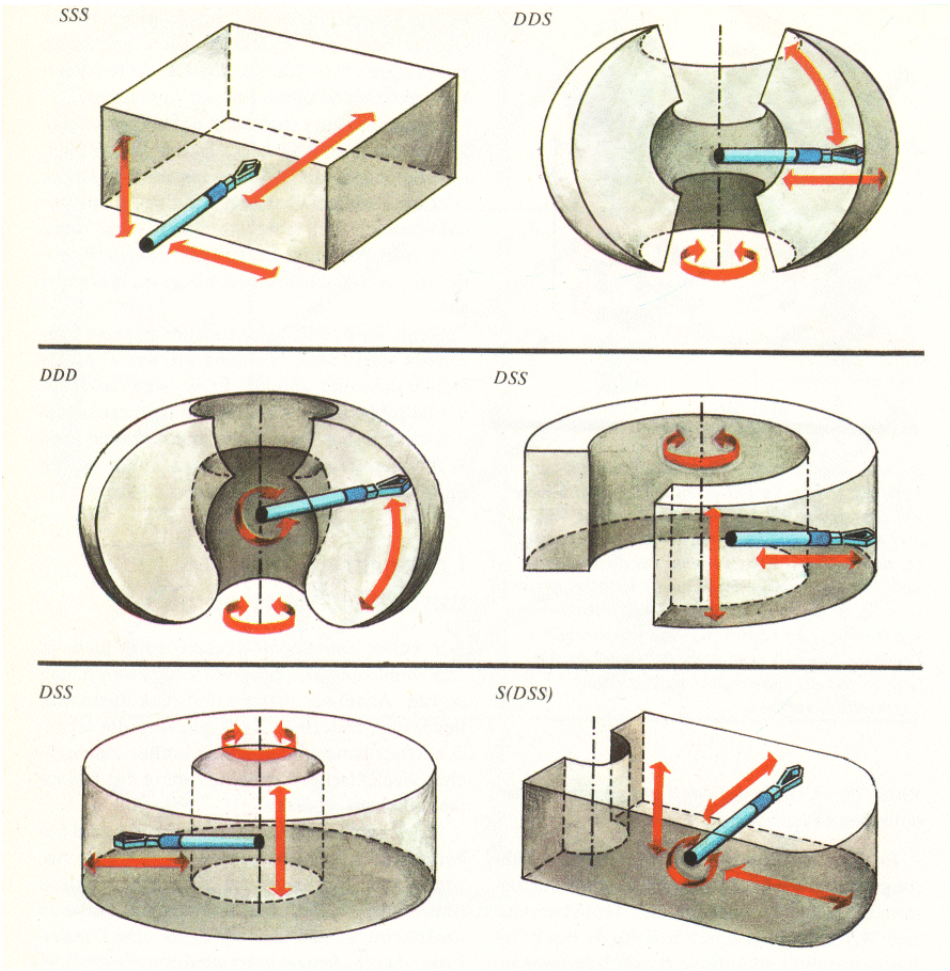
Während die Beweglichkeit durch die Anzahl der Freiheitsgrade bestimmt wird, erschließt sich der Arbeitsraum des Roboters hauptsächlich aus der Größe und Ausrichtung der Glieder. Wenn man z. B. ein System aus sechs Freiheitsgraden durch Drehgelenken besitzt, welche sich alle um dieselbe Axe drehen, so erhält man lediglich einen zweidimensionalen Arbeitsraum. Genau dasselbe Phänomen tritt auf, wenn sechs Drehgelenke alle auf einen Punkt fixiert sind, selbst wenn eine Drehung um jede Axe möglich ist. In diesem Fall erhält man eine Sphäre als Arbeitsraum. Obwohl für manche Anwendungsgebiete ein solcher Arbeitsraum ausreichend bzw. vorteilhaft sein kann, ist man im Allgemeinen gewillt, einen möglichst großen Arbeitsraum in allen drei Dimensionen zu schaffen. Die richtige Anordnung der Glieder in Position und Orientierung zueinander muss also gut bedacht sein.



(a) Kombinationsmöglichkeiten aus rotatorischer und translatorischer Bewegung

(b) Typische Kinematik von Industrierobotern

Abbildung 6.1.: Kinematik [23]



(a) Legende: S = Schubgelenk, D = Drehgelenk

Abbildung 6.2.: Arbeitsräume [23]

7. Steuerung

7.1. Vorwort

Eine große Herausforderung und ein wichtiger Forschungsschwerpunkt stellt die Steuerung solch komplexer Systeme dar. Bei hochentwickelten Maschinen wird mittlerweile mindestens genauso viel Zeit in die Programmierung, Steuerung und Regelung investiert, wie in die eigentliche Fertigung. Dies ist nicht sonderlich überraschend, hängen doch die Fähigkeiten eines Systems ganz entscheidend von diesen Faktoren ab. Weitläufiges Ziel ist es, wahrhaft intelligente Maschinen im Bezug auf deren Umwelt zu schaffen, welche beispielsweise selbständig Hindernisse erkennen und umgehen (oder vielleicht sogar aus dem Weg räumen) können, genauso wie eine eigenständige Bahnplanung "intuitiv" wählen zu können. Auch eine größtmögliche Interaktion mit der Umwelt, was vor allem das Sehen und Fühlen betrifft, ist, nicht nur zum Zwecke der Bahnplanung, von besonders großem Interesse. Diese Maschinen sollen also praktisch "motorische Intelligenz" besitzen, und, was einen weiteren großen Schritt darstellen würde, eventuell auch neue Bewegungsformen hinzulernen und sich altbewährte "merken". Dies hätte letztlich zur Folge, dass eine bereits länger im Betrieb stehende bau- und programmiergleiche Maschine mehr Fähigkeiten aufweisen würde, als eine entsprechend ältere. Dieser enorm große Schritt würde quasi der menschlichen Erfahrung gleichkommen, von der ich wie schon im Abschnitt 4.1 beschrieben glaube, dass sie ein bedeutenden Faktor in der Planung unserer Bewegungsabläufe darstellt.

Trotz enormer Forschungen auf diesem Gebiet sind solche Fähigkeiten bis jetzt noch Zukunftsmusik. Würde man den Grad der Intelligenz über die motorischen Fähigkeiten hinaus noch etwas weiter spinnen, so gelänge man sehr schnell in das Gebiet der Grenzwissenschaften. Dies würde das reine Gebiet der Robotik hier bei weitem überschreiten. Aber vielleicht macht genau das ja dieses Gebiet gerade auch so interessant. Es existiert in einer solchen Wissenschaft kein abgeschlossener Bereich. Vielmehr sind die Grenzen dieser Wissenschaft in fast jeglicher Richtung offen und erweiterbar.

Nichts desto trotz, sollen die nächsten Abschnitte lediglich einen kleinen Überblick über die heute üblichen und bewährten Steuerungsverfahren geben. Auch hierbei werde ich mich nur auf die grundlegendsten Techniken und Vorgänge beschränken müssen, um den Rahmen nicht zu sprengen. Wer einen tieferen Einblick in Materie wünscht, den verweise ich hiermit erneut auf entsprechende Fachliteratur (siehe Quellenangaben).

7.2. Mathematische Beschreibungsformen

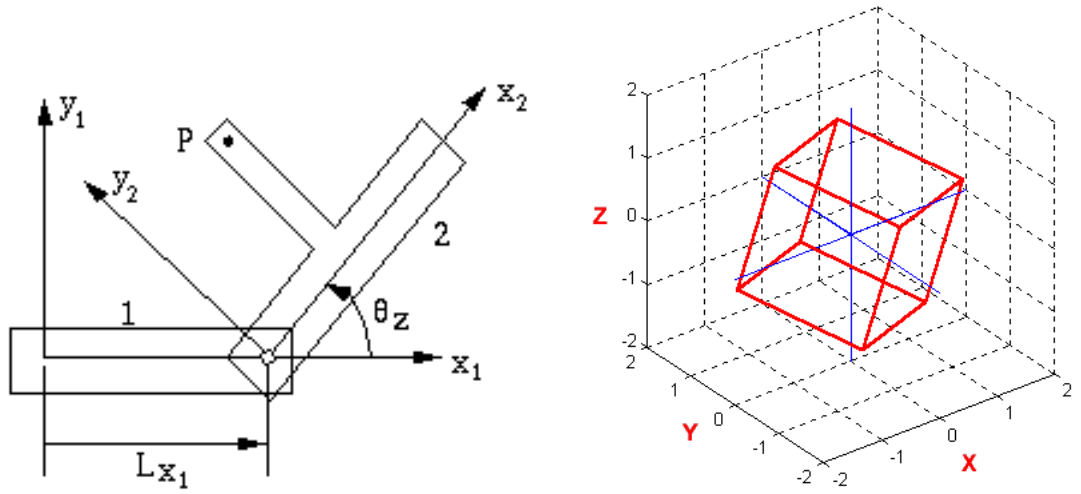
Um einen Manipulator bzw. Roboter zu steuern, existieren viele unterschiedliche Methoden. Die sicherlich einfachste Methode wird bei den Teleoperatoren, welche gewissermaßen die erste Robotergeneration darstellen, angewandt, wobei die einzelnen Gelenke der Maschine manuell direkt oder über einen zwischengeschalteten Rechner angesteuert werden. Eine vorherige Bewegungsplanung ist hierbei nicht nötig. Eine andere Methode stellt die so genannte *Punkt zu Punkt-* Steuerung dar, bei welcher eine vorherige Programmierung lediglich festlegt, von welchen zu welchen Punkt gefahren werden soll. Sind die Punkte fix durch das System vorgegeben (z. B. durch Endlagenschalter), so ist eine abstrakte Bahnplanung auch hier nicht erforderlich, da die anzufahrenden Punkte konstruktionsbedingt fest und unabänderbar sind. Bei einem freibeweglichen Manipulator, der lediglich durch die Grenzen seines Arbeitsraumes eingeschränkt wird, sieht das etwas anders aus. Will man diesen nicht als Teleoperator fernbedienen, sondern programmtechnisch steuern, so erweist sich eine entsprechende Ansteuerung der einzelnen Achsen als äußerst schwierig für das Erreichen einer gewünschten Position bzw. das Entlangfahren einer gewünschten Bahn. Das Problem liegt darin, dass die zugehörigen Gelenkwinkel für eine bestimmte Position einfach nicht bekannt sind und darüber hinaus sich eine Planung des Bewegungsablaufes über die Gelenkwinkel als äußerst unanschaulich und abstrakt erweist. Eine sehr viel günstigere und vor allem anschaulichere Methode erreicht man, indem man den realen Bewegungsablauf mathematisch in einer Art Kurvenzug darstellt. Der große Vorteil besteht darin, dass man sich den tatsächlichen Bewegungsablauf sehr einfach grafisch darstellen lassen kann und man sich darüber hinaus bei der Planung nur Gedanken um die eigentliche Bewegung machen muss, ohne an irgendwelche Gelenkwinkel denken zu müssen. Im Prinzip kann man sagen, man geht die Sachen von hinten an: Es wird direkt das Ergebnis beschrieben und nicht jene Sachen, die zu diesem führen. Allerdings sind auch bei Anwendung dieser Methode die Winkel der einzelnen Gelenke für jede Position nach wie vor unbekannt. Wie wird der Roboter also angesteuert? Diese Aufgabe erledigt in den meisten Fällen eine sogenannte *Inverse-Kinematik*, welche aus einer Position die dazugehörigen Gelenkwinkel berechnet (mehr dazu siehe Abschnitt 7.3).

7.2.1. Koordinatensysteme

Benutzt man also die Methode der Bahnplanung als ergebnisorientierte Steuerung, so ist es zwingend nötig, geeignete Koordinatensysteme zu definieren, in denen die mathematischen Berechnungen integriert werden können. Als erstes ist es nötig, ein universelles Bezugssystem zu schaffen. In der Robotik hat sich das so genannte Basiskoordinatensystem etabliert, welches seinen Ursprung an dem Punkt besitzt, auf dem der Roboter fest aufmontiert wurde. Die Punkte, welche sich auf diesen Raum beziehen, liegen somit in Basis- oder auch Welt-Koordinaten vor. Im Prinzip könnte man nun alle Berechnungen in diesem Basiskoordinatensystem durchführen und hätte das Ergebnis gleich in geeigneter Form vorliegen. Das kann man natürlich tatsächlich machen, würde aber in vielen Fällen die Anschaulichkeit, die durch diese Methode der Bahnplanung ja eigentlich erhöht

werden sollte, wieder zunichte machen. Vielleicht kann dies ein Beispiel am besten verdeutlichen, und da wir uns gerade im Violine-Projekt befinden, nehmen wir auch gleich eines aus diesem Zusammenhang: Nehmen wir einmal an, wir hätten die Violine im Arbeitsraum des Roboters auf irgendeine Art und Weise fest montiert und wollen nun eine mathematische Funktion entwickeln, die eine Bewegung über den Saiten in einer Ebene entlang einer Saite beschreibt. Nimmt man ferner an, dass der Hals der Violine eine Ebene im Raum bildet auf der die Saiten liegen und die Normale dieser Ebene kollinear zur Z-Achse und die Saite parallel zur X-Achse verläuft (sprich die Violine liegt "gerade" im Raum und wurde nicht gedreht), so wird die Entwicklung einer solchen Funktion keine größeren Schwierigkeiten bereiten, als wenn man dies in einem normalen zweidimensionalen Koordinatensystem täte. Man müsste lediglich mithilfe eines Ortsvektors den Ursprung vom Koordinatenursprung des Basiskoordinatensystems auf den Beginn der mathematischen Funktion verschieben. Nun ist es in der Realität aber meistens nicht so, dass die Violine so schön "gerade" im Bezugssystem liegt, sondern eher etwas schräg und gedreht, erst recht, wenn man bedenkt, dass diese ja auch einmal von der anderen Hand gehalten und eventuell geführt werden soll. Dieselbe Funktion auf der Saite, selbst wenn sie vorher einfacher Natur war, würde auf einer im Raum "schräg" liegenden Violine höchst komplexe Ausmaße annehmen, sofern sie immer noch das Basiskoordinatensystem als Bezugssystem besitzt. Ein wirrer Formelwust ohne Überschaubarkeit wäre die Folge. Tatsächlich lässt sich dieses Problem aber auf einfachste Weise vermeiden. Man "erschafft" sich einfach ein neues Koordinatensystem, indem die Violine wieder "gerade" und nicht gedreht liegt. Dieses neue Koordinatensystem kann man sich wie einen eigenen Raum vorstellen, der zum Basissystem gedreht und verschoben vorliegt (siehe Abbildung 7.1). Jetzt kann man wieder seine ursprünglichen mathematischen Funktionen verwenden, diesmal aber mit dem neuen Koordinatensystem als Bezugssystem. Nun besteht aber das Problem, das die im neuen System berechneten Werte zur weiteren Verarbeitung (z. B. durch die Inverse-Kinematik) noch unbrauchbar sind. Sie müssen also zwangsläufig umgewandelt werden, so, dass alle berechneten Werte wieder als Bezugssystem das Basissystem besitzen. Eine Möglichkeit, dieses Problem zu lösen, stellt eine sogenannte *Transformationsmatrix* dar, welche in der Lage ist, durch einfache Multiplikation die beiden verschiedenen Koordinatensysteme ineinander umzurechnen (siehe Abschnitt 7.2.2).

Fassen wir also zusammen: Zur Beschreibung von Punkten, Objekten und den meisten mathematischen Funktionen muss in jedem Fall ein Koordinatensystem als Bezugssystem dienen. Um komplexe Funktionen oder Beschreibungen zu vermeiden, ist es empfehlenswert, jedoch nicht zwingend nötig, eigene vom Basiskoordinatensystem verschiedene Koordinatensysteme zu definieren. Die relative Lage und Orientierung dieser neuen Systeme zum Basissystem werden durch Transformationsmatrizen eindeutig festgelegt und sind über diese Matrizen oder deren Inverse ineinander umrechenbar. Es ist auch möglich, Transformationsmatrizen zwischen zwei basisunabhängigen Koordinatensystemen zur schnelleren und einfacheren Umrechnung von Daten zu erstellen. Leztenendes müssen aber alle berechneten und bestimmten Daten in Weltkoordinaten vorliegen, da nur so eine Einheitlichkeit und eine Weiterverarbeitung z. B. durch die Inverse-Kinematik möglich ist.



(a) Koordinatenverschiebung im zweidimensionalen Raum

(b) Koordinatenverschiebung im dreidimensionalen Raum

Abbildung 7.1.: Koordinatensysteme [24, 25]

7.2.2. Die Transformationsmatrix

Wie bereits im Abschnitt 7.2.1 beschrieben, sind Transformationsmatrizen das universelle Werkzeug, um die einzelnen Koordinatensysteme ineinander umzurechnen. Auch der TCP lässt sich im Sinne der Vorwärtskinematik mithilfe einer solchen Matrix in Lage und Position bestimmen, wobei die Parameter für die Matrix in diesem Falle aus den Gelenkwinkeln des Roboters gewonnen werden. Dies ist natürlich von besonderer Bedeutung, da die relative Lage des TCP zu den im Arbeitsraum liegenden Objekten entscheidend für Steuerung und Programmierung ist.

In dem uns umgebenen dreidimensionalen Raum ist die Lage eines jeden Objektes eindeutig durch einen Ortsvektor, welcher aus drei Ortskoordinaten besteht, und der Orientierung durch drei Drehwinkel bestimmbar. Der Ortsvektor ist somit definiert durch

$$\overrightarrow{r} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{7.1}$$

Die drei Drehwinkel zur Bestimmung der Orientierung werden zu einer Drehungs- oder auch Orientierungsmatrix zusammengefasst:

$$\boldsymbol{R} = \begin{bmatrix} \overrightarrow{X} & \overrightarrow{Y} & \overrightarrow{Z} \end{bmatrix} \tag{7.2}$$

Die Vektoren X, Y und Z geben hierbei die Lage der Einheitsvektoren des neuen Koordinatensystems relativ zum vorhergehenden an. Spaltet man diese Vektoren in ihre drei

Vektorkoordinaten auf, so lässt sich die Drehungsmatrix als eine 3 x 3 Matrix schreiben:

$$\boldsymbol{R} = \begin{bmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{bmatrix} \tag{7.3}$$

$$\boldsymbol{R} = \boldsymbol{R_{ij}} \tag{7.4}$$

Der Index i gibt hierbei an, um welchen Einheitsvektor es sich handelt, während j beschreibt, in welcher Ebene sich, bezogen auf das vorherige Koordinatensystem, dieser Vektorpunkt verschiebt. Durch den Ortsvektor r und der Orientierungsmatrix R ist es nun möglich, einen beliebigen Punkt im Raum in Lage und Orientierung in ein anderes Koordinatensystem folgendermaßen zu transferieren (siehe auch Abbildung 7.2):

$${}^{B}P = {}^{A}_{B}\boldsymbol{R} \cdot {}^{A}P + {}^{A}_{B}\overrightarrow{r} \tag{7.5}$$

P kennzeichnen hierbei die Punkte im Raum, deren hochgestellter Index das Bezugssystem, in welchem der Punkt beschrieben ist. Im Falle der Rotationsmatrix R und des Ortsvektors r gibt der hochgestellte Index das Ausgangs- und der tiefgestellt das Zielkoordinatensystem an. Diese Angaben der jeweiligen Bezugskoordinatensysteme sind sehr empfehlenswert, da andernfalls durch die meist große Fülle an Transformationsmatrizen und Objektbeschreibungen keine Übersichtlichkeit mehr gewährleistet ist. Die endgültige

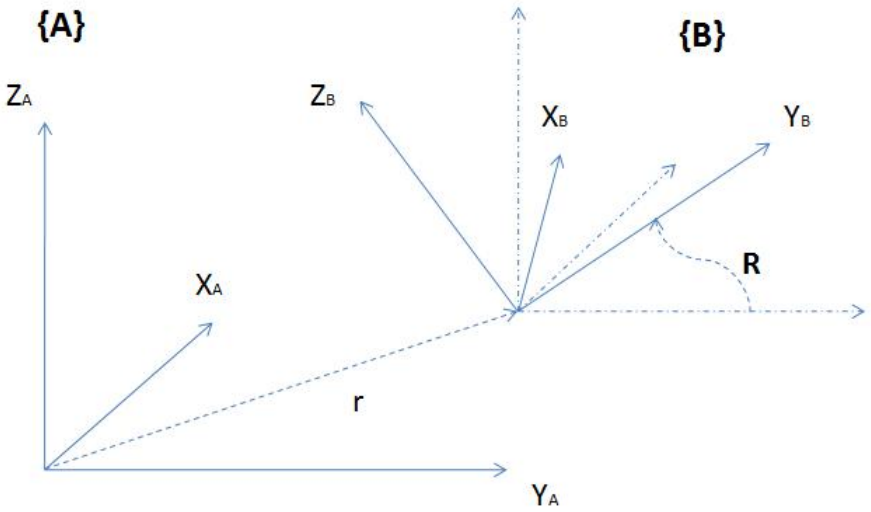


Abbildung 7.2.: Koordinatentransformation

Transformationsmatrix lässt sich nun aus dem Ortsvektor und der Orientierungsmatrix bestimmen. Dazu wird lediglich die Orientierungsmatrix mit dem Ortsvektor als Spaltenvektor erweitert. Als Ergebnis erhält man eine 3 x 4 Matrix, welche nun die Informationen

über relative Lage und Orientierung inne hält:

$$\boldsymbol{T} = \begin{bmatrix} r_{xx} & r_{yx} & r_{zx} & p_x \\ r_{xy} & r_{yy} & r_{zy} & p_y \\ r_{xz} & r_{yz} & r_{zz} & p_z \end{bmatrix} \tag{7.6}$$

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \overrightarrow{\boldsymbol{r}} \end{bmatrix} \tag{7.7}$$

Um mit dieser mathematischen Beschreibungsform rechnen zu können, wird die Matrix entsprechen 7.8 bzw. 7.9 noch um eine weitere Dimension ergänzt:

$$\boldsymbol{T} = \begin{bmatrix} r_{xx} & r_{yx} & r_{zx} & p_x \\ r_{xy} & r_{yy} & r_{zy} & p_y \\ r_{xz} & r_{yz} & r_{zz} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.8}$$

$$\boldsymbol{T} = \begin{bmatrix} & \boldsymbol{R} & & \overrightarrow{r} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.9}$$

In einem orthonormalen Koordinatensystem ist diese Zeile immer konstant und ergibt sich aus den Regeln der Matrizenrechnung. Da sie keine relevante Information enthält, wird sie bei Überlegungen und Schriften oft nicht mit angegeben. Für Berechnungen oder der Verarbeitung in einem Programm ist sie allerdings unerlässlich. Die Transformation eines Punktes würde somit folgendermaßen aussehen:

$${}^{B}P = {}^{A}_{B}\boldsymbol{T} \cdot {}^{A}P \tag{7.10}$$

oder im Umkehrschluss:

$${}^{A}P = {}^{A}_{B}\boldsymbol{T^{-1}} \cdot {}^{B}P \tag{7.11}$$

In diesem Falle wird zur Umkehrung der Transformation die Inverse der Transformationsmatrix mit dem Koordinatensystem multipliziert. [26, 28]

7.2.3. Interpretation

Aus Abschnitt 7.2.2 ist nun also der Aufbau einer solchen Transformationsmatrix bekannt. Zu erwähnen sei an dieser Stelle noch, dass dies nicht die einzige Methode darstellt, um Objekte oder Punkte im Raum vollständig zu beschreiben, aber doch die am häufigsten genutzte. Die hohe Anschaulichkeit einer solchen Matrix erweist sich in diesen Zusammenhang als großer Vorteil. Die Nutzung dieser Matrix geschieht auf denkbar einfache Weise:

Will man die Lage eines Punktes, eines Objektes oder den Ursprung eines neuen Koordinatensystem relativ zu einem Bezugsystem beschreiben, so sind dafür lediglich die einzelnen räumlichen Koordinaten in jeder Ebene *X, Y* und *Z* anzugeben. Dies entspräche dem Ortsvektor $\overrightarrow{r}$ und dieser wiederum der vierten Spalte der Transformationsmatrix,

wobei der letzte Wert wie schon beschrieben konstant mit eins anzugeben ist:

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.12}$$

Zu beachten sei hierbei, dass sich die angegebenen "Verschiebungswerte" X, Y und Z immer in den Ebenen des zugrunde liegenden Bezugssystems bewegen. Natürlich braucht eine Transformationsmatrix kein festes Bezugssystem. Es ist viel mehr eine relative "Änderungsangabe", dessen Bezug immer wieder aufs Neue frei gewählt werden kann. Ähnlich verhält es sich auch mit der Orientierungsmatrix, welche der ersten drei Spalten der Transformationsmatrix entspricht.[1] Die in 7.12 angegebene Form bewirkt keine Änderung der Orientierung im Raum.

Will man die Orientierung im Raum ändern, so ist es sinnvoll, die einzelnen Drehungen in verschiedenen Matrizen nacheinander um jeweils eine Achse anzugeben. Eine nachfolgende Multiplikation aller Matrizen erzeugt dann eine einheitliche Transformationsmatrix, welche in der Lage ist, die Orientierung entsprechend der Rotation um jede Achse beliebig zu beeinflussen. Mal angenommen, im Folgenden werde eine Rotation des Raumes um die X-Achse benötigt. Stellt man sich diese Drehung einmal gedanklich vor, so wird einem auffallen, dass weder eine Verschiebung des Einheitsvektors der X-Achse, noch eine Verschiebung eines beliebigen Einheitsvektors in Richtung der X-Achse von Nöten ist. Im Folgeschluss bedeutet dies, dass alle Glieder der Matrix, die den Einheitsvektor der X-Achse verschieben würden, nicht angetastet werden, genauso wie alle Glieder, die einen beliebigen Punkt in X-Richtung verschieben würden, ebenfalls unangetastet bleiben. Bezogen auf die Matrix bedeutet dies, dass die Glieder der ersten Zeile sowie die der ersten Spalte der Matrix alle mit Null angegeben werden müssen. Lediglich der erste Wert der Matrix, welcher die Verschiebung des X-Vektors in X-Richtung angibt, verbleibt mit der Einheitslänge eins, da andern falls der Raum in dieser Dimension gar nicht erst aufgespannt werden würde. Übrig bleibt also nur eine Verschiebung der Einheitsvektoren Y und Z, in Richtung der Y und Z-Achsen. Besonders einfach erweist sich hierbei die Angabe eines Drehungswinkels α. Aus den trigonometrischen Gesetzen erhält man somit folgende Form der Transformationsmatrix:

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & \cos\alpha & -\sin\alpha & Y \\ 0 & \sin\alpha & \cos\alpha & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.13}$$

Analog dazu erfolgt eine Drehung um die Y-Achse

$$\boldsymbol{T} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & X \\ 0 & 1 & 0 & Y \\ -\sin\alpha & 0 & \cos\alpha & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.14}$$

[1] Die letzte Zeile ist eine konstante und gehört somit nicht zur eigentlichen Orientierungsmatrix.

bzw. der Z-Achse

$$\boldsymbol{T} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & X \\ \sin\alpha & \cos\alpha & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.15}$$

wobei α jeweils den Drehungswinkel um die entsprechende Achse angibt. [26, 28]

7.3. Inverse-Kinematik

7.3.1. Vorwort

Die Inverse-Transformation stellt ein überaus komplexes und mathematisches Gebiet da, welches ich hier nur kurz anschneiden möchte. Ziel einer Inversen- oder auch Rückwärts-Transformation ist es, Basiskoordinaten, welche im Basiskoordinatensystem vorliegen, in geeignete Gelenkwinkel "zurückzurechnen", so dass die geforderte Orientierung und Lage des Endeffektors erreicht wird.

Diese Transformation ist nur selten explizit möglich und ihre Komplexität hängt entscheidend von Größe und Aufbau der offenen kinematischen Kette des Systems ab. Wie man sich leicht vorstellen kann, ergeben sich für die meisten Positionen im Raum mehrere Möglichkeiten der Gelenkwinkelstellungen, für einige sogar unendlich viele. Letzterer Fall tritt sehr oft auf, wenn zum Erreichen einer Position mehr Freiheitsgrade vorhanden sind als eigentlich von Nöten (redundante Freiheitsgrade). Diese "überflüssigen" Freiheitsgrade sind dann frei bestimmbar und nicht abhängig von der jeweiligen Position des Endeffektors. Eine explizite Lösung der Gleichungen ist meist nur bei sehr einfachen Aufbauten mit wenigen Freiheitsgraden möglich. Es ist aber ebenfalls möglich, einige Freiheitsgrade in ihrem Bewegungsspielraum einzuschränken (z. B. Stellung des Ellenbogens) und somit die Bewegungsgleichungen zu vereinfachen.

7.3.2. Analytische Verfahren

Es existieren in der Literatur eine ganze Reihe analytischer Verfahren über inverse Transformationen. Als eines der bekanntesten soll hier das Verfahren von *Paul (1981)* als Beispiel dienen.

Für die Entwicklung einer Inversen-Kinematik werden hierbei systematisch kinematische Beziehungen erstellt und in geeigneter Weise ausgewählt. Als Grundlage hierzu dient die *Denavit-Hartenberg*-Methode, welche u. a. die Lage zweier benachbarter Glieder des Manipulators relativ zueinander angibt, wobei von einer besonderen Anordnung und Orientierung benachbarter Koordinatensysteme ausgegangen wird, welche die einzelnen starren Glieder des Roboters beschreiben. Alle Teilkörper werden beginnend von Null in aufsteigender Reihenfolge durchnummeriert, wobei Null die Basis darstellt. Folgende Bedingungen müssen für zwei benachbarte Koordinatensysteme gelten (D-H-Konvention):

- die z_{i-1}-Achse liegt entlang der Bewegungsachse des i-ten Gelenks
- die x_i-Achse steht normal auf der z_{i-1}-Achse und zeigt von ihr weg

- die Anordnung der y_i-Achse ergibt ein rechtshändiges System

Die auf diese Weise angeordneten Koordinatensysteme lassen sich nun durch eine Rotation θ_i (Gelenkwinkel) um die z_{i-1}-Achse

$$\boldsymbol{Rot(z_{i-1}, \theta_i)} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.16}$$

eine Translation d_i (Gelenkabstand) entlang der z_{i-1}-Achse

$$\boldsymbol{Trans(z_{i-1}, d_i)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.17}$$

eine Translation a_i (Armelementlänge) entlang der x_i-Achse

$$\boldsymbol{Trans(x_i, a_i)} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.18}$$

sowie eine Rotation α_i (Verwindung) um die x_i-Achse

$$\boldsymbol{Rot(x_i, \alpha_i)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.19}$$

ineinander überführen. Durch Multiplikation der Matrizen erhält man die endgültige Matrix *D,* auch als *Denavit-Hartenberg-Matrix (D-H-Matrix)* bekannt:

$$D_{i-1,i} = Rot(z_{i-1}, \theta_i) \cdot Trans(z_{i-1}, d_i) \cdot Trans(x_i, a_i) \cdot Rot(x_i, \alpha_i) \tag{7.20}$$

$$\boldsymbol{D_{i-1,i}} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & l_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.21}$$

Diese spezielle Transformationsmatrix stellt also den mathematischen Zusammenhang zwischen zwei aufeinander folgenden Koordinatensystemen dar, wobei die Anzahl abhängig von den Freiheitsgraden f des Systems ist. Durch eine Multiplikation aller D-H-Matrizen erhält man die endgültige Transformationsmatrix, welche die relative Lage des Endeffektors zur Basis beschreibt:

$$\boldsymbol{T^0_{TCP}} = D_{0,1} \cdot D_{1,2} \ldots \cdot D_{f-1,f} \tag{7.22}$$

Diese Matrix entspricht der Vorwärtstransformation oder Vorwärtskinematik. Um nun daraus eine Inverse-Transformation zu erhalten, bedarf es einer Umkehrung der Vorgänge, wobei aus einer Position die entsprechende Anzahl an Gelenkwinkeln bzw. translatorischen Bewegungen (z. B. bei Schubgelenken) ermittelt werden muss. Ausgangspunkt ist dabei die Matrix T^0_{TCP}. Durch eine Linksmultiplikation mit allen Inversen der D-H-Matrizen in aufsteigender Reihenfolge erhält man folgende Gleichungssysteme:

$$\begin{array}{rrrrrl}
 & & & & \boldsymbol{T} & = D_{0,f} \\
 & & & D_{0,1}^{-1} & \boldsymbol{T} & = D_{1,f} \\
 & & D_{1,2}^{-1} & D_{0,1}^{-1} & \boldsymbol{T} & = D_{2,f} \\
 & D_{2,3}^{-1} & D_{1,2}^{-1} & D_{0,1}^{-1} & \boldsymbol{T} & = D_{3,f} \\
D_{3,4}^{-1} & D_{2,3}^{-1} & D_{1,2}^{-1} & D_{0,1}^{-1} & \boldsymbol{T} & = D_{4,f} \\
 & & & \ldots\ldots & & \\
D_{f-2,f-1}^{-1} & \ldots\ldots & & D_{0,1}^{-1} & \boldsymbol{T} & = D_{f-1,f}
\end{array} \tag{7.23}$$

Jede dieser Matrizengleichungen aus 7.23 führt auf zwölf nichttriviale einzelne Gleichungen. Ziel ist es nun, durch eine Untersuchung jeder dieser Matrizengleichungen Elemtengleichungen zu entwickeln, welche auf eine einzelne Gelenkvariablen zurückführen, bis für jede Gelenkvariable eine explizite Gleichung vorliegt. Da die Anzahl der insgesamt zu untersuchenden Gleichungen enorm mit der Anzahl der Freiheitsgrade ansteigt ($n = 12f$), geht mit dieser Methode ein hoher Rechenaufwand einher und das Verfahren wird für sehr komplizierte Systeme oft unbrauchbar. [28]

7.3.3. Numerische Verfahren

Da in vielen Fällen eine analytische Lösung nicht möglich oder zu umständlich ist, werden zunehmend auch numerische Verfahren zur Rücktransformation von Trajektorienpunkten angewandt. Solche Annäherungsverfahren profitieren vor allem von den schnellwachsenden Leistungen moderner Mikroprozessoren. Im folgenden werden zwei Verfahren kurz vorgestellt:

7.3.3.1. Milenkovic und Huang

Ein vor allem für seperatierbare Manipulatoren geeignetes teilnumerisches Verfahren, bei dem die Berechnungen in zwei Teilen vorgenommen wird: Die kinematische Kette des Systems wird hierbei in zwei Teilketten aufgespaltet, welche jeweils explizite Gleichungen auf Gelenkvariablen inne halten: die Positionsteilkette und die Orientierungsteilkette. Als erstes werden die Gelenkkoordinaten der Positionsteilkette so bestimmt, dass der Endeffektor die gewünschte Lage im Raum einnimmt. Die Orientierungskette wird dabei "festgehalten" und nicht verändert. Ist die korrekte Lage erreicht, wird die Positionskette festgehalten und nunmehr die Orientierungskette durch die Gelenkkoordinaten beeinflusst, bis die angestrebte Orientierung vorliegt. Da sich beide Teilketten gegenseitig beeinflussen, ist die Lösung auf iterativem Wege durch beliebig viele Wiederholungen dieser Vorgänge möglichst genau zu bestimmen. Grundvoraussetzung für dieses

Verfahren ist es, dass bei endlich vielen Wiederholungen die Werte aller Gelenkkoordinaten konvergieren. Die Anzahl der Wiederholungen ist dabei abhängig von der geforderten Genauigkeit sowie der zur Verfügung stehenden Rechenleistung.

7.3.3.2. Furusho und Onishi

Dieses Verfahren basiert auf dem modifizierten Newton-Verfahren, welches zur Verminderung der Rechenleistung angewandt wird. Trotz der wesentlich niedrigeren Konvergenzgeschwindigkeit beim modifizierten Verfahren erweisen sich die Näherungsergebnisse als ausreichend genau. Der große Vorteil dieses Verfahrens liegt darin, dass die Jacobimatrix (siehe Gleichung 7.25) für jeden Trajektorienpunkt nur einmal bestimmt werden muss.

Bei bekannten Gelenkwinkeln y eines Manipulators kann mithilfe der Vorwärtskinematik, dargestellt als Funktion f, direkt die Lage p des Endeffektors bestimmt werden:

$$p = f(y) \tag{7.24}$$

Wie bereits erwähnt, ist die Bestimmung der Gelenkwinkel aus der Endeffektorposition nicht immer explizit möglich. Allerdings lässt sich ein direkter Zusammenhang zwischen den Positions- und Gelenkwinkelgeschwindigkeiten herstellen, vorausgesetzt die dazugehörige Jacobimatrix J ist nicht singulär:

$$J(y) = \frac{df}{dy} \tag{7.25}$$

$$\frac{dy}{dt} = J^{-1}(y)\frac{dp}{dt} \tag{7.26}$$

Weiterhin gilt für den Vektor eines beliebigen Trajektorienpunktes *k+1*:

$$p(k+1) = f(y(k+1)) \tag{7.27}$$

Sind nun geeignete Näherungswerte für die Gelenkwinkel $\theta^i(k+1), i = 1, 2$ vorhanden, kann durch ein Iterationsverfahren über die Jacobimatrix der Gelenkwinkel $y(k+1)$ näherungsweise bestimmt werden. Dabei gelten folgende Zusammenhänge:

$$\begin{aligned} \theta^1(k+1) &= q(k) + J^{-1}(\theta^1(k))\Delta p(k) \\ \theta^2(k+1) &= \theta^1(k+1) + J^{-1}(\theta^1(k+1))\{p(k+1) - f(\theta^1(k+1))\} \\ y(k+1) &= \theta^2(k+1) \end{aligned} \tag{7.28}$$

Eine günstige geometrische Konfiguration kann hierbei die Transformation enorm erleichtern. [28]

7.4. Methoden der Steuerung

Wie bereits im Abschnitt 7.3 angesprochen, existieren eine Vielzahl von Methoden zur Generierung eines Bewegungsablaufes. Eine grobe Übersicht soll dieser Abschnitt geben.

7.4.1. Punkt-zu-Punkt-Bewegung

Ursprünglich wurden Robotersysteme ausschließlich für diese Art von Bewegungsablauf konzipiert. Auch heute noch wird dieses Konzept in vielen Bereichen angewandt. Kennzeichnend für diese Art von Steuerung ist ein sehr geringer Softwareaufwand sowie eine einfache Handhabung. Wie der Name schon vorgibt, wird hierbei eine Bewegung lediglich von einem Punkt zum anderen ausgeführt. Die Anzahl der vorzugebenden Punkte ist dabei beliebig variabel. Es existieren verschiedene Möglichkeiten zur Realisierung dieser Bewegungsmethode:

Sind die anzufahrenden Punkte vom System im Aufbau fest vorgegeben (z. B. durch Endlagenschalter), so ist für eine Programmierung lediglich eine Ablaufsteuerung von Nöten, welche Anzahl und Reihenfolge der Punkte vorgibt. Diese Steuerungsart ist allerdings eher bei Fertigungsstraßen oder ähnlichen üblich und bei Robotern selten. Sind die Punkte hingegen frei im Raum bestimmbar, so können sie entweder in Weltkoordinaten vorgegeben oder direkt durch Anfahren (Teach-In) gemessen werden. Letztere Methode entledigt sich durch die direkte Messung der Gelenkkoordinaten einer Koordinatentransformation, während bei ersterer Methode diese dementsprechend durchzuführen ist (vgl. Abschnitt 7.2.1). Sind die Punkte zeitinvariant, so ist eine Koordinatentransformation nur einmal nötig, was die Rechenzeit deutlich verringert. Sind die Punkte hingegen zeitlich variable, so ist eine Transformation mindestens nach jeder Lageänderung der anzufahrenden Position erneut durchzuführen.

Während die vom Endeffektor einzunehmenden Punkte mit beliebiger Genauigkeit vorzugeben sind, verhält sich die eigentliche Bewegung zwischen diesen Punkten völlig unkoordiniert. Sie ist im Wesentlichen abhängig vom Aufbau des Systems sowie vom regelungstechnischen Ansatz. Zu unterscheiden sei hier allerdings noch zwischen einem sequentiellen oder parallelen Bewegungsablauf. Bei einer sequenziellen Bewegung wird jeweils immer nur eine Achse angesteuert, während alle anderen verharren, wodurch meist große Umwege zum Erreichen einer Position in Kauf zu nehmen sind. Allerdings bedingt diese Methode eine teilweise Beeinflussung der Bewegungsbahn zwischen zwei Punkten. Parallele Bewegungen implizieren hingegen eine gleichzeitige Bewegung der benötigten Achsen zum Anfahren des nächsten Punktes. Die jeweiligen Achsen erreichen dabei ihre geforderte Endposition meist zu verschiedenen Zeitpunkten, abhängig von der Strecke und der Geschwindigkeit. Vorhersagen über den Bahnverlauf sind damit nur schwer möglich. [27]

7.4.1.1. Koordinierte Punkt-zu-Punkt-Bewegung

Abhilfe für dieses Problem schafft eine koordinierte Punkt-zu-Punkt-Bewegung. In vielen literarischen Werken wird diese schon als Vorstufe zur Bahnplanung angesehen. Es gibt verschiedene Ansätze zur Realisierung solcher Bewegungsabläufe. Eine sehr häufig verwendete ist die zeitoptimale Planung. Ziel ist es dabei, dass alle Achsen zeitgleich ihre Zielposition erreichen, wobei die Gesamtzeit auf ein Minimum zu beschränken ist. Diese Forderung kann umgesetzt werden, indem man für die entsprechenden Achsen die maximale Beschleunigung bzw. Verzögerung einschränkt oder die maximale Bewegungs-

geschwindigkeit herabsetzt. Eine Kombination aus beiden ist natürlich auch möglich. [27]

7.4.2. Bahnsteuerung

Die Bahnsteuerung in Umweltkoordinaten stellt eine moderne und immer häufigere Methode der Bewegungsplanung dar. Im Gegensatz zur Punk-zu-Punkt-Bewegung werden bei dieser Methode nicht nur die anzufahrenden Punkte, sondern der komplette Bewegungsverlauf in vielen eng aneinander liegenden Punkten vorgegeben. So gesehen entspricht eine Bahnsteuerung auch einer Steuerung von Punkt zu Punkt, wobei die Punkte hierbei nicht nur als Angabe für Anfangs- und Zielposition dienen, sondern als Vorgabe einer einheitlichen Bewegung. Des Weiteren werden diese Punkte meist aus mathematischen Berechnungen gewonnen (siehe Abschnitt 7.4.2.4). Da die Trajektorie bei einer Bahnsteuerung in Weltkoordinaten entwickelt wird, ist auch hierfür eine Koordinatentransformation von Nöten, wobei die Resultate dieser Transformationen häufig als Führungsgrößen einer unterlagerten Regelung verwendet werden. Abhängig von der Art der Koordinatentransformation von Welt- in Roboterkoordinaten unterscheidet man folgende Steuerungskonzepte: [27]

- Positionssteuerung (Resolved Motion Position Control - RMPC)
- Geschwindigkeitssteuerung (Resolved Motion Rate Control - RMRC)
- Beschleunigungssteuerung (Resolved Motion Acceleration Control - RMAC)
- Kraftsteuerung (Rosolved Motion Force Control - RMFC)

7.4.2.1. Positionssteuerung

Bei der Positionsteuerung werden durch eine direkte Umkehrung von 7.24 alle zugehörigen Roboterkoordinaten y berechnet. Dies geschieht, wie bereits in Abschnitt 7.3 erläutert, durch eine Inverse-Kinematik. Die aus dieser Koordinatenrücktransformation gewonnenen Gelenkwinkel werden im Folgenden oftmals als Führungsgröße für eine unterlagerte Regelung verwendet, welche demensprechend für das Erreichen dieser neuen Gelenkwinkelposition verantwortlich ist. Durch eine solche unterlagerte Regelung wird sicher gestellt, das kleine Abweichungen im Erreichen der neuen Gelenkposition, hervorgerufen durch Berechnungsfehler oder Modellungenauigkeiten, möglichst weitgehend ausgeglichen werden. Falls darüber hinaus noch andere Werte wie $\dot{y}$ und $\ddot{y}$ benötigt werden, müssen diese aus einer numerischer Differentiation von y gewonnen werden. Da numerische Differentiationen oftmals "aufrauend" wirken, sind diese Werte, bedingt durch eine Verstärkung des bereits vorhandenen Fehlers, nicht immer ausreichend gut. Auf eine numerische Differentiation sollte somit falls irgend möglich verzichtet werden. Mit einer Positionssteuerung lassen sich vor allem schnelle Bewegungen gut und gezielt ausführen, während eine "fühlende" Interaktion mit der Umwelt ohne eine hybride Kopplung, z. B. mit einer Kraftsteuerung, nicht möglich ist. [27]

7.4.2.2. Geschwindigkeitssteuerung

Bei der Geschwindigkeitssteurung wird das nichtlineare Problem der Inversen-Transformation auf ein lineares zurückgeführt, sofern Näherungswerte der gesuchten Gelenkwinkel bekannt sind, also die Koordinaten benachbart sind. Dabei wird aus $\dot{p}$ zunächst $\dot{y}$ und durch anschließender Integration y bzw. durch einmalige numerische Differentiation $\ddot{y}$ gewonnen. Dieses Verfahren wurde prinzipiell schon im Abschnitt 7.3.3.2 *(Furusho und Onishi)* erläutert und entspricht im Wesentlichen diesem. Allerdings bietet das Verfahren nach *Furusho und Onishi* nur für Manipulatoren bis zu sechs Freiheitsgraden eindeutige Lösungen. Eine Lösungsmethode bei Robotersystemen mit redundanten Freiheitsgraden stellt die Entwicklung einer verallgemeinerten Inverse wie z. B. der Pseudoinversen $J^\sharp$ dar, welche durch 7.29 genau die p liefert, die den kleinsten euklidischen Abstand zum vorhergehenden Punkt aufweisen. [27]

$$\triangle y = J^\sharp \triangle p \tag{7.29}$$

7.4.2.3. Beschleunigungs- und Kraftsteuerung

Mit $\ddot{p}$ (Beschleunigung in Weltkoordinaten) wird durch eine zweimalige Differentiation aus 7.24 $\ddot{y}$ gewonnen, woraus man durch Integration $\dot{y}$ und durch eine erneute Integration y erhält. Auf eine numerische Differentiation kann somit völlig verzichtet werden, was wie in 7.4.2.1 beschrieben von Vorteil ist. Nutzt man diese Größen als Führungsgrößen für die unterlagerte Regelung, spricht man von einer Beschleunigungssteuerung. Von einer Kraftsteuerung spricht man meist dann, wenn von einer statischen Betrachtungsweise ausgegangen wird, z. B. wenn der TCP mit einer definierten Kraft entlang eines Gegenstandes fährt. [27]

7.4.2.4. Bahnplanung

Die Berechnung der Bahn stellt den eigentlichen Vorgang dar, um die Bewegungen des Roboters zu beschreiben. Obwohl eine Bahnbestimmung auch in Roboterkoordinaten möglich ist, wird sie wie schon in 7.2.1 erläutert auf Grund der Übersichtlichkeit und der Eindeutigkeit im Bezug auf die Umwelt fast immer in Weltkoordinaten vorgenommen.

Grundlegend basiert eine solche Berechnung darauf, dass zuerst die notwendigen Punkte, welche anzufahren sind, bestimmt werden und anschließend zwischen diesen Interpoliert wird. Dabei wird Stetigkeit der Kurve und deren Ableitung, meist auch noch deren zweiter Ableitung gefordert, d. h., mit der Beschreibung des Kurvenzuges müssen Geschwindigkeit und Beschleunigungen durch Zeitparametrisierungen ebenfalls, genauso wie die eigentliche Lage im Raum, mit einbezogen werden. Es sind verschiedene Möglichkeiten denkbar, um eine solche Trajektorie zu erstellen (vgl. Tabelle 7.1):

Eine Möglichkeit besteht darin, den kompletten Kurvenzug mit allen für die Regelung benötigten Werte in einer ausreichenden Punktedichte direkt in Umweltkoordinaten zu berechnen und abzuspeichern. Ist der Bewegungsablauf fix und bedarf keiner Änderung, so genügt eine einmalige Koordinatentransformation eines jeden Punktes und eine anschließende Abspeicherung der Roboterkoordinaten, um diese Bewegung beliebig oft

auszuführen. Da eine solche Transformation dabei nur einmal auszuführen ist, bedarf es keiner hohen Rechenleistung, jedoch einer geeignet großen Kapazität an Speicherplatz. Eine andere Methode beinhaltet die Abspeicherung der Werte in Umweltkoordinaten und bei Ausführung der Bewegung eine Echtzeitkoordinatentransformation zu jeder Taktzeit des Systems. Durch diese Variation ist eine schnellere eventuell benötigte Abänderung der Trajektorie möglich. Allerdings sind hierbei hohe Rechenleistungen nötig, da zu jeder Taktfrequenz eine vollständige Koordinatentransformation für den jeweiligen nächsten Punkt abgeschlossen sein muss. Abhilfe für die Entstehung solch hoher Rechenleistungen schafft eine Kombination aus einer in Umweltkoordinaten erstellten Trajektorie und einer ergänzenden Berechnung in Winkelkoordinaten. Dabei wird eine Trajektorie mit nur soviel Punkten berechnet, wie zum Beschreiben der Kurve laut Genauigkeitsforderung unbedingt notwendig. Da die unterlagerte Regelung in den meisten Fällen aber wesentlich mehr Werte benötigt, wird darauffolgend nochmal zwischen diesen Punkten geeignet interpoliert, was in Roboterkoordinaten geschehen kann. Durch diese Methode sind wesentlich weniger Online-Koordinatentransformationen nötig, was die benötigte Rechenleistung beträchtlich reduzieren kann.

Großer Nachteil einer vorher in Umweltkoordinaten beschriebenen Trajektorie besteht darin, dass sie, erst einmal berechnet, nicht mehr an veränderte Umweltbedingungen angepasst werden kann. Verändern sich aber Lage und/oder Orientierung von Objekten im Raum während der Bearbeitung, so ist eine adaptive Trajektorienberechnung nötig. Eine Möglichkeit zur Umsetzung dieser Aufgabe bietet eine inkrementelle Erstellung der Trajektorie, d. h. die einzelnen Punkte werden in Abhängigkeit von den gerade vorherrschenden Bedingungen in Echtzeit berechnet. Dabei wird ein Punkt, gleich nachdem er berechnet wurde, in Roboterkoordinaten transformiert und sofort für die Regelung bereitgestellt. Dieses Vorgehen beansprucht mit Sicherheit die höchste Rechenkapazität von allen, wohingegen so gut wie kein Speicherplatz benötigt wird. Eine abgeschwächte Variante dieser Vorgehensweise kann man dadurch realisieren, indem man unmittelbar vor Beginn der Bewegung alle anzufahrenden Punkte bestimmt und darauffolgend eine Interpolation in Echtzeit zwischen diesen vornimmt. Damit wäre eine Teiladaptivität bewerkstelligt, die zwar variierende Anfangsbedingungen berücksichtigt, aber keine Veränderungen während der Laufzeit mehr zulässt. Schlussendlich ist auch noch eine Kombination aus den letzten beiden Varianten denkbar: Dabei spaltet man den Gesamtprozess in viele einzelne auf und verfährt für jeden Prozess wie in der zuletzt beschrieben Teiladaptivitätsmethode. Ein neuer Prozess ist hierbei gleichbedeutend mit einer Änderung der Lage eines Objektes im Raum. Da dadurch die jeweils neuen vorherrschenden Bedingungen in der Trajektorienberechnung berücksichtigt werden, liegt somit eine Steuerung mit quasikontinuierlicher Adaptivität vor. Allerdings ist diese Methode nur für zeitlich diskrete Änderung von Objektpositionen im Raum möglich, da sonst zu jeder Taktfrequenz ein neuer Prozess starten würde, was eine ständige Änderung der Anfangsbedingungen zur Folge hätte.

Zu erwähnen währe noch, dass bei einer adaptiven Steuerung, zu jedem der benötigten Zeitpunkte, die genaue Lage und Orientierung eines jedes Objektes messbar sein muss, was in der Robotik immer noch große Probleme bereitet (vgl. Kapitel 11). Während bei einer Teiladaptivität “nur” die jeweiligen neuen Anfangsbedingungen bestimmt wer-

den müssen, bedarf es bei einer kontinuierlichen Adaptivität einer ständigen Messung jeder Position, was in Bezug auf Genauigkeit und Rechenleistung mit enormen Aufwand verbunden ist. Abhilfe kann hier ein hybrider Steuerungsansatz, eine sogenannte Position/Kraft-Steuerung, schaffen, bei der auf eine genaue Bestimmung der Objektlage in Bezug auf Genauigkeit und/oder zeitliche Abtastrate verzichtet wird. Ausgeglichen werden diese Abweichungen durch eine Kraftvorgabe mithilfe taktiler Sensoren oder Drehmomentmessungen in den Gelenken. Anders ausgedrückt kann man sagen, der Endeffektor wird sich auf Grund von Unkenntnis über die genaue Lage eines Objektes an dieses herantasten. Realisierbar ist dieses Vorgehen aber nur, wenn die Abweichungen der Lagemessung nicht zu groß sind und ein direkter Kontakt mit der Umwelt besteht.

Anzuf. Punkte	Interpolation	Transf.	Rechenleistg.	Speicherkap.	Ges.-Adaptivität
fest	fest	einmalig	niedrig	hoch	sehr niedrig
fest	fest	echtzeit	hoch	hoch	niedrig
fest	echtzeit	echtzeit	sehr hoch	niedrig	niedrig
echtzeit/adaptiv	echtzeit	echtzeit	sehr hoch	sehr niedrig	mäßig
echtzeit/adaptiv	echtzeit/adaptiv	echtzeit	sehr hoch	sehr niedrig	hoch

Tabelle 7.1.: Verfahren zur Trajektorienerzeugung im Vergleich

Teil III.

Beschreibung des Praktikums

8. Das Violine-Projekt

Dieses Kapitel der Praktikumsarbeit soll im Folgenden einen grundlegenden Überblick über die bereits vorhandene Arbeit des *Violine-Projektes* vor Beginn meines Praktikums geben. Für eine genauere Beschreibung verweise ich auf die Praktikumsarbeit von Thomas Haase [29, 30].

8.1. Beschreibung der Trajektorie

Thomas Haase stellt mit seiner Praktikumsarbeit *Mathematische Analyse, Modellierung und Simulation des Violine spielens am LBR III* den Beginn des Projektes dar. Vorgabe war es dabei, eine Trajektorie auf der Grundlage einer Positionssteuerung zu entwickeln, da sich andere Verfahren wie die Kraft/Drehmomentensteuerung als zu langsam und schwingungsanfällig erwiesen haben. Es folgt eine kleine Zusammenfassung seiner Arbeit:

8.1.1. Bewegungsraumanalyse

In seinem ersten Kapitel *Bewegungsraumanalyse* beschreibt Haase die Objekte Bogen und Violine und definiert zugleich vier Basisunabhängige Koordinatensysteme sowie den TCP. Alle im System benutzten Koordinatensysteme sind in Abbildung 8.1 dargestellt. Für die Violine bzw. den Steg wurden zwei Koordinatensysteme erachtet, ein S und ein N-System, wobei das N-System lediglich zum S-System zur leichteren mathematischen Beschreibung 180° um die Z-Achse gedreht vorliegt. Dem Bogen hingegen wurde naturgemäß der TCP als Bezugssystem zugrunde gelegt und relativ zu diesem zwei Koordinatensysteme definiert: Das B (Bogen)-System und das H (Haar)-System. Das B-System ist im Bezug auf dem TCP ebenfalls nur um die Z-Achse gedreht, wobei der Drehwinkel experimentell zu ermitteln ist. Dies ist erforderlich, um die genaue Orientierung des Bogens relativ zum TCP zu erhalten. Das H-Koordinatensystem bedient sich einer festen Transformationsmatrix gegenüber dem B-System, welche sich aus den konstruktiven Gegebenheiten des Bogen ableitet (vgl. Abbildung 8.2). Eine genaue Vermessung des Bogens sowie der Violine ist somit von entscheidender Bedeutung für die Bestimmung der wesentlichen zur Berechnung notwendigen Punkte.

8.1.1.1. Probleme

Wie sich in einem neuerlichen praktischen Versuch zeigte, ist die genaue Vermessung des Bogens und der Violine äußerst schwierig und mitunter unzulässig hoch fehlerbehaftet. Da nach *Haases* Modell eine genaue Kenntnis über die geometrischen Gegebenheiten zur Bestimmung der Lage der Saiten sowie der Bogenhaare erforderlich ist, sollte es in

Betracht gezogen werden, alternative Bestimmungsmöglichkeiten für Lage und Orientierung zu nutzen. Eine aussichtsreiche Methode verspricht die Bestimmung der Saitenlage durch Anfahren des TPC an diesen und eine Messung der dazugehörigen Gelenkwinkel. Durch eine anschließende Vorwärtstransformation erhält man dementsprechend eine ziemlich genaue Lage der Saiten in Weltkoordinaten, welche zur weiteren Berechnung im Programm an geeigneter Stelle genutzt werden könnte. Weitere praktische Versuche werden dies zeigen (vgl. Kapitel 11).

8.1.2. Lage, Orientierung & Bogenführung

Aus den konstruktiven Gegebenheiten der Violine sind nun Lage sowie Orientierung der Saiten im Raum bestimmbar. Hierbei wurde Spiegelsymmetrie an der X-Z-Ebene angenommen. Diese Näherung dürfte ausreichende Genauigkeit liefern. Desweiteren wurde für jede Saite eine geeignete Orientierung für beliebige Spielpunkte berechnet, welche der Bogen beim Streichvorgang einzunehmen vermag. Dem Bogen selber wurden relativ zum TCP zwei Punkte, welche Anfangs- und Endpunkt für den Streichvorgang darstellen, berechnet. Somit sind aus Lage, Orientierung und Geometrie der Violine alle anzufahrenden Punkte berechenbar. Für eine vollständige Trajektorie ist es nun noch von Nöten zwischen diesen Punkten zu Interpolieren.

8.1.3. Interpolation

Für eine einheitliche Trajektorie wurden zwei unterschiedliche Interpolationsmethoden für jeweils verschiedene Aufgaben ersonnen: Für das Streichen einer Seite bedarf es einer linearen Interpolation, eventuell mit einer überlagerten Streichtrajektorie, welche orthogonal zur Saite und Streichrichtung angreift. Die Geschwindigkeit wird dabei durch die Punktedichte bestimmt. Für Anfahrt, Abfahrt und Saitenwechsel wurden einfache und gekoppelte Bézierfunktionen erstellt, welche möglichst natürliche fließende Bewegungen kreieren sollen. Dabei ist ein Kontakt oder gar Durchfahren der Saiten genauso zu vermeiden, wie Sprünge oder andere Unstetigkeiten der Kurve und möglichst auch deren erster Ableitung. Je nach Übergangsart wurden einfache oder gekoppelte Bézierfunktionen mit entsprechenden Anfangs- und Endpunkte gewählt. Die Geschwindigkeitsregelung ist hierbei ebenfalls durch die Punktedichte gegeben, welche in diesem Falle allerdings variable ist und durch eine gesonderte Funktion berechnet wird. Dies ist erforderlich, da die Funktion zur Berechnung der Bézierkurven Punkteabstände implementiert, die für eine fließende Bewegung unbrauchbar sind. Durch eine solche überlagerte Geschwindigkeitsregelung, welche durch eine Umstrukturierung der Punkteabstände gekennzeichnet ist, konnte u. a. erreicht werden, dass der Bogen bei "Anfahrt" von Null langsam und gleichmäßig bis zu einem Maximalwert beschleunigt und anschließend mit genau der Geschwindigkeit auf die entsprechende Saite trifft, mit der diese angespielt werden soll. Selbiges gilt für jegliche Überführungen sowie der Abfahrt. Letztlich können so alle berechneten Teiltrajektorien nahtlos aneinander gereiht und zu einer einheitlichen verbunden werden.

8.2. Das Programm

Ein Großteil der von *Haase* entwickelten mathematischen Berechnungsmethoden wurden von ihm gleichfalls in einem Programm umgesetzt. Mit diesem Programm war es bereits möglich vollständige Trajektorien zu erzeugen und in *Matlab* zu Simulieren, wobei sich allerdings Größen wie Geometrie, Lage und Punkteabstände noch auf abstrakte Werte bezogen.

8.2.1. Aufbau

Das vollständige Programm für die Erzeugung einer Trajektorie wurde in *Matlab* entworfen und in vielen Programmsegmenten untergliedert. Es fanden sich auch Versuche in *Simulink* wieder, welche aber unvollendet blieben und meines Wissens zum Teil auch wieder verworfen wurden. Die jeweiligen Programmsegmente entsprechen verschiedenen Aufgabenbereichen und stehen alle in Beziehung zueinander. Jedes dieser Segmente wurde in einem so genannten M-File geschrieben. Eine strukturelle Übersicht der M-Files ist im Ablaufplan nach Abbildung 8.3 erkennbar. Die Datei *Start.m* stellt hierbei die Startdatei des gesamten Programms dar, in welcher die Lage des Steges bezüglich der Umwelt anzugeben ist und alle anderen benötigten M-Files aufgerufen werden. Auch die abzuspielende Matrix ist hier in einer speziellen Form (siehe Tabelle 8.1) anzugeben. Desweiteren

Zeile	$T_{Abspiel}$	Werte
1	zu streichende Saite	*1 - 4*
2	Aufsetzpunkt am Bogen	*in Meter*
3	Absetztpunkt am Bogen	*in Meter*
4	Spielrichtung	1 = ziehen, -1 = schieben
5	Stegentfernung	*in Meter*
6	Note	*z. B. 1/8*

Tabelle 8.1.: Abzuspielende Matrix

sind bestimmte Grundparameter bestimmbar, wie die Taktzahl pro Minute (BPM), eine Referenznote (N_ref), eine Voreinstellung (Offset) für den Spielabstand über den Saiten, welcher der Trajektorie überlagert wird und für die Testphase wie auch für die Feinabstimmung benutzt werden kann. Auch die Zeit für einen Saitenwechsel (t_ueber) und der An- bzw. Abfahrt (T_Init) sind bestimmbar. Es folgt eine kurze Beschreibung über die wichtigsten Abläufe der Trajektorienerzeugung:

Bevor *Start.m* ausgeführt werden kann, sind einige Initialisierungen durchzuführen. Die geschieht mit der Ausführung von *MAINInit.m*, welche ihrerseits *lbr3ViolinInit.m* aufruft. Hierbei werden eine Reihe von Parametern gesetzt, welche für die Positionsregelung aber nicht mehr alle benutzt werden bzw. später neu gesetzt werden. Unausweichlich ist allerdings eine Festlegung der Anfangsparameter des LBR III[1] in Gelenkwinkelkoordinaten. Für eine Simulation können hierbei belibiege Werte angegeben werden, wobei

[1] Leichtbauroboter der III. Generation

natürlich darauf zu achten ist, dass die Anfangsposition keine singuläre Stellung einnimmt (z. B. vollkommen ausgestreckter Arm), währen in einem praktischen Versuch diese Werte gemessen und übertragen werden müssen, oder aber man dafür sorgt, dass der Roboter sich bei Ausführung der Bewegungsabläufe genau die im Programm angegebenen Gelenkwinkel einnimmt (explizites Anfahren der Position). Desweiteren wird in den Initialisierungsprogrammen auch die Abtastrate (sampleTime) festgelegt und verschiedene Pfade (Vorwärts- und Inverse-Kinematik) gesetzt. Allerdings sind auch diese Pfade nur für Simulationszwecke gedacht, da die Koordinatentransformationen später auf dem Echtzeitrechner(n) durchgeführt werden. Sind alle Initialisierungen gesetzt, kann folgend *Start.m* aufgerufen werden. Nach dem Aufruf von *Start.m* werden als erstes die Anfangsgelenkwinkel ausgelesen und in Weltkoordinaten transformiert. Folgend wird das Programmsegment *lage.m* gestartet. Hier werden nun alle konstruktiven Größen des Bogens und der Violine bestimmt, welche für die Trajektorienerzeugung von Bedeutung sind. Alle Koordinaten liegen nach Ausführung dieser Datei in *violinen_ koordinaten.mat* gespeichert vor und können von nun an in beliebige Programmsegmente geladen und benutzt werden. Nun folgt die eigentliche Erzeugung der Trajektorie, beginnend mit dem Aufruf von *Matrizenberechnung.m.* In diesem Segment werden nun alle anzufahrenden Punkte auf Grundlage der abzuspielenden Matrix berechnet und in T_{Matrix} gespeichert. Diese Berechnungen sind von besonderer Bedeutung, da sie gravierenden Einfluss auf den eigentlichen Spielvorgang besitzen. Die Matrix besitzt zu diesem Zeitpunkt den in Tabelle 8.2 dargestellten Aufbau. Wie leicht zu erkennen ist, beinhaltet sie in den Zeilen

Zeile	T_{Matrix}	Werte
1	fortlaufende Nummerierung	1...n
2	Aktion	0 = streichen, 1 = überführen
3	Wert der Transf.-Matrix (1,1)	Korrd.-Fragment f. Orientierung
4	Wert der Transf.-Matrix (2,1)	Korrd.-Fragment f. Orientierung
5	Wert der Transf.-Matrix (3,1)	Korrd.-Fragment f. Orientierung
6	Wert der Transf.-Matrix (1,2)	Korrd.-Fragment f. Orientierung
7	Wert der Transf.-Matrix (2,2)	Korrd.-Fragment f. Orientierung
8	Wert der Transf.-Matrix (3,2)	Korrd.-Fragment f. Orientierung
9	Wert der Transf.-Matrix (1,3)	Korrd.-Fragment f. Orientierung
10	Wert der Transf.-Matrix (2,3)	Korrd.-Fragment f. Orientierung
11	Wert der Transf.-Matrix (3,3)	Korrd.-Fragment f. Orientierung
12	Wert der Transf.-Matrix (1,4)	Korrd.-Fragment f. Lage
13	Wert der Transf.-Matrix (2,4)	Korrd.-Fragment f. Lage
14	Wert der Transf.-Matrix (3,4)	Korrd.-Fragment f. Lage
15	Note	z. B. 1/8

Tabelle 8.2.: Matrix der anzufahrenden Punkte

3 bis 11 die Orientierungsmatrix sowie in den Zeilen von 12 bis 14 den Ortsvektor. Die genaue Position eines jeden anzufahrenden Punktes im Raum ist mit dieser Matrix nun also bekannt. Weiterhin ist durch die Variable *Aktion* ebenfalls festgelegt, auf welche

Weise zwischen diesen Punkten Interpoliert werden soll. Als nächstes wird in *Start.m* die Datei *Trajektorie.m* aufgerufen. Sie ist für die Interpolation und die Zeitparametrisierung zuständig. Zu diesem Zwecke wurde sie in drei übergeordnete Einheiten aufgeteilt: *Start-Ende-Regelung, Streichen* und *Überführen*. In der Einheit *Start-Ende-Regelung* berechnet die Funktion *Bezier* die eigentliche Kurve von der Anfangsposition zum ersten Streichpunkt bzw. vom letzten Streichpunkt zur Endposition. Die Geschwindigkeitsregelung wird von der Funktion *VRegelungStartEnde* übernommen. Diese berechnet variable Punktabstände und übergibt diese an *distanzen3D,* welche die Kurve aufgrund der vorgegebenen Punktabstände umstrukturiert. Die Geschwindigkeit bei An- und Abfahrt ist dabei vor allem abhängig von *T_ Init*, aber auch von der Sollgeschwindigkeit des ersten bzw. letzten Streichvorganges. Da diese Sollgeschwindigkeit des Streichvorganges wiederum von anderen Faktoren abhängig ist, wird eine explizite Geschwindigkeitsvorgabe für An- und Anfahrten nur schwer realisierbar sein. Die Einheit *Überführen* nutzt neben der Funktion *Bezier* zusätzlich noch die Funktion *Bezierpunkte*, welche ermittelt, ob eine einfache oder gekoppelte Bézierfunktion für die jeweilige Überführung von Nöten ist. Zur Ermittlung der Bézierpunkte wird die Funktion *geradendarstellung* herangezogen, welche u. a. Werte wie Anstieg, Abstände von den Saiten und Schnittpunkte berechnet. Die Berechnung der Punkteabstände übernimmt *VRegelung. Distanzen3D* implementiert diese dann wie schon zuvor in die Überführungs-Kurve. Die letzte Einheit betrifft das Streichen einer Saite. Hier ist besonderes Augenmerk auf die Dauer eines Streichvorgang zu legen, welcher letztlich durch die Absolutzahl der Punkte bestimmt ist, genauso wie auf die Streichgeschwindigkeit, welche von den hier konstanten Punkteabständen festgelegt wird. Alle aus jeder Einheit berechneten Punkte werden fortlaufend in *T_ Traj* und *R_ Traj* abgelegt, wobei *T_ Traj* die Lage und *R_ Traj* die Orientierung inne hält. Beide Matrizen werden in die Datei *Trajektorie.mat* gespeichert, welche demzufolge die gesamte Trajektorie beinhaltet.

8.3. Fazit

Mir den hier vorliegenden mathematischen Analysen und der Erstellung eines Programmes zur Trajektorienerzeugung, wurde bereits ein sehr wichtiger Schritt in Richtung Violine-Spielenden-Roboter getan. Allerdings stellt dies nur den Beginn einer Reihe bevorstehenden Aufgaben und Herausforderungen dar. Wie sich bereits zu Beginn meiner Arbeit gezeigt hatte, erweist die Berechnung der Trajektorie, trotz ausgeklügelter Mathematik, als noch sehr fehlerbehaftet. Eine zu große Punkteschrittweite sowie unzulässig hohe Unstetigkeiten der Kurve vor allem in der Orientierungssteuerung bereiten noch genauso große Schwierigkeiten, wie ein gelegentlichen Durchfahren der Geige oder das Überschreiten des Arbeitsraumes. Desweiteren stellt die Eingabe einer Note noch eher ein mathematisches Unterfangen dar, als ein musikalisches Geschick. Eine deutlich einfachere auf musikalischen Grundlagen basierende Eingabemöglichkeit sollte also her, welche gleichzeitig schon bei der Eingabe in der Lage sein sollte, Fehler wie eine Arbeitsraumüberschreitung oder unzulässig hohe Streichgeschwindigkeiten zu erkennen, um eine spätere fehlerbehaftete Trajektorienerzeugung zu vermeiden. Auch sollten durch eine sol-

che Eingabemöglichkeit einmal musikalische Grundwerte wie ein exaktes Einhalten der Takt- und Notenlängen ermöglicht werden, welche für die Musik so besonders wichtig sind. Auch die auf den ersten Blick weniger wichtig erscheinenden Faktoren, wie Lautstärke und Klangfarben, könnten unter Beachtung der uns gegebenen Möglichkeiten einmal realisiert werden. Wieder etwas weiter von musikalischen Gesichtspunkten entfernt, stehen nach der erfolgreichen Trajektorienerzeugung noch weitere sehr entscheidende, eventuell auch sehr fehleranfällige Aufgaben an: Die Erstellung eines Echtzeitmodells zur letztendlichen Ansteuerung des Roboters, inklusive Echtzeitkoordinatentransformationen und fehlerlose Übertragung der Trajektorie. Auch eine genaue Bestimmung der Position von Violine und Bogen im Raum könnte wie schon angesprochen noch denkbar große Schwierigkeiten bereiten, da, wie man sich leicht vorstellen kann, eine Positionsregelung praktische eine "blinde" Steuerung ist, welche somit sehr exakte Daten u. a. in vorm von Objektbeschreibungen benötigt.

Wie man schon aus diesen paar kurzen Überlegungen erkennen kann, sind noch eine ganze Reihe von Problemen zu lösen und mit Sicherheit auch etliche Tests durchzuführen. Aber selbst wenn alle angesprochenen Aufgaben erfolgreich bewältigt werden konnten, heißt das noch lange nicht, dass der Roboter daraufhin erfolgreich eine Geige zu spielen vermag. Denn erstens wird mit diesem Projekt bisweilen "nur" das Problem der Bogenführung angegangen (über ein Greifen der Saiten besteht noch nicht einmal ein mathematischer Ansatz), und zweitens könnte sich auch sehr leicht herausstellen, dass man durch eine reine Positionssteuerung keine Violine "klangvoll" anstreichen kann. In diesem Falle wären alternative Ansätze wie eine hybride Kraft/Positionssteuerung zu entwickeln, zu Programmieren und zu testen. Da dies aber bisweilen weder bewiesen noch wiederlegt werden konnte, werden sich die folgenden Bemühungen weiterhin auf den Grundsatz einer reinen Positionssteurung beziehen.

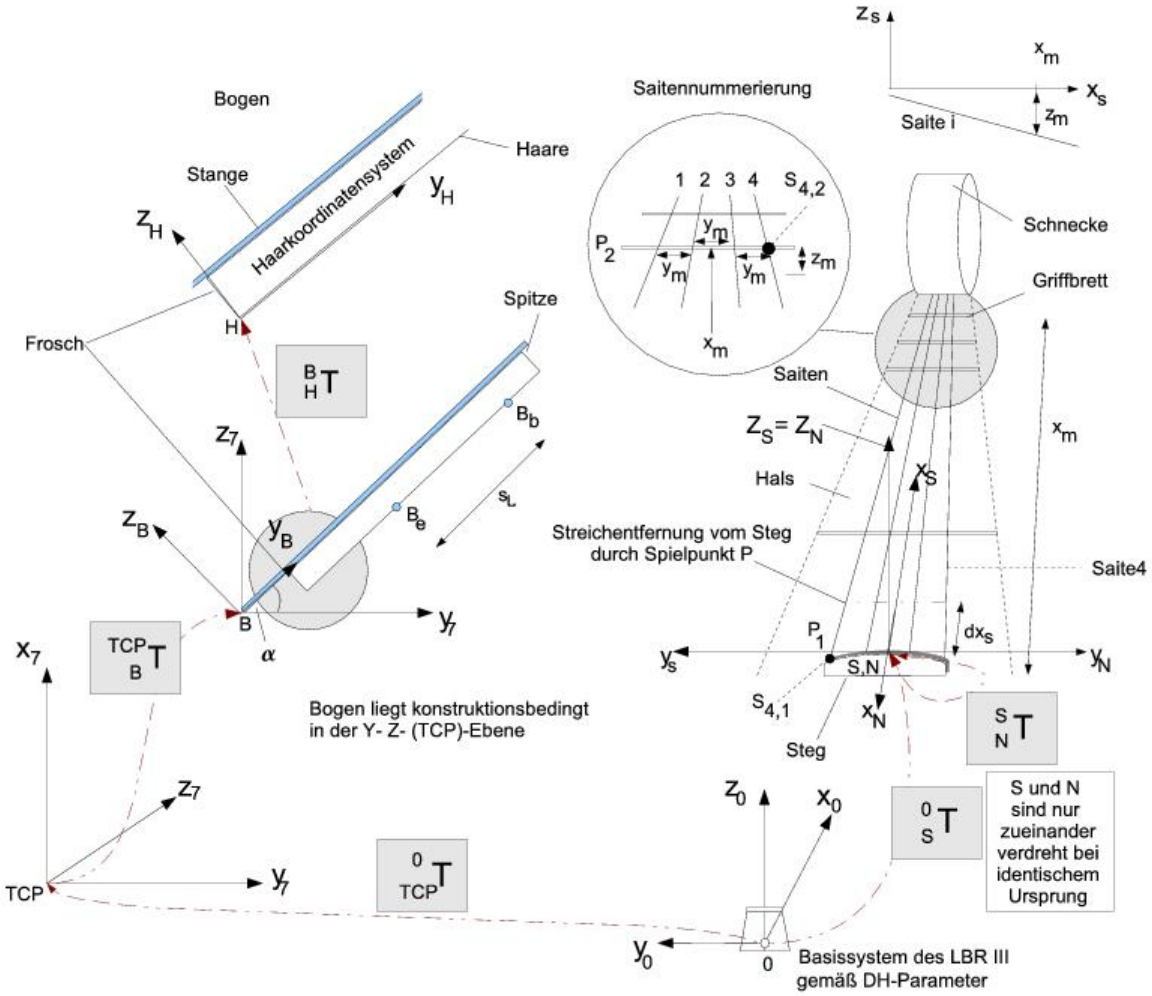


Abbildung 8.1.: Definierte Koordinatensysteme im System [29]

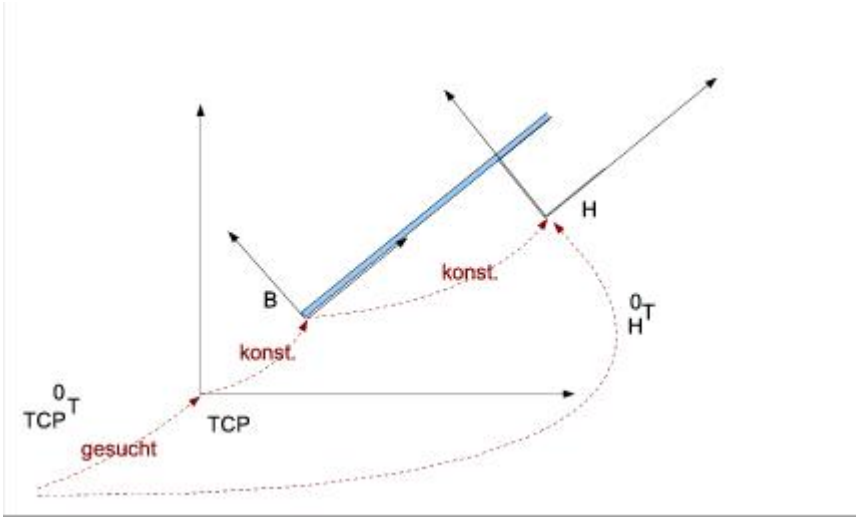


Abbildung 8.2.: Koordinatentransformation bezüglich des Bogens [29]

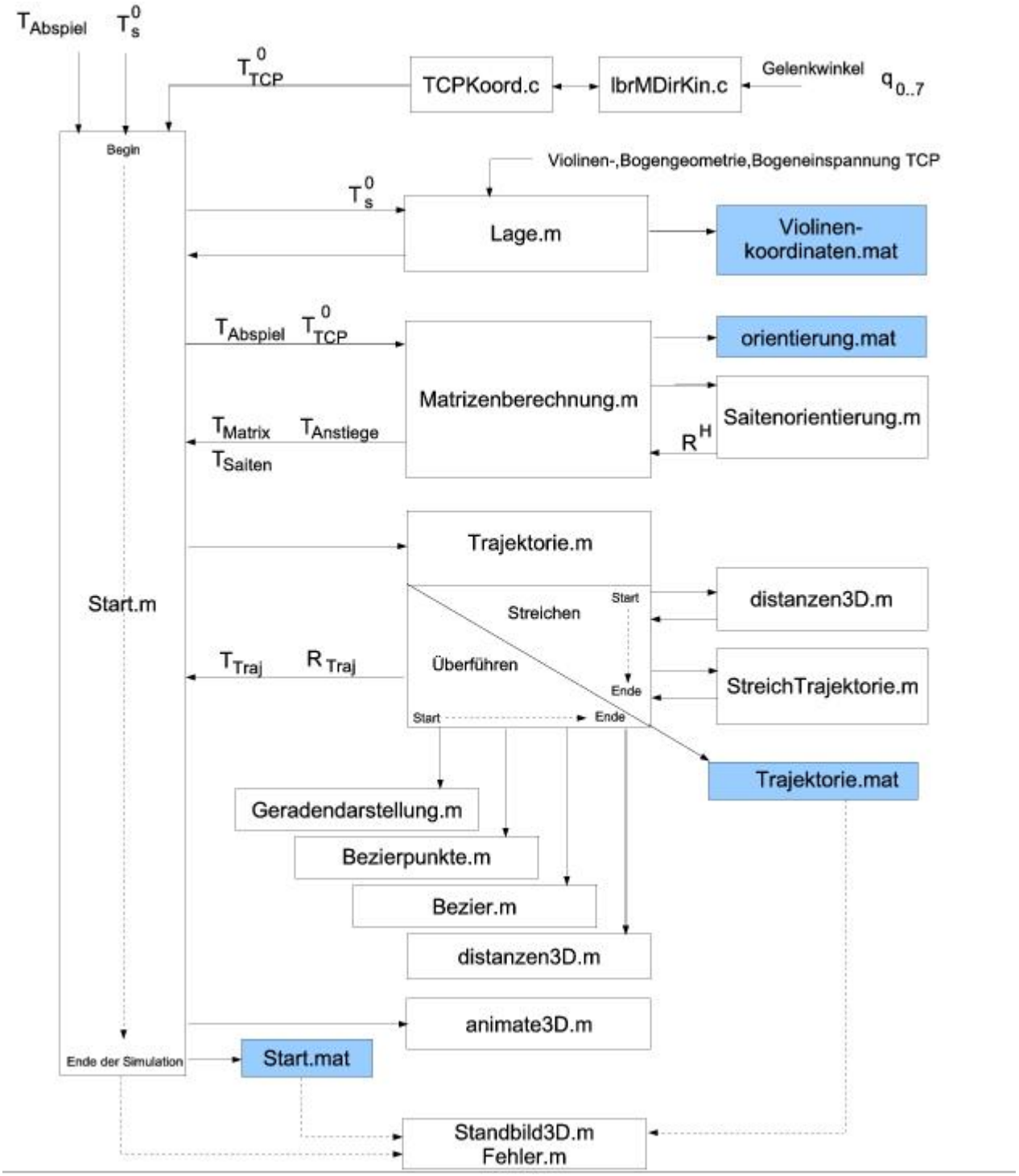


Abbildung 8.3.: Ablaufplan [30]

9. Trajektoriengenerierung

Nach einer einmonatigen Einarbeitungsphase in die entsprechende Fachliteratur der Robotik, Kinematik, Mechatronik und den jeweils dazugehörigen mathematischen Beschreibungsformen (vgl. Teil II) sowie der Praktikumsarbeit *Mathematische Analyse, Modellierung und Simulation des Violine-Spielens am LBR III* von *Haase*, beschäftigte ich mich zunächst mit der "Inbetriebnahme" des dazugehörigen Programms und dessen grundlegende Konzeption (vgl. Kapitel 8.1).

9.1. Programmanalyse & Simulation

Nach einigen erfolgreichen Trajektorienberechnungen mit entsprechender Eingabe der Abspiel-Matrix $T_{Abspiel}$, folgte zunächst eine optische Analyse der Bewegungsabläufe des TCP bzw. des Bogens, welche sich in einer matlabeigenen Simulation darstellen ließen. Diese Simulation stellte die in Matlab berechneten Trajektorienpunkte mit angeflanschten Bogen sowie die Lage der Violine direkt da (siehe Abbildungen 10.3). Allerdings sei hierbei zu sagen, dass diese Simulation noch keinerlei Regler, Massenträgheiten oder Koordinatentransformationen berücksichtigte. Folgende Beobachtungen konnten gemacht werden:

- der Bogen agierte bei Streich- und Überführungsvorgängen immer in der richtigen Lage und Orientierung
- bei Aktionsübergängen (An- und Abfahrt↔ Streichen ↔ Überführen) wurde die Geschwindigkeit meisten an der vorhergehenden oder folgenden erfolgreich angepasst, so dass fließende Bewegungen entstanden
- der Bogen wird nicht orthogonal auf die Saite gesetzt, sonder in Richtung der Streichbewegung herangeführt
- zwischen einigen Aktionsübergängen herrschten noch undefinierte Sprünge in Lage und/oder Geschwindigkeit
- bei Anfahrt des Bogens verhielt sich die Orientierungssteuerung unkontrolliert, während sie bei der Abfahrt im letzten Zustand verharrte, was ein Durchfahren der Violine begünstigte (kein Sicherheitsmechanismus)
- die Punktedichte war in Bezug auf eine Taktfrequenz des Roboter von 1000Hz, welche für die Ansteuerung vorgesehen war, viel zu gering

Nicht beobachtbar waren hierbei Faktoren wie eine erfolgreiche Rückwärtstransformation in Roboterkoordinaten, den sich daraus resultierenden Gelenkwinkelpositionen, Arbeitsraumüberschreitungen, tatsächliche relative Lagen von Objekten und TCP sowie die tatsächlich vorherrschende Geschwindigkeit. Diese müssen zu einem späteren Zeitpunkt mit Hilfe anderer Methoden simuliert oder getestet werden.

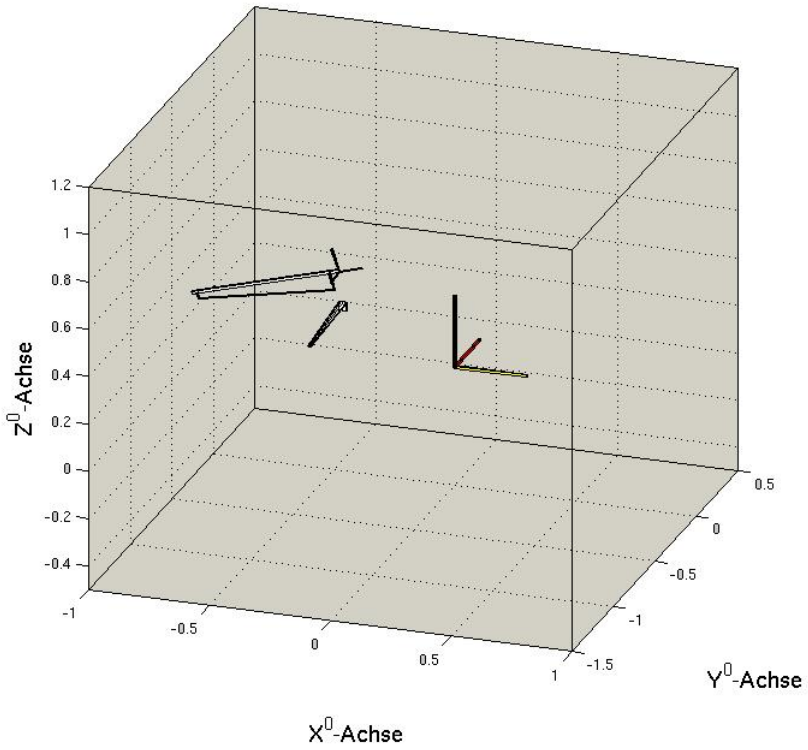
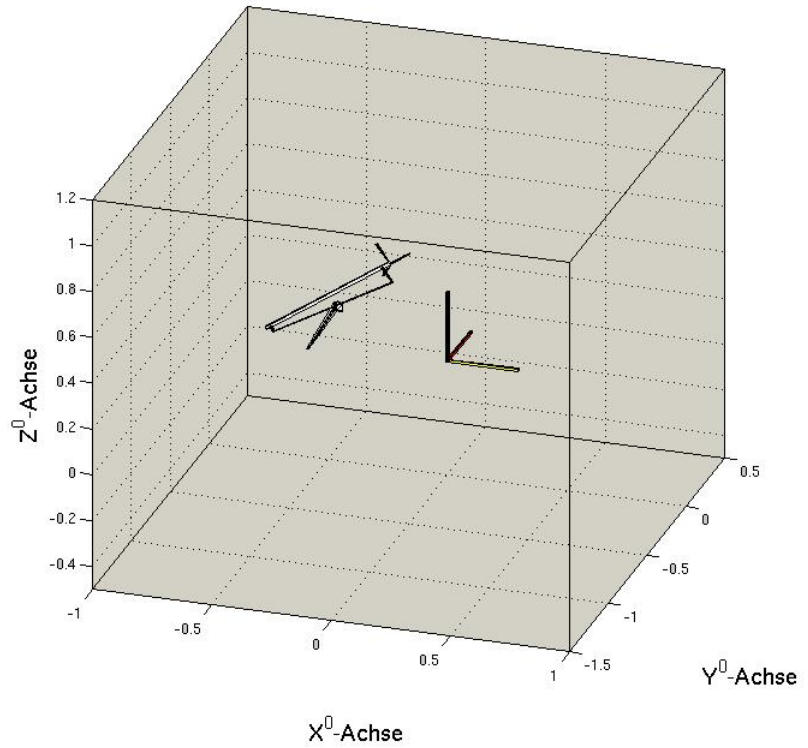


Abbildung 9.1.: Simulation der Bogenführung in *Matlab*



9.1.1. Trajektorienpunktedichtekorrektur

Als erstes sollte das Problem der zu hohen Punkteabstände angegangen werden. Dazu ist eine genaue Kenntnis über jene Parameter nötig, welche diese beeinflussen zu vermögen. Meine Hoffnungen lagen natürlich darauf, dass sich durch eine genaue Analyse und Verifizierung genau dieser Parameter die Probleme ausreichend gut beheben ließen, ohne die vorhanden trajektorienerzeugenden Algorithmen grundlegend verändern oder gar verwerfen zu müssen. Ich sollte nicht enttäuscht werden.

Tatsächlich ließen sich die Entfernungen zwischen den Punkten auf ein paar grundlegende Parameter zurückführen: Wie schon im Abschnitt 8.2.1 angemerkt, existieren für die Festlegung der Geschwindigkeiten, und somit auch für die Punktedichte, einige übergeordnete Faktoren: Die Abtastrate *T_Sim,* die Taktzahl pro Minute *BPM,* die Referenznote *N_Ref*, die Initialzeit *T_Init* und zusätzlich auch noch die Überführungszeit *t_ueber*. Um die Punktedichte eines Streichvorganges zu ändern, existieren, in Bezug auf einem positionsgesteuerten Bogen, keine Möglichkeiten zur Abänderung dieser, ohne dabei den musikalischen Effekt der Lautstärke, welcher durch Druck und Streichgeschwindigkeit hervorgerufen wird, zu beeinflussen. Dies geht daraufhin zurück, das die Geschwindigkeit direkt proportional abhängig von der Entfernung der Trajektorienpunkte und somit direkt umgekehrt proportional von der Punktedichte ist und darüber hinaus nicht mit einer Druckänderung kompensiert werden kann. Somit ist für den Streichvorgang lediglich eine oberste Grenze der Geschwindigkeit zu setzten, welche in Kombination mit der Notenlänge und Einhaltung des streichbaren Bogenbereiches später vom *Liedgenerator* übernommen wird. Die einzige in Betracht zu ziehende Möglichkeit währe eventuell eine Änderung der BPM. Da diese Änderung die gesamte Trajektorie betrifft, bleiben die Verhältnismäßigkeiten weitgehend konstant und eine musikalische "Verfälschung" findet nicht statt.

Etwas anders verhält es sich bei den Überführungsvorgängen, welche vor allem von den Faktoren Streichgeschwindigkeit vorher, Streichgeschwindigkeit folgend, Überführungszeitvorgabe, zurückzulegende Strecke und natürlich auch vom BMP abhängig sind. Dabei gelten folgende Zusammenhänge für die Streichgeschwindigkeiten und somit für v_{ist} und v_{soll} der Überführung:

$$v_{streich} = \frac{SL}{\frac{60 \cdot T_{Sim}}{BMP \cdot N_{ref} \cdot T_{Matrix}(15,i) \cdot T_{Sim}}} \tag{9.1}$$

oder nach Vereinfachung

$$v_{streich} = SL \cdot BMP \cdot N_{ref} \cdot T_{Matrix}(15,i) \cdot 60^{-1} \tag{9.2}$$

SL = Weglänge Streichvorgang

Diese Faktoren sind im Grunde nicht zu beeinflussen oder zumindest nicht wünschenswert. Die eigentliche Kurve der Überführung wird von der schon erwähnten Funktion *Bezierpunkte* bzw. *Bezier* erstellt, welche aus einfachen oder gekoppelten Bézierfunktion je nach Überführungsart eine geeignete berechnen. Sollen dieses Algorithmen unangetastet bleiben, was aufgrund der guten Funktionalität dieser hier angestrebt wird, so verbleiben nur noch die zwei Parameter *BPM* und *t_ueber* zur Variation der Punktedichte. Der BPM ist wie schon erwähnt ein Parameter, welcher die gesamte Trajektorie

beeinflusst. Da die Überführungszeit aber eindeutig von den Parameter t_ueber festgelegt wird, ist eine Beeinflussung der Punktedichte bei Überführungen nur sehr indirekt durch Veränderung der Start- und Endbedingungen laut 9.1 und 9.2 möglich. Aus diesen Erläuterungen ist nun erkennbar, dass eine Anpassung der Punktedichte für den Saitenwechsel am günstigsten mit t_ueber beeinflusst werden kann und sollte. Dennoch ruft auch dieses Beeinflussungsmethode sehr unelegante Effekte hervor: Die mit t_ueber festgelegte Überführungszeit ist universell und gilt für alle Überführungen. Da Saitenwechsel je nach Art (aus Sicht des zurückzulegenden Weges) unterschiedliche Längen aufweisen, werden die Überführungsgeschwindigkeiten stark variieren, was sehr unnatürlich wirkt und selbst bei nur sehr kleinen Überführungen große Zeit in Anspruch nehmen wird. Eine übergeordnete Berechnungseinheit namens dem schon erwähnten *Liedgenerator* wird diesen Missstand nachfolgend beheben.

9.1.1.1. An- und Abfahrt des Bogens

Bei der An- und Abfahrt des Bogens erwiesen sich die Probleme etwas anderer Natur. Während in der Abfahrtstrajektorie erst gar keine Orientierungssteuerung programmiert wurde, welche, wie sich noch herausstellen wird, unerlässlig ist, verhielt sich die Orientierungstrajektorie bei Anfahrt recht chaotisch. Basierend auf einer Winkelberechnung in den entsprechenden Ebenen, verlief sie in zu großen Sprüngen und war auch nicht durch die BPM beeinflussbar. Der Vorgang der Orientierungssteuerung lief auf eine Differenzenbildung der aktuellen und geforderten Orientierung hinaus. Dabei wurde die Matrix $T_{aktuell}$ invertiert und durch anschließender Multiplikation mit der Matrix $T_{Lage_{soll}}$ die entsprechende Differenzenmatrix T_{diff} erstellt (siehe 9.3).

$$T_{diff} = T_{aktuell}^{-1} \cdot T_{Lage_{soll}} \tag{9.3}$$

Während sich die Matrix $T_{Lage_{soll}}$ aus dem anzufahrenden Punkt des ersten Streichvorganges ableitet, wird die Matrix $T_{aktuell}$ aus der Anfangsposition erstellt und ständig aktualisiert. Durch eine geeignete Auswahl der Werte in T_{diff} ist es möglich, mithilfe trigonometrischer Berechnungen die erforderlichen Winkel zum Erreichen der Sollposition zu ermitteln. Implementiert in eine Orientierungsmatrix (vgl. 7.2.3) kann somit der TCP geeignet geneigt werden. Dies darf bei einer Taktfrequenz von 1000Hz natürlich nicht in einen Schritt passieren. In der Programmierung wurden deswegen (wahrscheinlich) die Differenzwinkel mit dem Faktor 20 dividiert. Das dies bei weitem nicht ausreicht ist recht trivial. Angenommen es ist eine Winkeländerung von 180° (in einem Gelenk) von Nöten, so würde sich eine Winkeldifferenz im ersten Taktzyklus von $\frac{1}{20} \cdot 180° = 9°$ ergeben. Wie aus Kapitel 4.2 bereits bekannt ist, beträgt die maximale Winkeländerungsrate 120°/s, was eine maximale Änderungsrate von 0,12° pro Takt entspräche. Ein zu großer Dividend kann aber ebenfalls nicht genutzt werden, da in diesem Falle die Änderungsraten, vor allem bei Annäherung der gewünschten Position, stark abnehmen würden, was eine asymptotische Annäherung der Soll- zur Istposition zur Folge hätte. Aus diesem Grunde ist es nötig, einen variablen Dividenden zu kreieren. Ein angemessen großer Dividend, welcher nach jedem Taktzyklus dekrementiert wird und schließlich, beim Erreichen der eins, konstant gehalten wird, soll Abhilfe schaffen. Mit einem solchen Dividenden sind

langsame Änderungsraten möglich und das Integral-Verhalten zur Annäherung an den Sollwert bleibt weiterhin bestehen. Nachfolgende Simulationen bestätigten diese Fakten.

Nach der erfolgreichen Abänderung des Orientierungsverhaltens bei Anfahrt, bot es sich nun an, dieselben Algorithmen für eine Orientierungssteuerung bei Abfahrt zu verwenden. Ziel war es dabei, das der Bogen nach der Abfahrt wieder exakt die gleiche Lage und Orientierung einnimmt, von welcher er gestartet ist. Zusätzlich sollte ein Streifen oder gar Durchfahren der Saiten und der Violine unbedingt verhindert werden. Erster Schritt war somit eine Implementierung der modifizierten Algorithmen des Orientierungsverhaltens in die Abfahrtstrajektorie. Nach erfolgreicher Umsetzung blieb nun noch das Problem der Sicherstellung des Nichtdurchfahrens der Violine aus jeder Ausgangssituation. Hierbei für die Wahl auf den Einbau einer Korrektur-Orientierungsmatrix. Diese sollte den Bogen nach dem letzten Streichvorgang um einen definierten Winkel in der Y-Z-Ebene anheben, wobei der Vorgang auch hier integrales Verhalten aufweisen sollte. Multipliziert man diese Matrix mit der Abfahrtsorientierungsmatrix, erhält man eine fließende und vor Kollisionen sichere Abfahrt des Bogens in die entsprechende anfängliche Position. Folgend ist ein Ausschnitt des Programmcodes der Abfahrt des Bogens dargestellt[1]:

```
for r = 1:Punktanzahl
        Laufvariable = Laufvariable+1;
        T_Traj(1:3,Laufvariable) = [Kurve(1,r) Kurve(2,r) Kurve(3,r)
            ];
        T_Traj(4,Laufvariable) = 4;

        %Divident fuer den Drehwinkel (siehe unten)
        Divident = 8000-r;
        if(Divident < 1) Divident = 1; end
        %----------ENDE


        y= -45;


        R_Korrektur = [ 1     0       0          0;
                        0     cos(y)  -sin(y)    0;
                        0     sin(y)  cos(y)     0;
                        0     0       0          1];

        %inverse der aktuellen Lage
        T_aktuell_inv=inv(T_aktuell);

        %multiplikation
        T_diff = T_aktuell_inv * (T_aktuell * R_Korrektur);
        if(r > Punktanzahl/10) T_diff = T_aktuell_inv * (T_Lage_soll)
            ; end
```

[1] Nachzulesen in *trajektorie / Trajektorie_II*

```
%Drehwinkel:
al = (atan2(T_diff(2,1), T_diff(1,1))) / Divident;
be = (atan2(-T_diff(3,1), sqrt((T_diff(1,1))^2 + (T_diff(2,1)
    )^2))) / Divident;
ga = (atan2(T_diff(3,2), T_diff(3,3))) / Divident;
ca = cos(al); sa = sin(al); cb = cos(be); sb = sin(be); cg =
    cos(ga); sg = sin(ga);

R=      [ca*cb    ca*sb*sg-cg*sa    ca*sb*cg+sa*sg;
         cb*sa    sa*sb*sg+ca*cg    sa*sb*cg-ca*sg;
         -sb      cb*sg             cb*cg ];

Versch = [  T_diff(1,4)/(Punktanzahl+1-r);
            T_diff(2,4)/(Punktanzahl+1-r);
            T_diff(3,4)/(Punktanzahl+1-r)];

T_mult = [R Versch;0 0 0 1];
T_aktuell = T_aktuell*T_mult;
R_Traj(1:9,Laufvariable) = [ T_aktuell(1,1) T_aktuell(2,1)
    T_aktuell(3,1) ...
                             T_aktuell(1,2) T_aktuell(2,2)
                                T_aktuell(3,2) ...
                             T_aktuell(1,3) T_aktuell(2,3)
                                T_aktuell(3,3) ];
R_Traj(10,Laufvariable) = 4;
end;
```

Die Variable *Punkteanzahl* gibt die Anzahl der Trajektorienpunkte an und bestimmt somit die Anzahl der Durchläufe. Etwas darunter ist der *Dividend* für den Drehwinkel (letztes Drittel) definiert. Unter dem Dividenden ist die Korrekturmatrix für eine sichere Abfahrt mit den einstellbaren Winkel y erkennbar, welcher etwas später mit T_{diff} verrechnet wird. Alle berechneten Werte werden abschließend in R_{Traj} gespeichert.

9.1.2. Stetigkeit der Kurve

Mit den in 9.1.1 beschrieben Modifizierungen und Erweiterungen des Programms sollte die Punktedichte nun in einem realistischen Rahmen liegen. Um dies zu Prüfen soll ein Fehleranalyseprogramm die Trajektorie "abtasten", welches eigens zu diesen Zweck geschrieben wurde und in der Lage sein wird, alle Punkteabstände zu prüfen und bei Angabe von Maximalwerten Fehlermelden auszugeben. Somit würden auch nicht sichtbare Fehler identifiziert werden können. Dabei war nicht nur die Anzahl der Fehler entscheidend, sondern auch deren Größe und genaue Lage. Als besonders sinnvoll erwies sich die Programmierung einer Ausgabe des entsprechenden Fehlerortes in Bezug auf die aktuelle Aktion, bzw. deren Übergänge. Nach Fertigstellung des Fehleranalyse-Programmes meldete selbiges weitere Fehler in der Trajektorie. Diese befanden sich allerdings nicht in der Anfahrts-, Abfahrts- oder Überführungstrajektorie, wodurch die Modifizierungen

aus 9.1.1 bestätigt wurden. Vielmehr befanden sich sämtliche Fehler in den Bereichen der Aktionsübergänge, z. B. Streichen ↔Abfahrt, was darauf schließen lässt, dass die einzelnen aktionsbedingten Trajektorienstücke nicht reibungslos aneinander gereiht werden konnten. Allerdings traten diese Fehler nur sehr vereinzelt und ohne erkennbares Schema auf, was eine Fehlerursache aufgrund fehlerhafter Algorithmen nicht gerade untermauert. Hinsichtlich dieser Fakten soll folgend eine Glättung der wenigen Sprünge in der Trajektorie durch eine nachträgliche Interpolation betroffener Stellen diesen Missstand beseitigen.

Ziel war es dabei, Stetigkeit der Kurve bis zur ersten Ableitung zu garantieren. Zu diesem Zwecke wird die Trajektorie an den sprunghaften Stellen aufgetrennt und geeignete Zwischenwerte eingeführt. Folgendes Schema wird hierbei angewandt, wobei der Sprung hier zwischen i und $i+1$ angenommen wird:

$$Diff_{ges} = P_i - P_{i+1} \tag{9.4}$$

$$Diff_{vorher} = P_i - P_{i-1}; \quad Diff_{nacher} = P_{i+2} - P_{i+1} \tag{9.5}$$

$$\triangle Diff = Diff_{nacher} - Diff_{vorher} \tag{9.6}$$

$$Diff_{mittel} = \frac{(Diff_{vorher} + Diff_{nacher})}{2} \tag{9.7}$$

$$n_{P_{neu}} = \left| \frac{Diff_{ges}}{Diff_{mittel}} + 2 \right|, \qquad n \,\epsilon\, \mathbb{N} \tag{9.8}$$

$$\frac{d\,Diff}{di} = \frac{\triangle Diff}{n_{P_{neu}} + 1} \tag{9.9}$$

Die P kennzeichnen hierbei die Werte des Trajektorienvektors jeder einzelnen Zeile in der *i-ten* Spalte. Durch $n_{P_{neu}}$ ist nun die Anzahl der einzufügenden Punkte bekannt während mit $\frac{d\,Diff}{di}$ die Änderungsrate mit steigenden i definiert wird. Folgend wird die die Trajektorie um n-Werte verlängert und die entsprechenden neuen Werte eingefügt:

$$\triangle P_{i+x} = Diff_{vorher} + \frac{d\,Diff}{di} \cdot x, \qquad x = 1...n_{P_{neu}} \tag{9.10}$$

$$P_{i+x} = P_{i+x-1} + \triangle P_{i+x} \tag{9.11}$$

Anzumerken sei hierbei, das sich diese Methode der Sprungglättung nur auf einzelne Sprünge, nicht aber auf kontinuierlich fehlerhaften Trajektorien anwenden lässt. Die nachfolgenden Tests dieser Interpolation[2] lieferten sehr gute Ergebnisse.

9.1.3. Erweiterte Interpolationsmethoden

Die in 9.1.2 beschriebenen Interpolationsmethoden liefern für die meisten Trajektorien ausreichend gute Ergebnisse bezüglich der Stetigkeit der Kurve. Dennoch konnte gezeigt werden, dass bei sehr komplexen Trajektoriengenerierungen, i.d.R. durch Nutzung der

[2]Nachzulesen in *sprung_ glaettung / Sprung_ Glaettung_ VII.m*

programmunterstützten Eingabe (vgl. Unterabschnitt 9.2.2.2 in Abschnitt 9.2), gegebenfalls Sprüngen auftraten, welche direkt aufeinander folgten und somit mit bisheriger Programmierung nicht aufgefangen werden konnten. Dieser Missstand soll nun mit einer Erweiterung der vorhandenen Interpolationsmethoden weitmöglichst ausgeräumt werden. Die grundlegenden Algorithmen aus 9.4 bis 9.11 werden zu diesem Zwecke allerdings nicht angetastet, wohl aber die Zuspeisung der Daten mit welche der Interpolator arbeitet.

Die notwendigen Daten für eine Interpolation der Kurve an richtiger Stelle werden vom Programmfragment *Fehlererkennung* in Form einer Fehlermatrix bereit gestellt. Die Matrix ist wie in Tabelle 9.1 dargestellt aufgebaut. Die Angabe der Spaltennummer stellt

Zeile	Fehlermatrix
1	$\text{Zeilennummer}_{Fehler}$
2	$\text{Spaltennummer}_{Fehler}$
3	tausendfache Größe d. Fehlers

Tabelle 9.1.: Aufbau der Fehlermatrix

hierbei die Spalte direkt vor dem Fehlerort dar. Diese Matrix wird in Folge entsprechend dieser Form direkt an den Interpolator übergeben. Um die Interpolationsalgorithmen auch auf kontinuierlich fehlerhafte Trajektorienstücke anwenden zu können, erschien es sehr sinnvoll, eine Modifizierung dieser Matrix vorzunehmen. Hierfür sollte die Matrix als erstes erweitert werden, so dass in ihr nicht nur Anfangsspalte sondern auch Endspalte des Fehlers eingeschrieben würden. Folgen in einer Trajektorie nun mehrere Fehler aufeinander, so werden in Folge dessen mehre Fehleranfangs- und Endwerte zusammen fallen, welche wiederrum aus der Matrix leicht entfernt werden können. Übrig bleiben nur Anfangs- und Endpunkte von jeder kontinuierlichen Fehlerkette. Im weiteren Vorgehen wird nun der Interpolator geeignet umgeschrieben, dass er aus dieser modifizierten Matrix lesen kann, wobei er durch die neue Form der Matrix jede kontinuierliche Fehlerkette als einen Sprung ansehen wird, welcher er nach bisherigen Methoden zu Interpolieren vermag. Dabei muss beachtet werden, das bei einer Korrektur der Trajektorienmatrix nicht nur Werte eingefügt werden müssen, sondern auch entsprechend der Länge der Fehlerkette entfernt.

Durch diese Modifizierungen ist es nun auch möglich kontinuierlich fehlerbehaftete Trajektorien bei Nutzung derselben Algorithmen zu beheben. Es soll aber darauf hingewiesen werden, dass nach einer Interpolation zu stark fehlerhafter Trajektorien zwar in Folge dessen stetige Bewegungsabläufe entstehen, dies aber zu einer nicht unerheblichen Verfälschung der ursprünglich angedachten Kurve führen kann. Somit weißt auch diese Interpolationsmethode Grenzen auf.

9.2. Methoden der Liedgenerierung

Nachdem nun alle Probleme der Trajektorienbeschaffenheit weitgehenst behoben sind, will ich mich nun mit einen der Trajektorienberechnung übergeordneten Programm namens *Liedgenerator* beschäftigen. Ursprünglich erdacht wurde dieses zur automatischen

Kreierung der abzuspielenden Matrix $T_{Abspiel}$ durch Eingabe beliebiger musikalischer Werte, wie Tonart, Notenlänge, Pausen und Lautstärke. Darüber hinaus wurde das Programm mit einer kompletten Menüführung konzipiert, mit deren Hilfe eine leichte Bedienung und zusätzliche Funktionen möglich sind.

9.2.1. Die Bedienoberfläche

Die Implementierung des *Liedgenerators* stellt praktisch eine übergeordnete Maske in dem vorhandenen Programm *Violine*[3] dar. Er dient zugleich auch als neue Startdatei: Wird nun im Kommandofenster die Startdatei *Start.m* aufgerufen, so wird in Folge dessen als erstes der Liedgenerator gestartet, welcher von sich aus alle weiteren Funktionen übernimmt bzw. weitergibt. Das Menü ist folgend aufgebaut:

1. Neuen Titel schreiben...
2. laden...
3. speichern...
4. Note zum aktuellen Titel hinzufügen
5. Daten zeigen
6. aktuellen Titel korrigieren
7. Trajektorie berechnen
8. Traj_Vektor.mat erzeugen & beenden
9. Simulieren...
10. Beenden
11. Fehleranalyse

Die meisten dieser Befehle sind selbsterklärend. Menüpunkt eins stellt die eigentliche Aufgabe des Generators dar. Auf diesem wird später noch genauer eingegangen (Abschnitt 9.2.2). Mit den Menüpunkten zwei und drei ist es möglich, Lieder, in Form einer abzuspielenden Matrix $T_{Abspiel}$, oder Trajektorien, in Form von T_{Traj} und R_{Traj}, unter eigenen Namen zu laden oder zu speichern. Die sich aktuell im temporären Speicher befindlichen Daten kann man sich mit Menüpunkt fünf anzeigen lassen. Mit dem Befehl *Trajektorie berechnen* werden die Daten der Matrix $T_{Abspiel}$ an *Start.m* übergeben und im weiteren Verlauf wie üblich berechnet. Für die Methode der *programmunterstützen Eingabe* von Titeln war es allerdings notwendig, die Algorithmen der Trajektorienberechnungen zu modifizieren, damit weitere Fähigkeiten der Bogenbewegung möglich werden konnten und der Trajektoriengenerator überhaupt die neue Form der Eingabedaten zu verarbeiten

[3] siehe Kapitel 8.2

vermag[4]. Auch hierauf werde ich später nochmal explizit eingehen. Wurde die Trajektorie erfolgreich erstellt, kann man sich diese mit Menüpunkt acht in einem geeigneten Vektor zur Übertragung der Daten auf einem Echtzeitsystem umwandeln lassen. Die Simulationsmöglichkeiten aus Menüpunkt neun stellen lediglich matlabeigene Simulationen dar, wobei Daten aus verschiedenen Trajektorien gewählt werden können. Punkt elf startet die schon in Abschnitt 9.1.2 erwähnte Fehleranalyse, welche hierfür noch etwas konzipiert wurde.

9.2.2. Generierung eines Titels

Mit der Anwahl von Punkt eins im (Haupt-) Menü, ist es möglich, einen "Titel" oder auch nur einzelne Streichvorgänge in Form der abzuspielenden Matrix $T_{Abspiel}$ zu erstellen. Dabei stehen zwei Varianten zur Auswahl: Die manuelle Eingabe und die programmunterstützte Eingabe.

9.2.2.1. Manuelle Eingabe

Die sogenannte manuelle Eingabe bezieht sich noch auf die Form von $T_{Abspiel}$ nach *Haase* (vgl. Tabelle 8.1). Ziel war es dabei, vor allem nicht mehr Aufsetz- und Absetzpunkt des Bogens auf die Saite angeben zu müssen. Weiterhin sollte die Angabe einer Tonart und das wählen einer Lautstärke möglich werden. Alle benötigten Daten sollten zudem nicht mehr durch Abänderung des Programmcodes, sondern durch entsprechende Eingaben im Kommandofenster zur Verfügung gestellt werden können. Mit der "manuellen Eingabe" eines Titels sind folgende Eingaben erforderlich: Tonart, Notenlänge, Lautstärke, Streichrichtung und der Spielabstand vom Steg. Da in das Projekt ein Greifen der Saiten noch nicht implementiert wurde, sind als Angabe für die Tonart bis jetzt nur vier Möglichkeiten entsprechend der Tonart der leeren Saiten "freigeschaltet": E, A, D und G. Die Angabe der Lautstärke geschieht über eine abstrakte Skale von eins bis zehn. Hierbei wird die Streichgeschwindigkeit des Bogens über den Saiten definiert. Notenlänge und Lautstärke ergeben zusammen den Streichweg (vgl. 9.12[5]), woraus wiederum geeignete Ansetz- und Absetzpunkte des Bogens gewählt bzw. berechnet werden (vgl 9.13 bis 9.15).

$$l_{Streich} = t_{Note} \cdot \frac{Lautstaerkefaktor}{10} \cdot (l_{Haare} - 0,05) \tag{9.12}$$

$$l_{Diff} = l_{Haare} - l_{Streich} \tag{9.13}$$

$$P_{Ansetz} = P_{Haarbeginn} + \frac{l_{Diff}}{2} \tag{9.14}$$

$$P_{Absetz} = P_{Ansetz} \pm l_{Streich} \tag{9.15}$$

Mit der Eingabe der Streichrichtung und des Abstandes vom Steg, welcher vor allem für die Klangfarbe von Bedeutung ist, sind nun alle Werte für die Matrix $T_{Abspiel}$ nach *Haase* bekannt und werden folgend eingetragen. Gleichzeitig wird eine zweite Matrix *Lied*

[4] Das modifizierte Programmfragment trägt folgend den Namen *Matrizenberechnungen_II.*

[5] t_{Note} ist hierbei Dimensionslos

erstellt, welche die einzelne Werte in einer weniger abstrakten Form für die Überprüfung durch den Nutzer abspeichert. Nachfolgend ist es nun möglich beliebig viele weitere Noten anzuhängen. Nach jeder eingegebenen Note wird zudem das Programm *Fehleranalyse* aufgerufen, welches die eingegebenen Werte auf Fehler, vor allem in Bezug auf Überschreitung des Bogenbereiches und unzulässig hohe Spielgeschwindigkeiten (je nach festgelegten Grenzwerten), überprüft. Die Festlegung der An- und Absetzpunkte des Bogens auf die Saite basiert auf der Methode der Mittelung: Ist die Weglänge bekannt, so werden An- und Absetzpunkte so gewählt, dass der Streichvorgang im mittleren Bereich des Bogens ausgeführt wird. Unter Einbeziehung der Streichrichtung und einer möglichst genauen Eingabe der Haarlänge abzüglich einer Toleranz, können diese Punkte genau definiert werden (vgl. Kapitel 11).

9.2.2.2. Programmunterstützte Eingabe

Die *programmunterstützte Eingabe* stellt eine verbesserte Methode der manuellen Eingabe dar. Hierbei sollte bei der Eingabe von Daten vor allem mehr auf musikalische Gesichtspunkte eingegangen werden und weniger auf die bewegungstechnischen. Desweiteren sollte es möglich werden, "richtige" Notenlängen zu kreieren, die neben der eigentlichen Streichdauer auch variable Überführungszeiten sowie gewollte Pausen berücksichtigen würden.

Das dies von nicht unerheblicher Bedeutung ist, möchte ich hier einmal kurz anführen: Angenommen, es seien in einem Stück zwei Streichvorgänge zu je 1/8 und 1/4 von Nöten. Berechnet man nun zu diesen Noten die erforderlichen Streichzeiten, so wäre eine Einhaltung der Notenlängen scheinbar gewährleistet. Musikalisch ist aber nicht die Dauer des jeweiligen Streichvorganges von Bedeutung, sondern vielmehr die Dauer vom Beginn eines Tones bis zum erklingen des darauffolgenden. Im Idealfall sollte sich der Ton der ersteren Noten direkt an dem der darauffolgenden anschließen, so, dass keine "Klangpause" entstünde, vorausgesetzt, es wurde keine explizit vorgesehen. Da für einen Saitenwechsel aber eine gewisse Zeit für die Überführung von Nöten ist, lässt sich dieser Missstand nur dadurch beheben, in dem man die Klangpause (= Überführungszeit) von der Streichzeit subtrahiert. Obwohl dadurch der eigentliche Ton nicht mehr der vorgegebenen Dauer entspräche, wäre nur so eine Einhaltung der Notenlänge durch Streich- und Überführungszeit und somit der Takt zu gewährleisten.

Handelt es sich wie bei der *manuellen Eingabe* um konstante Überführungszeiten, wäre dies kein sonderliches Problem. Allerdings erweist sich diese Methode bei näherer Betrachtung als recht unelegant: Da als Bezugspunkt der konstanten Überführungszeit zur Vermeidung zu hoher Geschwindigkeiten immer der längstmögliche Weg herangezogen werden muss, ergibt sich für jede Note eine relativ lange Klangpause, was musikalisch nicht wünschenswert ist. Da sich bei der jetzigen Programmierung nach jeder Note eine Überführung anschließt, würden sich sogar Klangpausen beim Anspielen ein und derselben Saite ergeben, was nicht gerade sinnvoll erscheint. Aus diesen Grunde sind in der *programmunterstützten Eingabe* vor allem zwei wichtige Neuerungen realisiert worden: Variable, von der Wegstrecke abhängige Überführungszeiten und das Vermeiden von Überführungen beim Anspielen derselben Saiten, sofern dies der Bogenbereich zulässt.

Diese Tatsachen schließen natürlich eine automatische Bestimmung der Streichrichtung mit ein. Somit sind für die *programmunterstützte Eingabe* nur noch folgende Eingaben nötig: Tonart, Notenlänge, Lautstärke und der Abstand vom Steg. Zusätzlich wird auch noch die Eingabe einer Pausennote möglich werden.

Adaptive Überführungszeiten Um eine adaptive Überführungszeit zu implementieren, ist es notwendig, die Weglänge der Überführung zu kennen. Diese Längen werden für jede Überführung von der Funktion *Bezierpunkte* in *Trajektorie_II*[6] zur weiteren Verarbeitung bereitgestellt. Darauf hin wird unter Zuhilfenahme der Weglänge für jede dieser Überführungen ein Überführungszeitfaktor wie folgt berechnet:

$$F_{Ueberf} = \sqrt{\frac{l_{aktuell}}{l_{max}}} \tag{9.16}$$

Die Radizierung soll hierbei einem zu schnellen Abfallen der Überführungszeiten entgegenwirken. Durch Multiplikation des Wertes *t_ueber (=t_ueber_fest),* welcher nunmehr als maximale Überführungszeitkonstante fungiert, mit dem Überführungszeitfaktor F_{berf}, erhält man laut 9.17 die entsprechende Überführungszeit jeder Überführung in Abhängigkeit der Weglänge.

$$t_{uber_{neu}} = t_{ueber_{fest}} \cdot F_{Ueberf} \tag{9.17}$$

Pausennote Wie schon in Abschnitt 9.2.2.2 erwähnt, soll es ebenfalls möglich werden, eine gezielte Pause in einen Stück zu integrieren. Dabei bietet sich eine Erweiterung der variablen Überführungszeiten sehr gut an. Der Liedgenerator wurde hierfür so konzipiert, dass nach jeder Noteneingabe eine Pausenangebe möglich ist. Diese wird genauso wie eine Notenlänge angegeben (z. B. 1/4) und in die entsprechende Spalte der letzten Note der Matrix $T_{Abspiel}$ geschrieben. Die modifizierte Funktion *Matrizenberechnungen_II* schreibt diese dann an entsprechender Stelle in die Matrix T_{Matrix} (siehe Anhang 9.2.2.3). Nun wird diese gewünschte Pausenzeit zusammen mit dem Überführungszeitfaktor verrechnet:

$$t_{ueber_{end}} = t_{ueber_{fest}} \cdot \sqrt{\frac{l_{aktuell}}{l_{max}}} \cdot \frac{60}{BMP \cdot N_{ref}} \cdot T_{Matrix}(16, a) \tag{9.18}$$

$$t_{ueber_{end}} = t_{ueber_{fest}} \cdot F_{Ueberf} \cdot t_{Pause} \tag{9.19}$$

Die Pause wird also einfach in die Überführung optisch unmerkbar integriert. Musikalisch gesehen hat dies denselben Effekt, als wenn der Bogen für die Zeit der Pause verharren würde. Einzige Voraussetzung: Es muss sich nach jeder Note eine Überführung anschließen.

Adaptive Streichzeiten Noch notwendiger als bei konstanten Überführungszeiten erwiesen sich variable Streichzeiten bei Integration variabler Überführungszeiten. Andern Falls würde jedes zu spielende Stück, selbst bei ausgezeichneter Klangqualität und richtigen Noten, in Bezug auf Taktverhalten und Liedgeschwindigkeit recht chaotisch klingen.

[6] entspricht der Funktion *Trajektorie* mit modifizierten Algorithmen

Um eine variable Streichzeit zu kreieren ist es notwendig, die sich anschließende Überführungszeit (ohne Pausennote) zu kennen. Nur so ist es zusammen mit der Überführungszeit möglich, die angegebene Notenlänge musikalisch korrekt umzusetzen. Die Überführungszeit wird wie in Paragraph 9.2.2.2 beschrieben berechnet. Allerdings ist es von Nöten, die Überführungszeit bereits vor der Berechnung der Überführung zu kennen. Zur Umgehung dieses Problems wurde ein Programm kreiert, das alle laut $T_{Abspiel}$ anfallenden Überführungswege vor Erstellung der Trajektorie berechnet und in einer Matrix namens *MLaenge* der Trajektorienberechnungen jederzeit zur Verfügung stellt. Aus diesen Längen, welche in *Laenge_ Ueberf.m* bestimmt werden, kann dann wieder wie in 9.17 die Überführungszeit der folgenden Überführung berechnet und von der Streichzeit abgezogen werden, sofern die Streichzeit größer als die Überführungszeit ist.

Notwendigkeit einer Überführung Überführungen sind im Eigentlichen für Vorgänge vorgesehen, bei denen der Bogen einen Saitenwechsel vollziehen muss. Wenn nun aber die Möglichkeit besteht, aufeinander folgende Noten auf ein und derselben Saite zu spielen, wäre eine Überführung nach jeder dieser Noten unsinnig. Wenn darüber hinaus auch noch sehr schnelle Noten gespielt werden sollen (z. B. 32stel), was mit einer entsprechend kurzen Streichzeit einhergehen würde, könnte die Streichzeit nicht mehr um die Überführungszeit reduziert werden, was wie schon erwähnt Verfälschungen im Takt und ein allgemein schlechtes Klangbild zur Folge hätte.

Um diesen Missstand zu umgehen, sollte der Liedgenerator in der Lage sein, selbstständig überflüssige Überführungsaktionen zu erkennen und die Trajektorie demensprechend anzupassen. Um dies zu bewerkstelligen, ist ein Umbau der Algorithmen zur Kreierung der Matrix der anzuspielen Punkte T_{Matrix} von Nöten. Der jetzige Aufbau basiert auf den Gedanken, dass sich nach *jeder* Streichaktion eine Überführung anschließt. Hierfür werden in der besagten Matrix für den Beginn sowie dem Ende einer jeden Streichaktion jeweils in zwei aufeinander folgenden Spalten Lage und Orientierung als Vektor abgespeichert. Zusätzlich wird in jeder Spalte angegeben, ob sich ein Streichvorgang oder eine Überführung anschließt. Nach dem von *Haase* beschrieben Schema, sind die Werte für die folgende Aktion somit alternierend zugeordnet, da sich Streich- und Überführungsvorgang einander abwechseln. Dies geschieht auf folgender Weise: In *Matrizenberechnungen* werden zu jeder Note (=Streichvorgang) zwei Spalten generiert, welche jeweils Anfangs- und Endpunkt inne halten. Um zwei Streichvorgänge ohne Überführung generieren zu können, ist es notwendig, zwei mal hintereinander linear interpolieren zu können. Da der Endpunkt des ersteren Streichvorganges mit dem Anfangspunkt des folgenden zusammenfällt, ist es nun nicht mehr nötig, bzw. währe es sogar fatal den Endpunkt des ersten Streichvorganges als eigenen Lagevektor in eine eigene Spalte von T_{Matrix} zu schreiben, da in diesen Falle zwei lagegleiche Punkte aufeinanderfolgend in der Matrix generiert wären. Will man eine Überführung vermeiden, so müsste demensprechend zwischen zwei Punkten mit der euklidischen Länge von Null linear interpoliert werden, was natürlich nicht möglich ist. Aus diesem Grunde wird in der modifizierten Datei *Matrizenberechnungen_ II* für Streichvorgänge an denen sich keine Überführung anschließt nur der Anfangspunkt generiert, es sei denn, es wird genau an besagter Stelle eine Pausen-

note gewünscht. In dem Falle werden die zwei lagegleichen Vektoren generiert, zwischen denen dann allerdings nicht linear, sondern mit Hilfe der Beziér-Funktion interpoliert wird (siehe Tabelle 9.3 und Programmcode *Matrizenberechnungen_II*).

Zu klären bleibt jetzt noch die Frage, in wie fern der Liedgenerator eine Unterscheidung zwischen einer Überführung oder dem Auslassen dieser trifft: Aus der zu spielenden Tonart ist eindeutig auch die zu spielende Saite festgelegt[7]. Sind aufeinanderfolgend zwei Töne auf der gleichen Saite zu spielen, so wird der Liedgenerator prüfen, ob ein Auslassen einer Überführung möglich wäre. Hierfür wird als erstes geprüft, ob der Streichvorgang der zweiten Note beginnend vom Endpunkt der ersten in umgekehrter Richtung zulässig ist, d. h. ob der Bogenbereich für die geforderte zurückzulegende Strecke nicht überschritten wird. Nach dem Scheitern einer solchen Überprüfung wird selbiges in gleichbleibender Richtung getestet. Ist der Streichvorgang möglich und keine Pausennote explizit vorgesehen, so wird in der Matrix $T_{Abspiel}$ in der Spalte der zweiten Note vermerkt, dass eine vorherige Überführung nicht "erwünscht" ist. Diese Berechnungen geschehen alle in Echtzeit bei Eingabe der Noten, wobei die Matrix auch rückwirkend abgeändert werden kann.

Zu erwähnen wäre noch, dass bei der Korrektur einer Note zwar geprüft wird, ob sich diese ohne Überführung an die Vorherige anschließen lässt, die darauffolgende allerdings "entkoppelt" wird, um einen Ketteneffekt der Überführungsmöglichkeiten aller folgenden Noten zu vermeiden. Somit wird sich nach einer korrigierten Note immer eine Überführung anschließen.

9.2.2.3. Anhang: Speichermatrizen

Zeile	$T_{Abspiel}$	Werte
1	zu streichende Saite	*1 - 4*
2	Aufsetzpunkt am Bogen	*in Meter*
3	Absetztpunkt am Bogen	*in Meter*
4	Spielrichtung	1 = ziehen, -1 = schieben
5	Stegentfernung	*in Meter*
6	Note	*z. B. 1/8*
7	Grund-Überf.-Zeit	*<nicht mehr verwendet>*
8	vorherige Überführung?	0 = ja, 1 = nein
9	Pausenzeit	*z. B. 1/8*

Tabelle 9.2.: Speichermatrix $T_{Abspiel}$

[7] Mit einer späteren Implementierung des Greifens der Saiten können durchaus mehre Möglichkeiten zum spielen desselben Tones bestehen. Die Eindeutigkeit währe in diesem Fall zu prüfen bzw. festzulegen.

Zeile	T_{Matrix}	Werte
1	fortlaufende Nummerierung	1...n
2	Aktion	0 = streichen, 1 = überführen
3	Wert der Transf.-Matrix (1,1)	Korrd.-Fragment f. Orientierung
4	Wert der Transf.-Matrix (2,1)	Korrd.-Fragment f. Orientierung
5	Wert der Transf.-Matrix (3,1)	Korrd.-Fragment f. Orientierung
6	Wert der Transf.-Matrix (1,2)	Korrd.-Fragment f. Orientierung
7	Wert der Transf.-Matrix (2,2)	Korrd.-Fragment f. Orientierung
8	Wert der Transf.-Matrix (3,2)	Korrd.-Fragment f. Orientierung
9	Wert der Transf.-Matrix (1,3)	Korrd.-Fragment f. Orientierung
10	Wert der Transf.-Matrix (2,3)	Korrd.-Fragment f. Orientierung
11	Wert der Transf.-Matrix (3,3)	Korrd.-Fragment f. Orientierung
12	Wert der Transf.-Matrix (1,4)	Korrd.-Fragment f. Lage
13	Wert der Transf.-Matrix (2,4)	Korrd.-Fragment f. Lage
14	Wert der Transf.-Matrix (3,4)	Korrd.-Fragment f. Lage
15	Note	z. B. 1/8
16	Pausenzeit	z. B. 1/8
17	Saite	1 - 4
18	Spielrichtung	1 = ziehen, -1 = schieben

Tabelle 9.3.: Speichermatrix T_{Matrix}

9.2.3. Fazit

Der Liedgenerator stellt ein nach bisherigen Beurteilungsmöglichkeiten gelungenes Werkzeug dar, um Trajektorien von ganzen Liedern nach steuerungstechnischen wie auch musikalischen Gesichtspunkten fehlerfrei zu generieren. Zusammen mit den Fehleranalysemöglichkeiten, den Sprunginterpolator und den Trajektorienkorrekturen ist es schon jetzt möglich, sehr gute Trajektorien zu erzeugen.

9.3. Neue Programmstruktur

Aufgrund der im Abschnitt 9.1 und 9.2 getätigten Verifikationen und dem Hinzufügen neuer Programmfragmente wie dem Liedgenerator, der Sprungglättung und der Fehleranalyse, ergibt sich als Folge dessen eine neue Programmstruktur (siehe Abbildung 9.2). Die Struktur der eigentlichen Trajektorienerzeugung wurde dabei kaum angetastet. Es wurden lediglich die alten Programmfragmente durch die verifizierten *Matrizenberechnungen_II* und *Trajektorie_II* ersetzt und entsprechend eingefügt. *Laenge-Uberf.m* berechnet wie schon erwähnt bereits vor den Interpolationen zwischen den Punkten der Matrix der anzufahrenden Punkte T_{Matrix} die Längen aller Überführungen gemäß $T_{Abspiel}$ und stellt diese der Funktion *Trajektorie_II.m* zur Erzeugung variabler Streichvorgänge zur Verfügung. Nach den Interpolationen schließt sich nunmehr eine Fehleranalyse an, welche die Funktion *Sprung_Glaettung* mit Daten Füttert. Eine erneute Fehleranaly-

se überprüft die endgültige Trajektorie nochmals nach Fehlern und Unstetigkeiten und gibt diese aus. Die Daten aus *Sprung_ Glaettung.m* werden an den Liedgenerator übermittelt, welcher diese in *Traj_ Vektor* umwandelt und für eine Übertragung in Simulink bereitstellt.

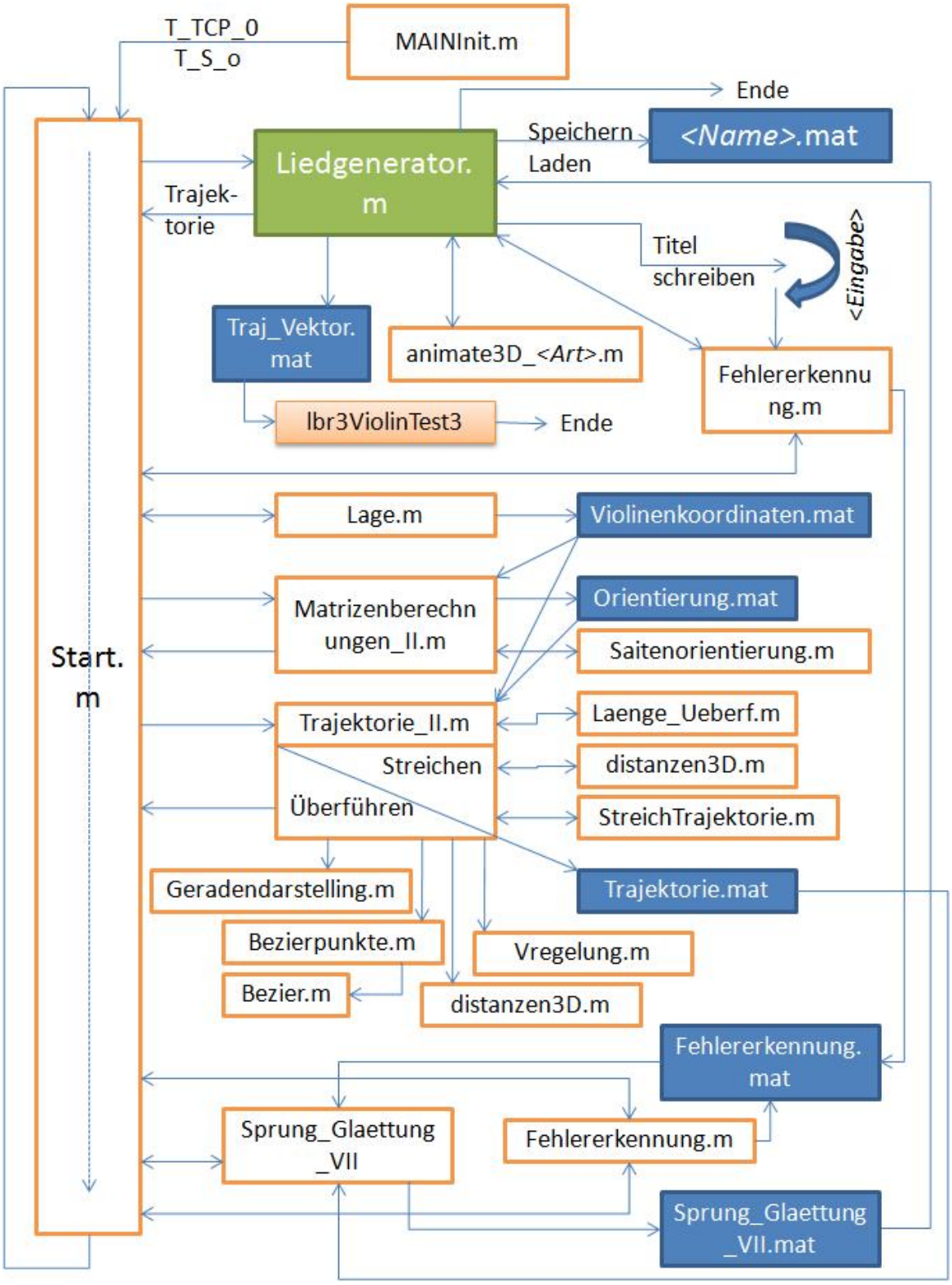


Abbildung 9.2.: Ablaufplan

10. Entwicklung echtzeitfähiger Simulink-Oberflächen

Alle in diesem Bericht bisher beschriebenen Berechnungen und Vorgänge liegen in Form einer textuellen Programmierung in Matlab vor. Um mit den berechneten Trajektoriendaten nun letztlich den Roboter ansteuern zu können, bedarf es der Erstellung von sogenannten Simulink-Modellen, da die bereits vorhandene Schnittstelle zum Roboter auf einer solchen Simulink-Oberfläche vorliegt, auf der u. a. das Robotermodell implementiert wurde. Auch der Einbau einer Inversen-Kinematik und anderer spezifischer Programme lassen sich unter dem Gesichtspunkt der Echtzeitfähigkeit am besten unter einer solchen Oberflächen erstellen. Zu guter letzt eignen sich solche Modelle auch hervorragend zum Testen und Simulieren der bereits vorhandenen Daten. Wurde ein lauffähiges Simulink zur Ansteuerung des Manipulators erfolgreich erstell, so ist anschließend eine Kompilierung und Übertragung auf das Echtzeitsystem, hier QNX, nötig. Nach der Kompilierung sind an diesem Modell dann keine Änderungen mehr möglich. Allerdings besteht die Möglichkeit mit einem zweiten Modell dieses zu Steuern, was in unserem Falle auch realisiert werden wird.

10.1. Testversion I: Übertragungsverhalten und Simulation

In der ersten Version zur Erstellung der Simulink-Oberfläche sollte vor allem das Übertragungsverhalten der Trajektoriendaten aus den Matrizen T_{Traj} und R_{Traj} analysiert werden, sowie der Einbau einer Inversen-Kinematik. Zusätzlich sollte auch noch das Roboter-Modell sowie ein vorgeschalteter Impedanzregler implementiert werden, was vor allem zu Test- und Simulationszwecke vorgesehen war. Die eigentliche Ansteuerung wird letztendlich aber wie vorgesehen mithilfe von Positionsdaten, d. h. mit Gelenkwinkel, welche die Inverse-Kinematik einmal liefern wird, vorgenommen. Alle Daten sowie deren evtl. benötigten Ableitungen oder Integrale können aus einen solchen Modell problemlos entnommen und zur weiteren Überprüfung genutzt werden. Der Aufbau des Modells ist in Abbildung 10.1 dargestellt.

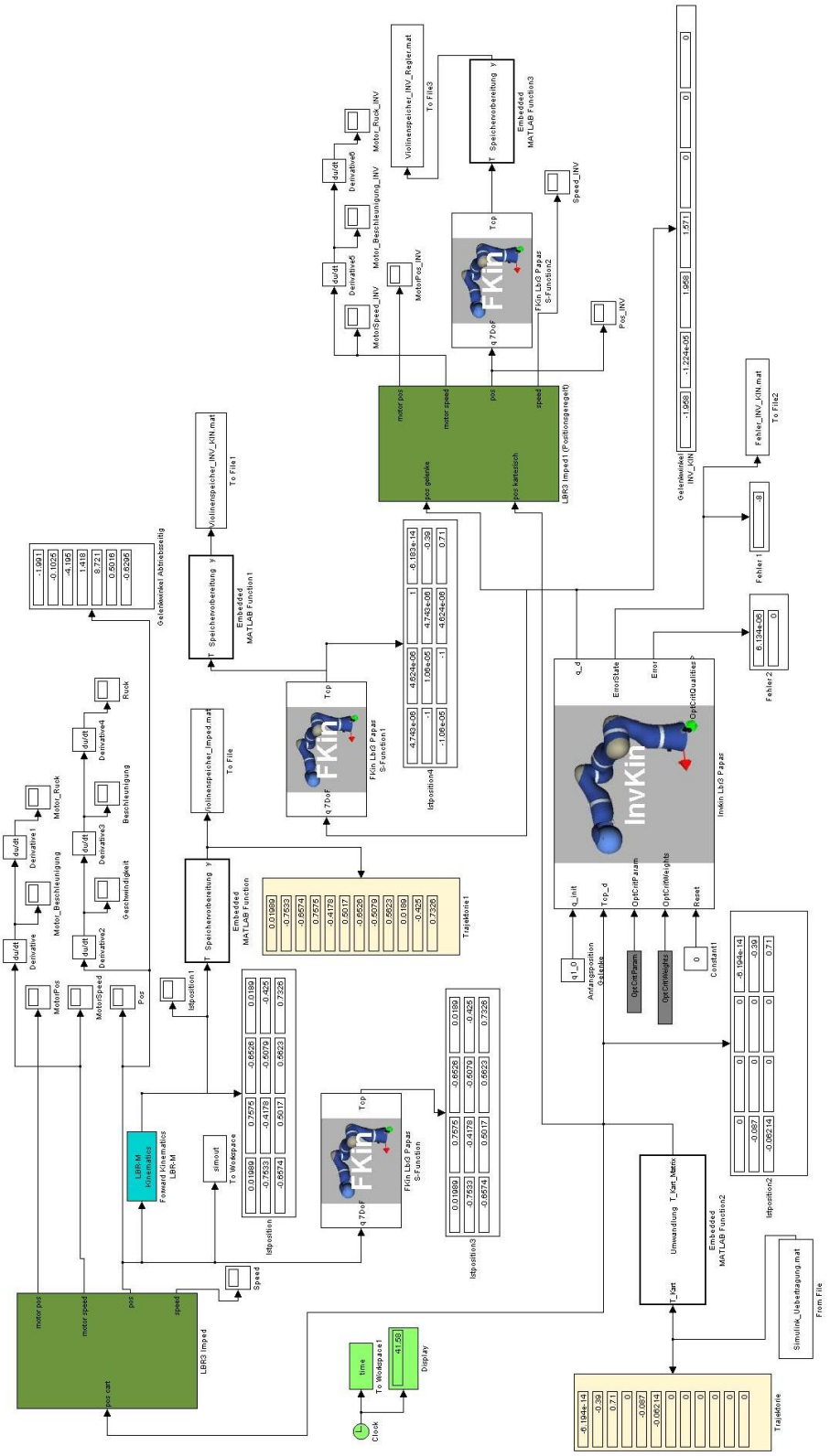


Abbildung 10.1.: Simulink-Modell zur Ansteuerung des Manipulators, Version I

10.1.1. Beschreibung des Modells

Der Block namens *Simulink_ Uebertragung.mat* ganz unten links stellt die Schnittstelle zu Matlab dar. Die Berechneten Trajektoriendaten aus T_{Traj} und R_{Traj} werden hierfür in der Funktion *Simulink_ Uebertragung* für eine Übertragung ins Simulink-Modell geeignet konvertiert und in einer gleichnamigen *Mat*-Datei abgelegt. Lage und Orientierung der Trajektorie liegen in einer einzigen Matrix vor, die zusätzlich eine Spaltennummerierung inne hält. Wird das Simulink-Modell gestartet, so werden die einzelnen Daten Spalte für Spalte, entsprechenden der eingestellten Taktzeit (hier 1000Hz) ausgelesen und zur weiteren Verarbeitung zur Verfügung gestellt. Der mittig unten platzierte große Block mit der Aufschrift *InvKin* beinhaltet die Inverse-Kinematik. Da diese zur Umwandlung der Daten von Welt- in Roboterkoordinaten die kartesischen Koordinaten in einer 3x4-Matrix benötigt[1], ist zuvor noch eine Transformation der Trajektoriendaten von der Form eines Vektors in eine solche Matrix nötig. Diese Aufgabe übernimmt der Block *Umwandlung.*

DIE INVERSE-KINEMATIK: Die in diesem Projekt genutzte Inverse-Kinematik stellt eine instituteigenene Entwicklung dar. Das Verfahren basiert auf eine analytisches Lösungsschema, auf das hier im einzelnen nicht einegenagen werden soll. Die Berechnungen der Gelenkwinkel geschehen unter Einbeziehung verschiedener Aspekte, wie des Nichtdurchfahrens von singulären Stellungen, des Fernhaltens von Gelenkwinkelgrenzen und eine möglichst kleine Gelenkwinkeländerungsrate zum erreichen der geforderten Position. Die Gewichtungen der einzelnen Anforderungen werden dabei durch einzelne Funktionen festgelegt, wobei das erreichen der Postion die niedrigste Priorität besitzt.

Nun kann die Inverse-Kinematik aus den kartesischen Lagedaten die entsprechenden Gelenkwinkel berechnen, so fern diese nicht außerhalb des Arbeitsbereiches liegen oder der Roboter an seine Gelenkwinkelgrenzen stoßen würde. Auch singuläre Stellung führen zu Problemen und sind dementsprechend zu vermeiden. Dargestellt werden die berechneten Gelenkwinkel in dem rechts daneben platzierten Feld namens *Gelenkwinkel INV_ KIN.*

Eine andere Methode zur Ansteuerung des Roboters bietet die Methode der Impedanz-Regelung. Diese berechnet keine Gelenkwinkel. Vielmehr ist sie mit einer Kraftregelung vergleichbar, welche den Roboter durch Kräftevorgaben in die richtige Richtung lenkt, bis die gewünschte Position erreicht wurde. Dabei gilt: Um so größer der positionelle Abstand von Soll- zu Ist-Lage, um so höher die Kraftvorgabe zum Erreichen dieser, ähnlich wie bei einer sich aufspannenden Feder. Mit einen nachgeschalteten Roboter-Modell lassen sich die Trajektoriendaten schon sehr gut testen, auch wenn diese Methode zur Ansteuerung des Roboters im Violine-Projekt aufgrund der hohen Schwingungsanfälligkeit und der zu niedrigen Positioniergeschwindigkeit nicht genutzt werden wird. Andererseits lassen sich durch das schwingende Verhalten Unstetigkeiten oder andere “unschöne” Kurvenzüge in der Trajektorie sehr gut visualisieren. Impedanzregler und Robotermodell wurden beide

[1] Die unterste konstante Zeile wird von dem Inversen-Block hier automatisch kreiert.

in den grünen Block integriert, welcher hier zweimal Verwendung findet. In der "oberen" Variante wird der Impdanzregler direkt mit den transformierten kartesischen Koordinaten gespeist, welcher seinerseits den virtuellen Roboter ansteuert. Die dargestellten abtriebsseitigen Gelenkwinkel werden hierbei nicht berechnet, sondern im Modell "gemessen". Durch eine anschließende Vorwärtstransformation lässt sich die tatsächliche Bewegung des Roboters in Umweltkoordinaten umrechnen und weiter analysieren, was hier mithilfe von Differentiationen in Bezug auf Geschwindigkeit, Beschleunigung und sogar den Ruck getätigt wurde (siehe kleinere Blöcke oben mittig). In der "rechten" Variante wurde der Impedanzregler zu Testzwecken mit einer Positionsregelung, welche ihre Daten von der Inversen-Kinematik bezieht, gekoppelt. Alle berechneten Daten jeder Variante werden, nachdem sie mithilfe einer Vorwärtskinematik in Umweltkoordinaten zurücktransformiert wurden, in einzelnen *Mat*-Dateien gespeichert. Jede dieser Speichereinheiten beginnen mit dem Namen *Violinespeicher* und besitzen folgend eine Kennzeichnung für die Art der erzeugten Daten. Später können diese Daten alle in einer matlabeigenen Simulation visualisiert werden. Eine weitere Datei namens *Fehler_INV_KIN.mat* speichert die von der Inversen-Kinematik ausgegebenen Fehler, um später genaue Angabe über Anzahl und Fehlerort zur genaueren Untersuchung dieser zu besitzen.

Aufgrund der mit der ersten Version des Simulink-Modells möglichen Analysemöglichkeiten, konnte die Trajektorie einer weiteren genaueren Untersuchung unterzogen werden, diesmal auch mit realen Daten aus der Inversen-Kinematik bzw. dem Roboter-Modell. Die Ergebnisse lieferten an verschiedenen Stellen Anlass, erneut kleine Verfeinerungen der Algorithmen vorzunehmen.

10.2. Testversion II: Analyse des Arbeitsraumes und der Gelenkwinkelpositionierung

In der zweiten Version des Modells wurde im Prinzip nur eine abgemagerte Version der ersteren realisiert (siehe Abbildung 10.2). Alle nicht mehr benötigten Blöcke wie dem Impedanzregler wurden unter dem Gesichtspunkt der Rechenleistung entfernt. Auch das Robotermodell wurde aus diesem Grunde vorerst entfernt, welches aber natürlich später wieder Verwendung finden wird. Neu ist die Einbindung eines Simulators namens *RobView*, dessen Schnittstelle ganz rechts in den Block *sfun_move_lwr3* realisiert wurde. Der Simulator ist nicht Teil des Simulink-Modells und muss somit explizit gestartet werden. Folgend wird er seine Daten in Echtzeit aus dem Simulink-Modell beziehen. Im Gegensatz zu den Matlab-Simulationen, bei denen lediglich der TCP mit angeflanschten Bogen zu visualisieren ist, ist es mit dem *RobView* möglich, die genaue Bewegung des Ballfangroboters entsprechend den berechneten Winkeln der Inversen-Kinematik zu analysieren (siehe Abbildungen 10.3). Dies ist besonders wichtig, um eine genaue Vorstellung über die wirkliche Lage des Armes sowie die Ausnutzung des Arbeitsraumes zu bekommen.

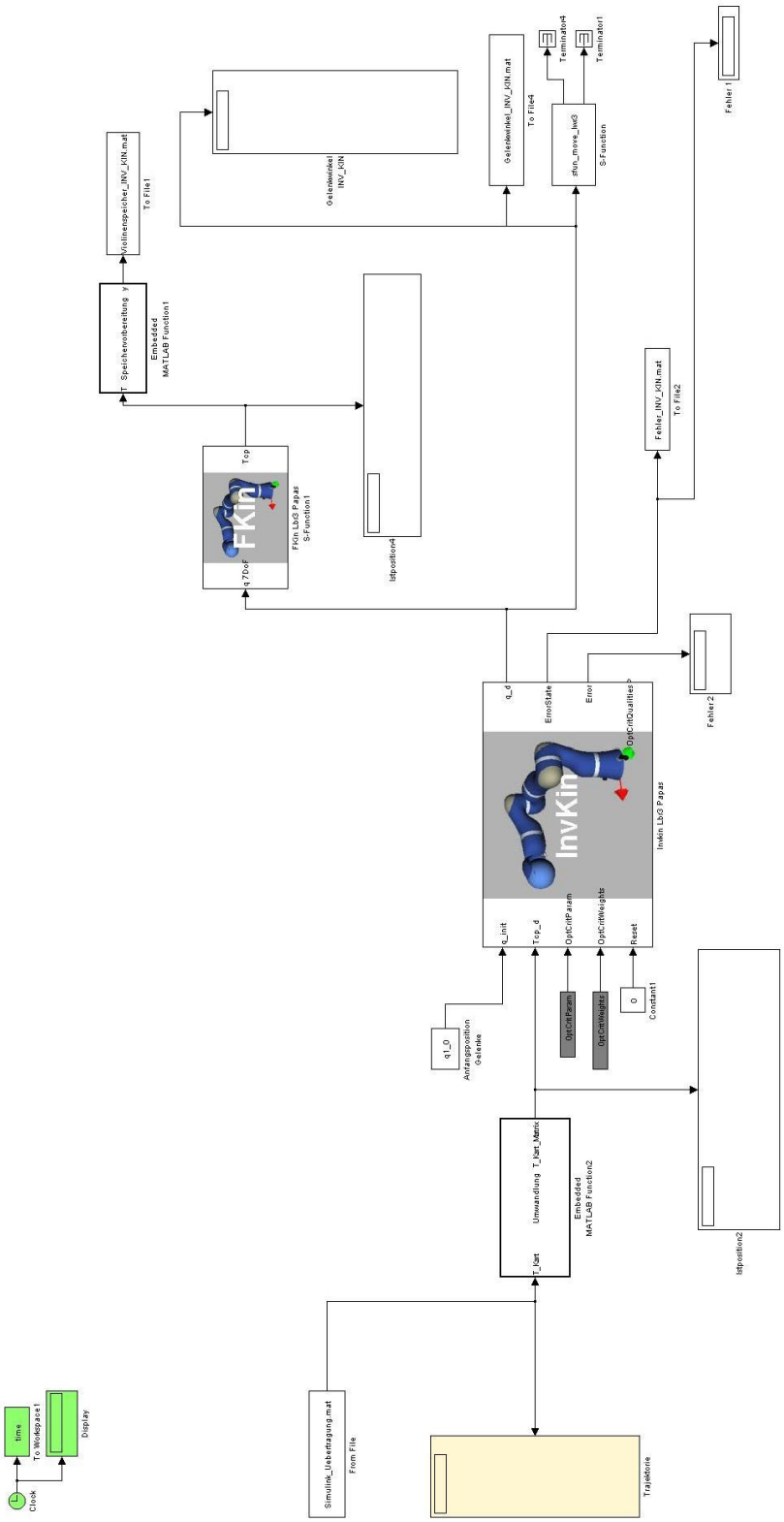


Abbildung 10.2.: Simulink-Modell zur Ansteuerung des Manipulators, Version II

10.2.1. Arbeitsraum

Große Probleme bei der Ansteuerung des Roboters mithilfe der berechneten Trajektoriendaten zeigten sich in der Einhaltung des Arbeitsraumes. Da das Spielen einer Violine einen relativ großen Arbeitsbereich benötigt, ist eine angemessene Platzierung der Violine oder eine Einschränkung des streichbaren Bereiches bereits in der Trajektorienberechnung von Nöten. Letzteres sollte nach Möglichkeit vermieden werden. Eine speziell hierfür erstellte Trajektorie, welche den vollen Arbeitsbereich in wenigen Streichvorgängen in Anspruch nehmen würde, sollte sicherstellen, das der Roboter in der Lage sein wird, alle möglichen Spiellagen für das Spielen beliebiger Titel zu erreichen. Gelangt der Manipulator hingegen an seine Gelenkwinkelgrenzen oder in eine singuläre Stellung, so ist die Inverse-Kinematik in Folge dessen nicht mehr in der Lage, geeignete Gelenkwinkel zu generieren, was zu einem abrupten Abbruch der Bewegung und ein Wiedereinsetzen an undefinierter Stelle zur Folge hätte. Dies stellt eine große Gefahr für den Roboter dar, da zu große Sprünge zur Zerstörung der Antriebselemente führen können. Auch nach optischen Gesichtspunkten sollte die Bewegung möglichst authentisch wirken, was zum Beispiel die Stellung des Ellenbogens betrifft. Mit einer unnatürlichen evtl. für den Menschen sogar undenkbaren Haltung sind zwar auch Streichbewegungen möglich, jedoch nicht wünschenswert. Die Haltung des Armes kann vor allem durch die Anfangsposition, mit welcher die Inverse-Kinematik gespeist wird, beeinflusst werden. Wird z. B. der Ellenbogen in der Anfangsposition auf eine Seite gesetzt, so ist, bedingt durch Eigenschaft der Inversen-Kinematik möglichst kleine Gelenkwinkeländerungen zu vollziehen, ein Umschwenken auf die andere Seite unwahrscheinlich. Bei Bedarf sollte hierbei allerdings in Betracht gezogen werden, die Inverse-Kinematik mit einem oder mehreren Referenzpunkten zur Einhaltung der richtigen Stellung zu versehen. Die im Programm definierte Position der Violine wurde unter dem Gesichtspunkt einer möglichst natürlichen Lage auf iterativem Wege festgelegt und erfüllt alle Bedingungen der Arbeitsraumeinhaltung. Bei einer Veränderung der Position wäre eine erneute Überprüfung, z. B. mit der bereits weiter oben erwähnten speziellen Trajektorie, nötig, um fehlerhafte Rückwärtstransformationen zu vermeiden.

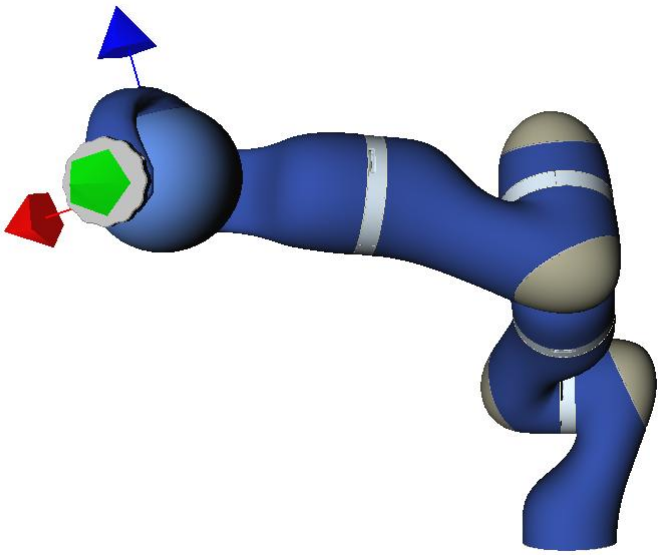
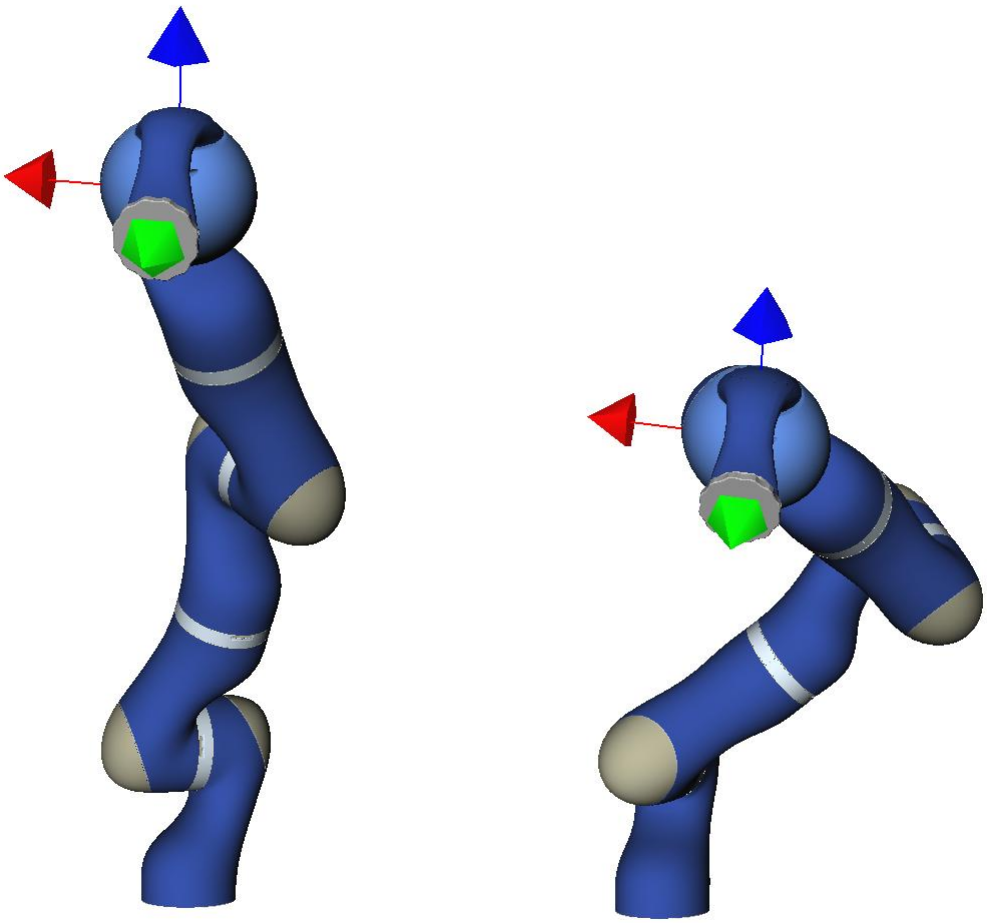


Abbildung 10.3.: Simulation der Bogenführung mit *RobView*



10.3. Das Echtzeitmodell: Entwicklung eines lauffähigen Modells für eine Übertragung auf das Echtzeitsystem QNX

Um den Roboter nun letztlich ansteuern zu können, bedarf es der Erstellung einer Simulink-Oberfläche, welche sich auf das hier verwendete Echtzeitbetriebssystem QNX kompilieren über übertragen lässt, um dort alle nötigen Aufgaben zu tätigen. Die Aufgabe der Kompilierung, Übertragung und der Einbindung benötigter Pfade wird von *RT-LAB* übernommen. Wie schon erwähnt, wird das Modell dabei in zwei "Seiten" aufgeteilt: Die statische Seite, in welcher das Robotermodell und das eigentliche Programm zur Ansteuerung vorliegt, ist nach der Kompilierung und Übertragung auf QNX nicht mehr beeinflussbar. Alle Steuerungsbefehle müssen dementsprechend von der zweiten Seite, der Steuerungsseite getätigt werden. Auch evtl. benötigte Daten sind auf die Steuerungsseite zu überführen, sofern sie während der Laufzeit ersichtlich sein sollen. In Abbildung 10.4 ist die oberste Maske des Gesamtmodells ersichtlich, wobei der linke Block die statische und der rechte die Steuerungsseite inne hält. Die Abbildungen 10.5 bis 10.7 zeigen die statische Seite während die Abbildungen 10.8 und 10.9 die Steuerungsseite darstellen.

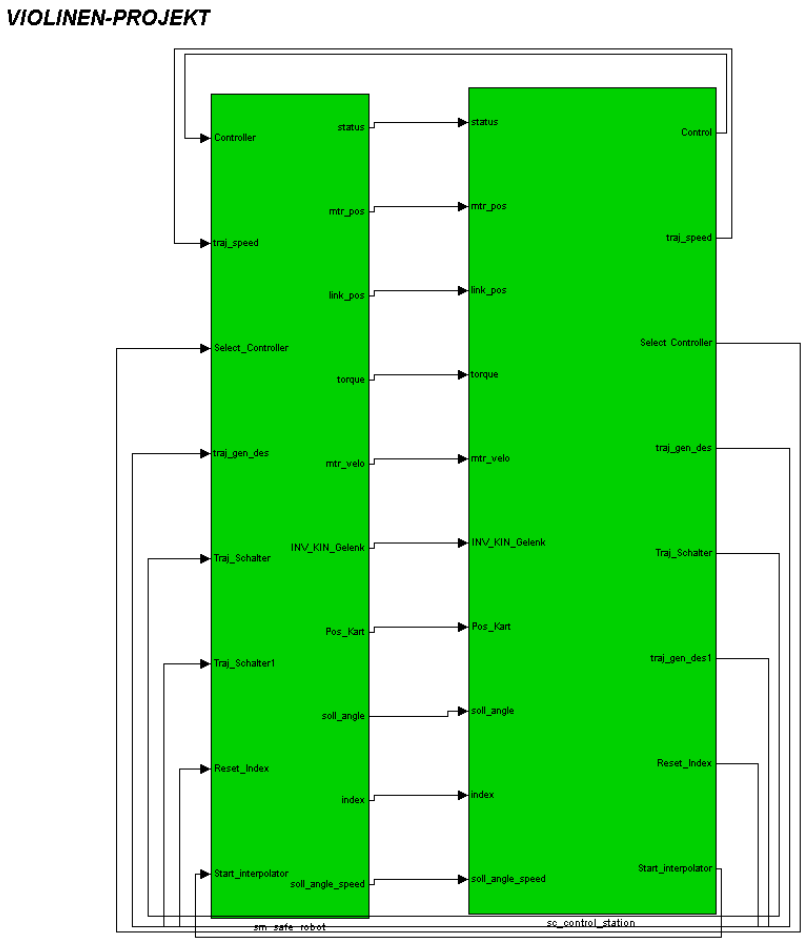


Abbildung 10.4.: obere Maske des Gesamtechtzeitmodells

10.3.1. Statische Seite

Hauptaufgabe der statischen Seite des Modells ist es, wie schon erwähnt, den Roboter anzusteuern bzw. mit Daten zu füttern. Um diese Aufgabe zu bewältigen ist es erforderlich, ein so genanntes Roboter-Modell zu implementieren. In Abbildung 10.5 ist dieses oben in Form eines roten Blocks namens LBR-III erkennbar. Das Robotermodell stellt eine vorgefertigte Steuerungseinheit dar, auf hier nicht näher eingegangen werden soll. Für eine Positionssteuerung ist es lediglich von Nöten, dem Modell die anzufahrenden Gelenkwinkelstellungen in *Rad* vorzugeben. Die Einspeisungsstelle befindet sich hierbei am LBR-III-Block (rot), Eingang *angle.* Da für eine Lagesteuerung keine Soll-Geschwindigkeit (speed) sowie kein Soll-Drehmoment (torque) erforderlich ist, werden diese Werte auf Null definiert. Der Eingang *Control* ist in diesem Modell außer Funktion. Alle anderen Eingänge dienen der Übergabe der Parameter der Regler des Manipulators. Diese Parameter werden im *Gain_ Provider3* generiert und entsprechend der Ansteuerungsme-

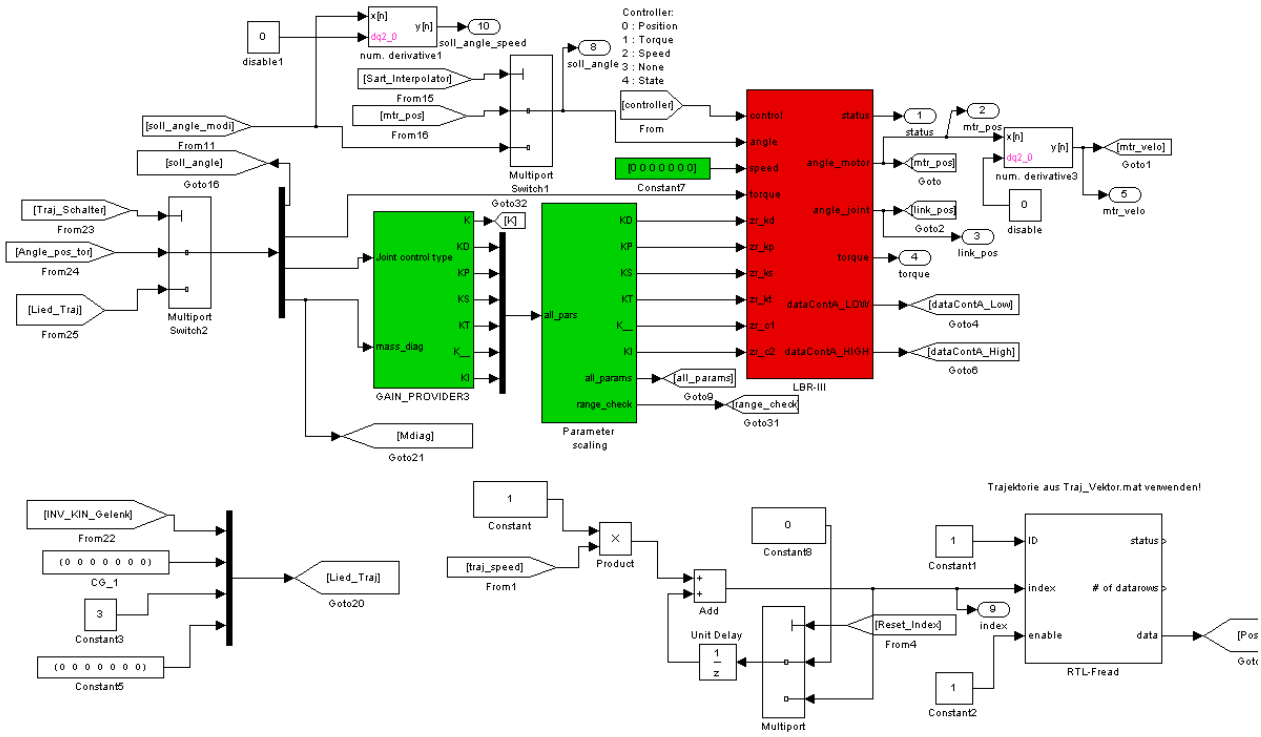


Abbildung 10.5.: statisches Seite des Echtzeitmodells (Teil I)

thode an einem Skalierer übergeben. Der Wert *3* am Eingang *Joint control type* setz die Regelparameter des Gain-Provider hierbei auf Positionsregelung.

Generierung der Gelenkkoordinaten

Eine weitere wichtige Aufgabe des Modells impliziert das Berechnen der Gelenkwinkelkoordinaten aus den kartesischen Trajektoriendaten und eine Übergabe an das Modell zum richtigen Zeitpunkt. Da die gesamte Trajektorie bisweilen im Projekt im Voraus berechnet und gespeichert wird, muss sie komplett an das Modell übergeben und für eine Verarbeitung der Inversen-Kinematik geeignet zur Verfügung gestellt werden. Um diese Aufgabe zu bewältigen wird die Trajektorienmatrix, welche bisweilen in einer matlabeigenen Datei vorliegt, ins ASCII-Format umgewandelt, gedreht und erneut gespeichert. Anschließend wird die Matrix mit einer ID versehen und ins RT-LAB geladen und somit für ein Auslesen der einzelnen Koordinaten durch *RTL-Fread* bereitgestellt. RTL-Fread liest in Folge dessen aus der geladenen Matrix entsprechend der am Block angegebenen ID die durch den Wert *Index* festgelegte Spalte und übergibt diese an den Ausgang *data.* Umgewandelt in eine 3x4-Transformationsmatrix können aus diesen Daten mithilfe der Inversen-Kinematik die entsprechenden Gelenkkoordinaten berechnet bzw. angenähert werden (siehe Abbildung 10.6). Diese Koordinaten, gespeichert

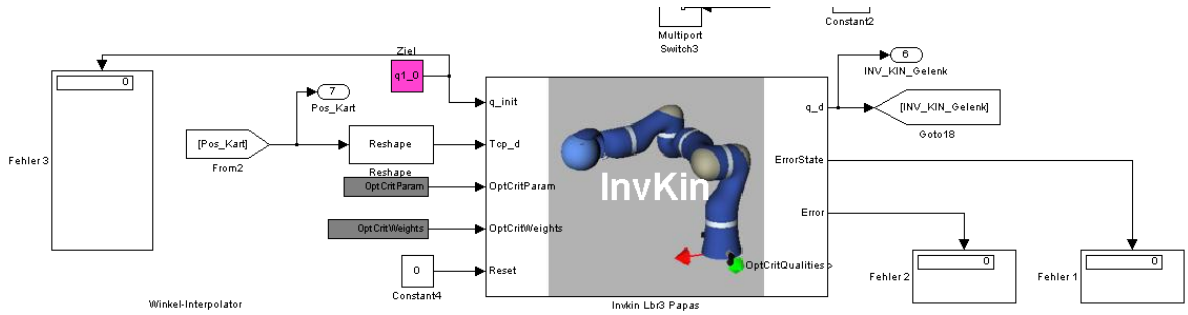


Abbildung 10.6.: statisches Seite des Echtzeitmodells (Teil II)

in *INV_KIN_Gelenk,* werden mit den für den Gain-Provider benötigten Daten zusammengeführt und an *Lied_Traj* übergeben. Im Folgenden können diese Daten nun als Eingangskoordinaten für den LBR-III durch eine entsprechende Umschaltung des *Multiport Switch2* durch *Traj_Schalter* übergeben werden. Zu beachten sei hierbei allerdings, das dies nur möglich ist, wenn die momentane Lage des Manipulators exakt mit der virtuellen Startposition des Trajektoriengenerators zusammenfällt, welche seine Koordinaten beginnend von dieser generiert. Ist dies nicht der Fall, muss entweder die Trajektorie an die aktuelle Position angepasst werden, was meist durch eine Neuberechnung der gesamten Bahn mit hohem Aufwand verbunden ist, oder, wie in diesem Falle realisiert, die im Programm vorgegebene Position muss vom Manipulator vor Beginn der Trajektorienfahrt angefahren werden. Eine einfache aber effektive Lösung dieses Problems stellt die Implementierung eines Winkelinterpolators dar, welcher, steuerbar durch eine Vorgabe der max. Winkelgeschwindigkeit, direkt zwischen Start und Zielposition in Gelenkkoordinaten inkrementell interpoliert. Diese Aufgabe wird vom Block gleichen Namens realisiert (siehe Abbildung 10.7 oben). Als Eingangsdaten dienen natürlich die aktuelle Position des Roboters, sowie die Zielposition entsprechend der Intitialposition des Trajektoriengenerators. Letzteres wird auf der Steuerungsseite definiert und als *traj_gen* übertragen. Auch diese Daten, in denen bereits die Regelparameter implementiert wurden, können nun durch den besagten Multiport-Switch 2 als Soll-Winkel ans Modell übergeben werden. Die Umschaltung durch *Traj_Schalter* geschieht auf der Steuerungsseite. Wie leicht zu erkennen ist, werden diese Daten allerdings nicht direkt ans Robotermodell übergeben, sondern in *soll_angle* gespeist und an *soll_angle_modi* zurückgeführt. Dazwischen befindet sich ein weiterer Winkelinterpolator selben Typs, erkennbar in Abbildung 10.7 rechts. Dieser besitzt keine funktionale Aufgabe, sondern dient lediglich als Sicherung, falls unvorhersehbare Sprünge der Gelenkkoordinaten auftreten, die max. Geschwindigkeit überschritten wird oder die Steuerungsseite falsch bedient wurde. Dies erscheint sehr sinnvoll, da wie bereits erwähnt gerade Sprünge zu einer Zerstörung der hochentwickelten Maschine führen können. Möglich wird eine solche Sicherung durch die spezielle Funktionsweise des Interpolators: Wird in einem Takt die maximale Winkeldifferenz und somit

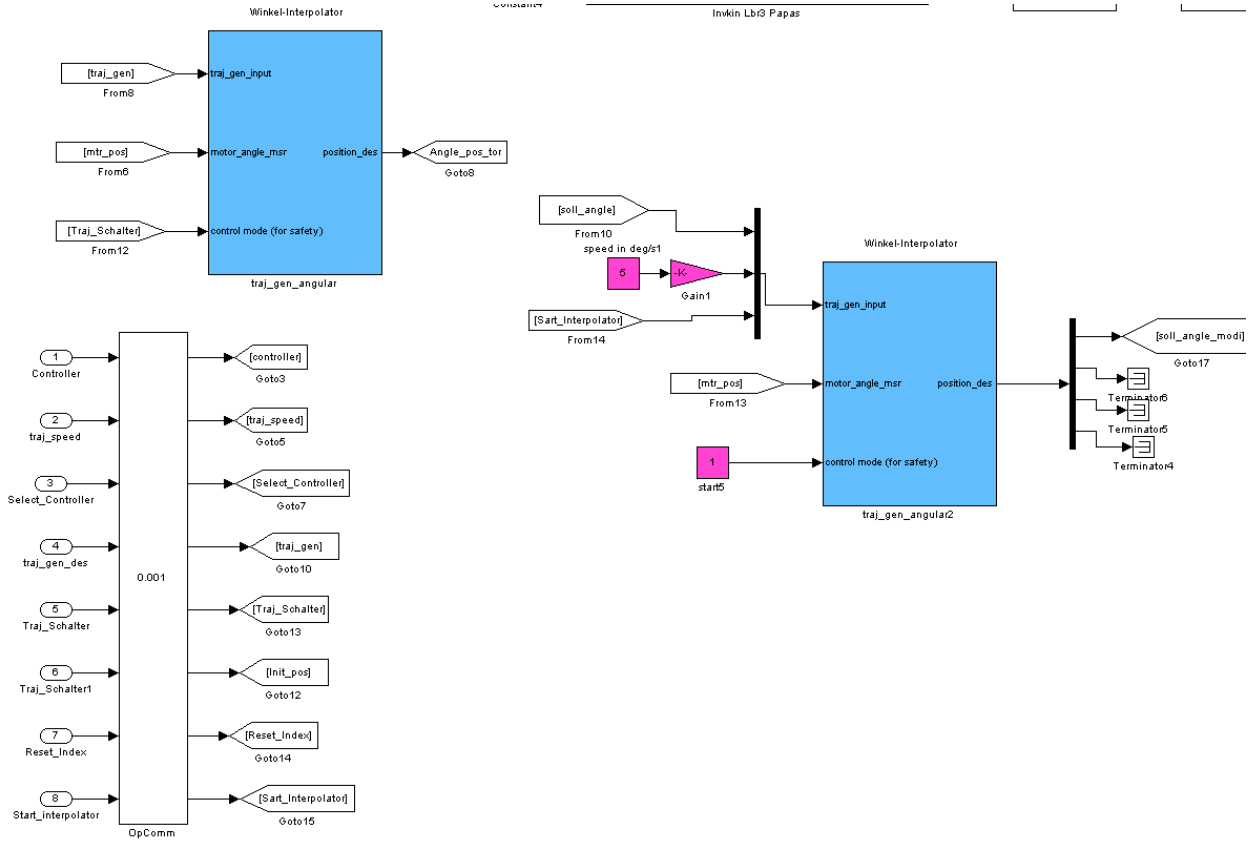


Abbildung 10.7.: statisches Seite des Echtzeitmodells (Teil III)

die als Maximum vorgegebene Winkelgeschwindigkeit überschritten, so wird zwischen diesen beiden Winkelkoordinaten geeignet linear interpoliert, wobei die Maximaldifferenz ausgenützt wird. Dies bedeutet im Folgeschluss natürlich, das der Manipulator der geforderten Position etwas "nachhinkt", und dies genau so lange, bis er durch Ausnützung der maximalen Winkelgeschwindigkeit die virtuelle Zielposition wieder erreichen kann. Demzufolge sollte dieser Interpolator bei dem eigentlichen Geige-spielen möglichst nicht aktiv werden, da dies unweigerlich eine Verfälschung der Bahnkurve mit sich ziehen würde.

10.3.2. Steuerungsseite

Da nach der Kompilierung und dem Laden des Modells keine Beeinflussung der statischen Seite mehr möglich ist, ist es notwendig, ein zweites Modell einzurichten, welches in der Lage sein wird, während der Laufzeit den Programmablauf manuell zu beeinflussen. Zusätzlich sollen in diesem Modell benötigte Daten während der Verarbeitung sichtbar

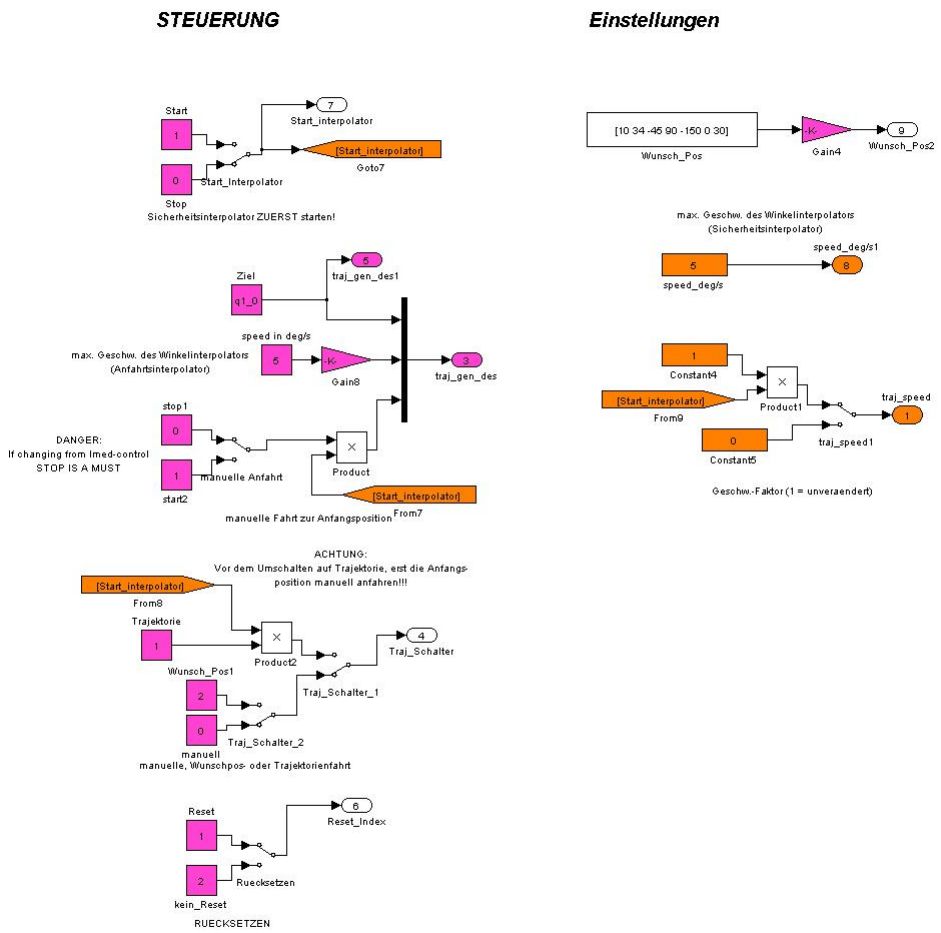


Abbildung 10.8.: Steuerungsseite des Echtzeitmodells (Teil I)

gemacht werden. In 10.8 ist der erste Teil des Modells ersichtlich. Zu erkennen sind vor allem mehrere manuell bedienbare Schalter, mit denen der Programmablauf beeinflussbar sein wird. Die wichtigste Aufgabe wird hierbei von den Schaltern *Traj_Schalter_1* und *Traj_Schalter_2* übernommen. Mit diesen ist es möglich zwischen der Anfahrt zur Initialposition, welche vom Winkelinterpolator übernommen wird, der Anfahrt einer Wunschposition, welche durch die Konstante *Wunsch_Pos* in Gelenkwinkeln (Grad) bestimmbar ist, und der eigentlichen Trajektorienfahrt zu wählen. Besonderes Augenmerk ist dabei darauf zu legen, des die Trajektorienfahrt erst gestartet werden darf, wenn die in der Trajektorienberechnung angegebene Initialposition vom Roboter angefahren wurde. Die andererseits entstehenden Sprünge würden zwar in dieser Version des Modells vom Winkelsicherheitsinterpolator abgefangen, dennoch ist eine solche Reaktion wenig wünschenswert und würde auch unter Umständen zu einer Verfälschung der Bahnkurve

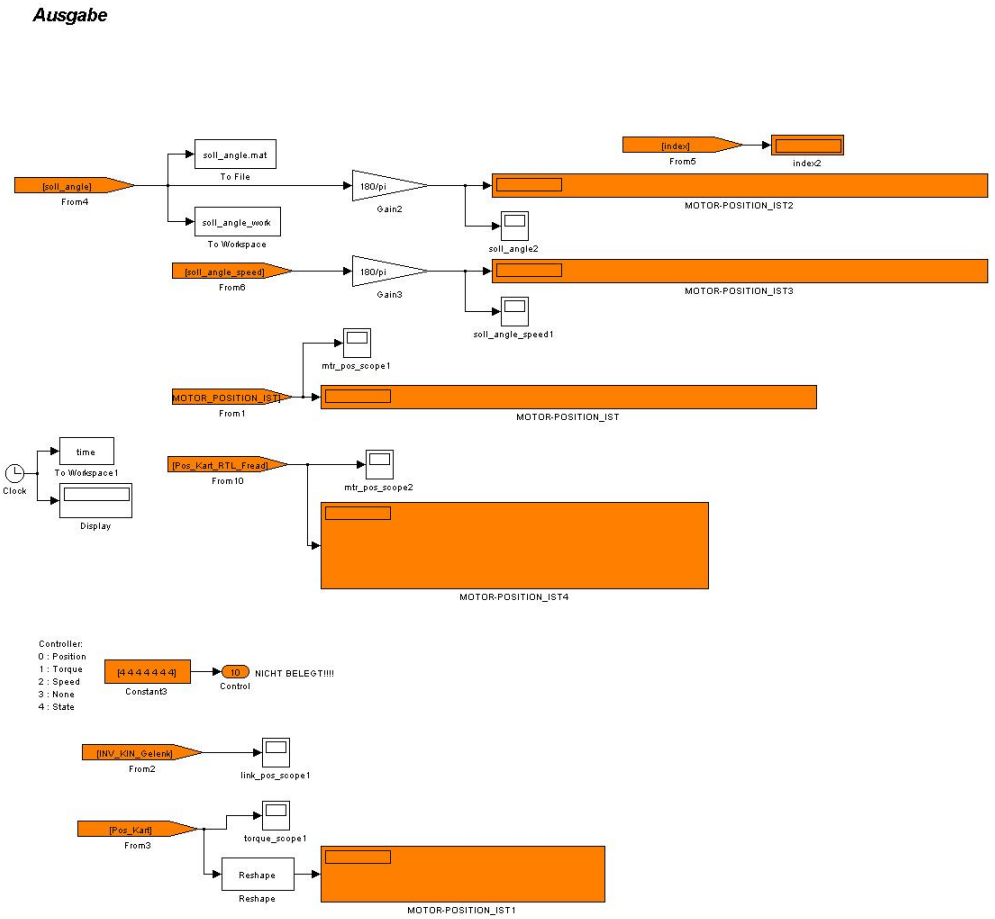


Abbildung 10.9.: Steuerungsseite des Echtzeitmodells (Teil II)

führen. Weiterhin ist noch zu beachten, dass bei einer Anfahrt der Wunschposition die Geschwindigkeit bisweilen lediglich durch den Sicherheitsinterpolator begrenzt wird. Befindet sich die Schalter *Traj_Schalter* auf manuelle Anfahrt, so ist diese noch explizit mithilfe vom Schalter *manuelle Anfahrt* zu starten. Auch die Geschwindigkeit in Grad pro Sekunde und die Zielposition sind hier einstellbar. Wurde die Anfahrt zur Initposition abgeschlossen, so ist eine Umschaltung auf die Trajektorienfahrt mit *Traj_Schalter_1* möglich. Die Geschwindigkeit, mit welcher die übergebene Trajektorie von *RTL-Fread* ausgelesen werden soll, ist durch *traj_speed* bestimmbar. Übergeben wird hierbei ein Faktor, welcher mit einem Grundwert (statische Seite) multipliziert wird. Das Ergebnis aus beiden gibt an, mit welchen Zeilenabstand aus der Trajektorie gelesen wird. Ist das Ergebnis eins, so wird die Kurve Zeile für Zeile ausgelesen und an die Inverse-Kinematik übergeben. Ist das Ergebnis größer eins, so werden nur entsprechende Zeilen ausgelesen, wobei die restlichen Daten verloren gehen. Bei einem Ergebnis kleiner eins wird zwischen den Zeilen interpoliert. Ein diskretes Integrierglied setzt hierfür den In-

dex, welcher der Zeile der auszulesenden Matrix entspricht, auf den benötigten Wert. Rücksetzbar ist dieser Index mithilfe des Schalters *Ruecksetzen,* was allerdings auch hier währen der Fahrt nicht empfehlenswert ist. Mit diesen vier Schaltern wären nunmehr alle nötigen Funktionen realisiert. Durch mehre Testversuche[2], in denen die Bewegung und deren Ableitungen während der Laufzeit und die Trajektorie anschließend durch geeignete Programme analysiert wurden, konnte hingegen gezeigt werden, das zu Beginn der Laufzeit, direkt nach dem Starten des Modells durch RT-Lab, große undefinierte Sprünge sich ereigneten, welche nicht vom Sicherheitsinterpolator abgefangen wurden. Dies stellte natürlich eine große und nicht hinnehmbare Gefahr für das System dar. Dass diese fehlerhaften Positionsorderungen nicht vom Winkelinterpolator abgefangen wurden und dieser selbst die letzte Instanz vor der direkten Übergabe der Gelenkwinkel ans Roboter-Modell darstellt, weißt darauf hin, dass die fehlerhaften Daten vom Interpolator selbst erzeugt wurden. Tatsächlich muss sich der Interpolator nach dem Start des Modells selber erst auf die aktuell gemessene Position der Antriebselemente Initialisieren. Somit ist eine direkte Nutzung des Interpolators von Anfang an nicht möglich. Das Problem soll nun damit gelöst werden, dass der Interpolator erst nach dem Starten des Modells manuell zugeschaltet wird. In der Zeit davor soll lediglich die gerade gemessene Position der Antriebselemente als Sollvorgabe dienen. Zu diesem Zwecke wurde im statischen Modell ein weiterer "Multiswitchport" eingebettet, welcher durch den Schalter *Start_ Interpolator* angesteuert wird. Wird vom Schalter der Wert Null übergeben, so wird lediglich die aktuelle Position geordert. Bei einer Umschaltung auf eins werden wieder die eigentlichen Gelenkwinkelsollvorgaben an das Robotermodell übergeben. Ein Rückschalten auf Null wird nicht empfohlen, da in diesem Falle der Sicherheitsinterpolator inaktiv wäre und keine Tests für einen solchen Vorgang vorliegen. Zur weiteren Sicherheit, und vor allem um einer falscher Bedienung vorzubeugen, wurde der Schalter *Start_ Interpolator* mit den meisten anderen Schaltern des Steuerungsmodell verknüpft, welche in Folge dessen deaktiviert bleiben, solange der Wert nicht auf eins gesetzt wurde. Trotz der hier realisierten Sicherheitsmechanismen sollte von der für diese Steuerung vorgesehene Befehlskette nicht abgewichen werden. Die Befehlskette ist in Abbildung 10.10 ersichtlich. Testdurchgänge mit dem modifizierten Modell erwiesen sich als wesentlich erfolgreicher als zuvor und konnten nun auch erstmals mit aktiven Antriebselementen von statten gehen. Trotz einer leichten Unruhe, vorzugsweise in langsamen Bewegungen ersichtlich, konnte sich das Ergebnis schon zeigen. Getestet wurde allerdings noch ohne Bogen und Violine und mit einer "Ein-Noten-Trajektorie". Diese implizierte die An- sowie Abfahrt des Bogens aus bzw. zur Initialposition und einen Streichvorgang.

[2] Für die erste Versuche wurde der Roboter mit nichtaktiven Motoren angesteuert um eine Beschädigung der Antriebselemente bei fehlerhafter Ansteuerung zu vermeiden.

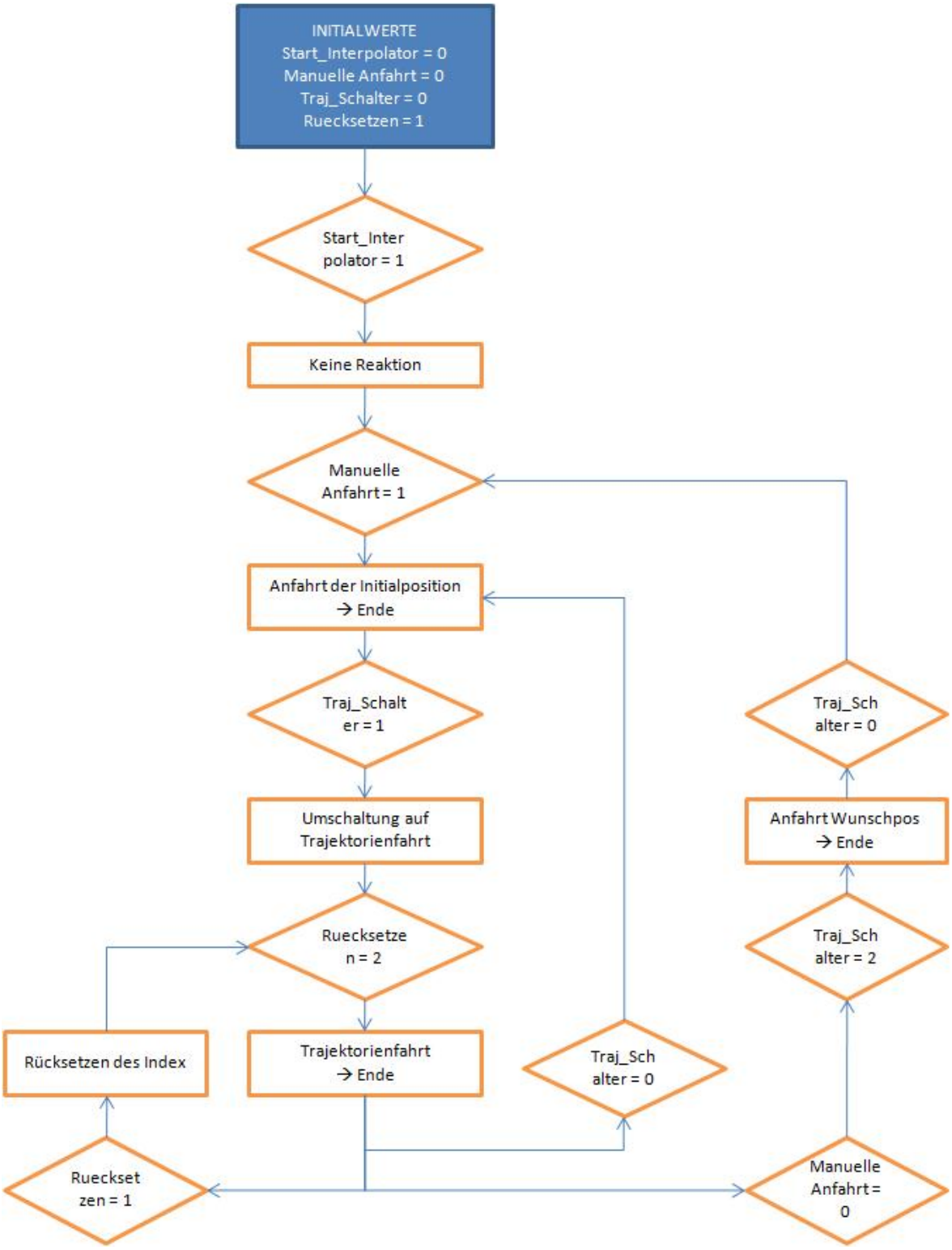


Abbildung 10.10.: Befehlskette

10.4. Inkrementelle Trajektoriengenerierung

Ein weiteres großes Gebiet des Violine-Projektes stellt die Erstellung eines Programmes zur inkrementellen Trajektoriengenerierung dar. Mithilfe solcher Algorithmen, soll es möglich werden, die Trajektorie während der Laufzeit in Echtzeit berechnen zu können, wobei lediglich eine abzuspielende Matrix als Informationsträger an das Modell vor Beginn der Berechnungen übergeben werden soll. Großer Vorteil einer solchen Trajektoriengenerierung stellt die geringe Größe an benötigten Speicherplatz dar, welcher bei einer Offline-Berechnung bei langen Stücken erhebliche Ausmaße annehmen kann (vgl. 7.1).

10.4.1. Idee

Für die Realisierung eines solchen Programms soll hier ein Spagat zwischen einer Modellerstellung in Simulink und einer textuellen Programmierung in Matlab erfolgen. Das Simulink-Modell wird dabei als übergeordnete Maske fungieren und die grundlegende Steuerung der Trajektoriengenerierung aufbauend auf der abzuspielenden Matrix übernehmen, während die textuelle Programmierung, angelehnt an die bereits vorhandenen Algorithmen, die eigentlichen einzelnen Punkte berechnen wird. Die informationstragende abzuspielende Matrix wird auch hier mithilfe des Liedgenerators erstellt, welcher die Berechnungen bis zur endgültigen Bestimmung der anzufahrenden Punkte vornimmt (vgl. 9.2).

10.4.2. Modelle

10.4.2.1. Oberste Modell-Maske

In Abbildung 10.11 ist die bis zu diesem Zeitpunkt fertiggestellte oberste Maske des gesamten Modells ersichtlich.

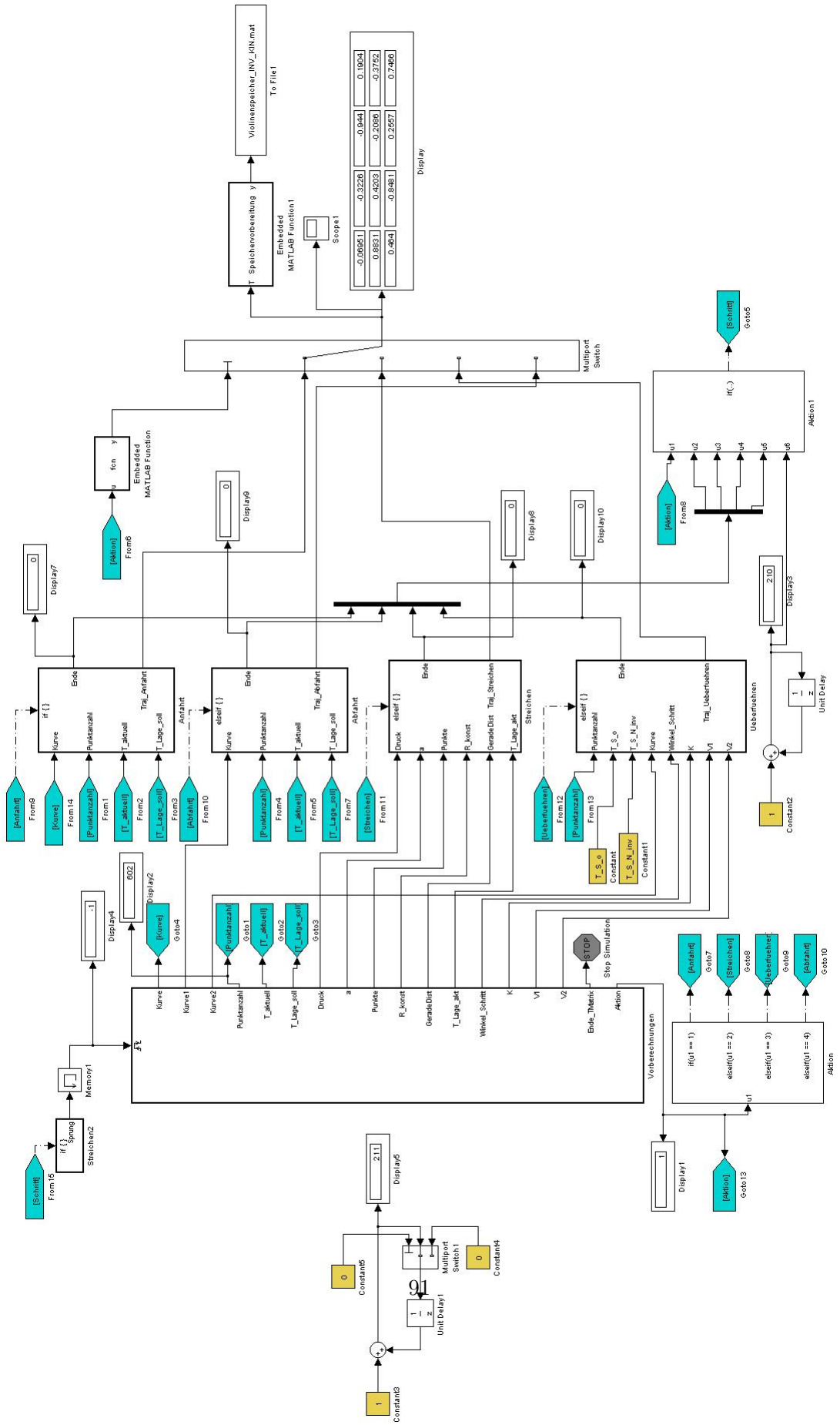


Abbildung 10.11.: Oberste Maske der inkrementellen Trajektoriengenerierung

Hauptbestandteil dieses Modell sind die fünf Untersysteme realisiert in den Blöcken namens *Vorberechnungen, Anfahrt, Abfahrt, Streichen* und *Ueberfuehren*. In ihnen spielen sich die wesentlichen Vorgänge zur Trajektoriengenerierung ab. Wie die Namen schon vermuten lassen, werden dabei die einzelnen Vorgänge getrennt voneinander generiert.

10.4.2.2. Vorberechnungen

Der Block *Vorberechnungen* hat folgende Aufgaben: Er liest aus den Arbeitsbereich Matlabs alle benötigten Daten wie Violinenkoordinaten, Transformationsmatrizen, Spielgeschwindigkeiten, usw. aus und übergibt sie an eine modifizierte Variante des bereits bekannten Programmfragments namens *Trajektorie_II*. Weiterhin werden auch die vom Liedgenerator vorher zu berechnenden Matrizen *TAbspiel* und *TMatrix* ausgelesen[3], welche das zu spielen Stück definieren. Es ist also Sorge zu tragen, dass sich all diese Informationen vor Beginn der Berechnungen im Arbeitsbereich (Workspace) von Matlab befinden. Die Eingangsvariable *a* des Blockes dient als Zähler des Systems, mit welchem die auszulesende Spalte der Matrix *TMatrix* definiert wird. Der Ausgang *Ende_Matrix* wird beim Erreichen der letzten Spalte von *TMatrix* auf den Wert Eins gesetzt und stoppt in Folge dessen die Simulation. Die Ausgangsvariable *Aktion* bestimmt das als nächstes auszuführenden Teilsystem. Das Untersystem *Vorberechnungen* ist in Abbildung 10.12 dargestellt.

10.4.2.3. An- und Abfahrt

Die beiden Untersysteme An- und Abfahrt sind sich vom strukturellen Aufbau sehr ähnlich und werden aus diesem Grund hier zusammen betrachtet. Für die textuelle Programmierung wurden hierbei Programmfragmente aus *Trajektorie_II* entnommen und geeignet modifiziert, um sie für eine Echtzeitberechnung nutzbar zu machen. Die Werte der Rotation werden anhand der übergegebenen Werte *T_aktuell* und *T_Lage_soll* vollständig in Echtzeit berechnet, während für den translatorischen Bewegungsablauf Daten aus einer in *Vorberechnungen* ermittelten Kurve entnommen werden. Die Variable *r* dient dabei als zählendes Glied, welches auf den Wert der Variable *Punktanzahl* gegrenzt wurde. Die Trajektoriendaten werden in der Form von Umweltkoordinaten direkt auf den Ausgang gegeben. Wurde die Berechnungen abgeschlossen, so wird mit dem Ausgang *Ende* dies signalisiert und der Block *Vorberechnungen* wird in Folge dessen die Vorarbeiten für die nächste Aktion laut *TMatrix* übernehmen und diese starten bzw. das Programm beenden. Das Untersystem Anfahrt ist in Abbildung 10.13 dargestellt.

10.4.2.4. Streichen

Das Untersystem *Streichen* funktioniert ähnlich wie das der An- und Abfahrten, wobei hier noch nicht einmal eine Rotationsregelung von Nöten sein wird. Die benötigte Orientierung wird als konstante an das System übergeben. Die lineare Translation wird mit

[3] Eine Verbindung der beiden Matrizen zu einer einheitlichen konnte aus zeitlichen Beweggründen nicht mehr realisiert werden, wäre aber durchaus sinnvoll.

den entsprechend modifizierten Punktabständen, ausgehend von der zu spielenden Notenlänge, der gewünschten Lautstärke und der nachfolgenden Überführungszeit, bereits in Umweltkoordinaten übergeben und ausgelesen. Hierbei sind durchaus noch Modifizierungen vorstellbar, um zum Beispiel eine Koordinatentransformation in das System mit zu integrieren und somit die Zeit der Datenvorbereitungen für Streichvorgänge zu reduzieren. Zählgliedfunktionen sowie die Beendigung der Aktion sind identisch mit denen der bereits im vorherigen Abschnitt beschriebenen aus An- und Abfahrt. Das Untersystem ist in Abbildung 10.14 dargestellt.

10.4.2.5. Überführen

Im Untersystem *Ueberfuehrungen* hingegen, werden die Daten der translatorischen Bewegung zwar auch als Kurve übergeben, allerdings findet die komplette Koordinatentransformation in Echtzeit während der Überführung selbst statt. Die verkürzt die Rechenzeiten der Datenvorbereitung jeder Überführung deutlich. Die Rotationsregelung wird wie schon in den anderen Untersystemen komplett in diesen Programmfragment berechnet und transformiert. Als zählendes Glied dient diesmal die Variable *q,* welche wiederum auf die *Punktanzahl* begrenzt wird. Das Untersystem ist in Abbildung 10.15 dargestellt.

10.4.3. Fortschritt

Das Programm der inkrementellen Trajektoriengenerierung stellt eine experimentelle Methode dar. Ziel sollte es vor allem sein, dass die aus dem Liedgenerator berechneten und sehr wichtigen Daten nutzbar gemacht werden können. Der Liedgenerator muss hierbei die Berechnungen bis zur Matrix der anzuspielenden Punkte *TMatrix* vornehmen, um die gewünschten Verifikationen einsetzen zu können (vgl. 9.2.2.2). Dies bedeutet aber wiederum, dass eine inkrementelle Bearbeitung erst nach Erstellung besagter Matrix möglich wird. Das Echtzeitmodell wird somit seine Berechnungen ab Programmfragment *Trajektorie_II* (in Äquivalenz zur textuellen Matlabprogrammierung) ansetzen, um von dort an eine Trajektorie in Übereinstimmung mit den komplexen Vorgaben des "Spielens" generieren zu können. Diese Vorgangsweise impliziert natürlich auch, dass die Violinen-Koordinaten vor Beginn der inkrementellen Bahngenerierung bekannt sein müssen, obwohl auch hier Verifikationen denkbar sind.

Will man nach der Fertigstellung eines laufenden Programmes später einzelne "Noten" (analog zu ein bis zwei Spalten in *TMatrix*, abhängig von der Kombinationsweise der Noten) dem Programm übergeben und ausführen lassen, so ist jedoch Vorsicht geboten: Denn ähnlich wie bei einer natürlichen Spielweise (von einem Menschen ausgeführt) wird die gerade zu spielende Note von der vorherigen sowie der kommenden beeinflusst. Ersteres stellt weniger ein Problem dar, da die vorherige Note bereits bekannt ist und somit schon in die Erstellung der anzufahrenden Punkte mit integriert wurde, bzw. werden kann. Anders sieht dies bei der folgenden Note aus. Da der Liedgenerator in der Lage ist, rückwirkend auf die Matrix *TMatrix* zu wirken, ist das Vorgehen bei einer Ansteuerung des inkrementellen Programmes bei Übergabe einzelner Noten zu überprüfen und gegebenfalls Modifizierungen vorzunehmen.

Nichts desto trotz befinden sich das Programm noch in der Entwicklungsphase und wird voraussichtlich in dieser Praktikumsarbeit auch nicht mehr fertiggestellt werden können. Folgende Aufgaben wurden bereits mit Erfolg getätigt:

- Aufspaltung und Umstrukturierung der textuellen Matlabprogrammierung und Integration dieser in geeignete Untersysteme
- Umschrieb der textuellen Programmierung für eine Nutzbarmachung im Simulink-Modell[4]
- Erweiterung der Funktionsparameter und Über- bzw. Rückgabe aller benötigten Daten
- Verknüpfung der Untersysteme und Erstellung eines ausführbaren Modells

Beim Start des Modells werden bereits Umweltkoordinaten in Echtzeit berechnet und ausgegeben, welche allerdings in dieser Form noch nicht nutzbar sind. Ein noch nicht analysiertes Problem stellt die fehlerhafte Übertragung der Orientierungsmatrix dar, welches zu untersuchen wäre. Desweiteren existieren nach dem Umschrieb der Algorithmen Probleme in der Zeitparametrisierung, welche in *distanzen3D* realisiert wird. Das Problem besteht darin, dass die Abstandsvariable *Abstand,* welche die reelle Entfernung vom zuletzt in der Zeitparametrisierung neu generierten Trajektorienpunkt zum nächsten der durch die Beziérfunktionen generierten Trajektorienpunkt bestimmt, in einigen Fällen nicht gegen die Abstandsvariable L konvergiert, welche den vorher berechneten Wunschabstand angibt. Als Folge daraus meldet das Programm eine Bereichsüberschreitung in der Variable *M_neu,* welche die neu generierte Kurve inne halten wird. Die genauen Ursachen dieses divergenten Verhaltens sind noch ungeklärt.

Nach der Lösung dieser zwei Probleme und evtl. noch andere bisweilen unentdeckter Ungereimtheiten sollte einer erfolgreichen Trajektoriengenerierung jedoch nichts mehr im Wege stehen, wobei die erzeugte Trajektorie vor ihrer Anwendung penibelst auf die Ausführbarkeit auf einem Manipulator untersucht werden sollte.

[4] hierbei sei zu beachten, dass keine variablen Veränderlichen mehr zugelassen sind und somit alle Variablen statisch in Typ und Größe bereits während der Kompilierung definiert werden

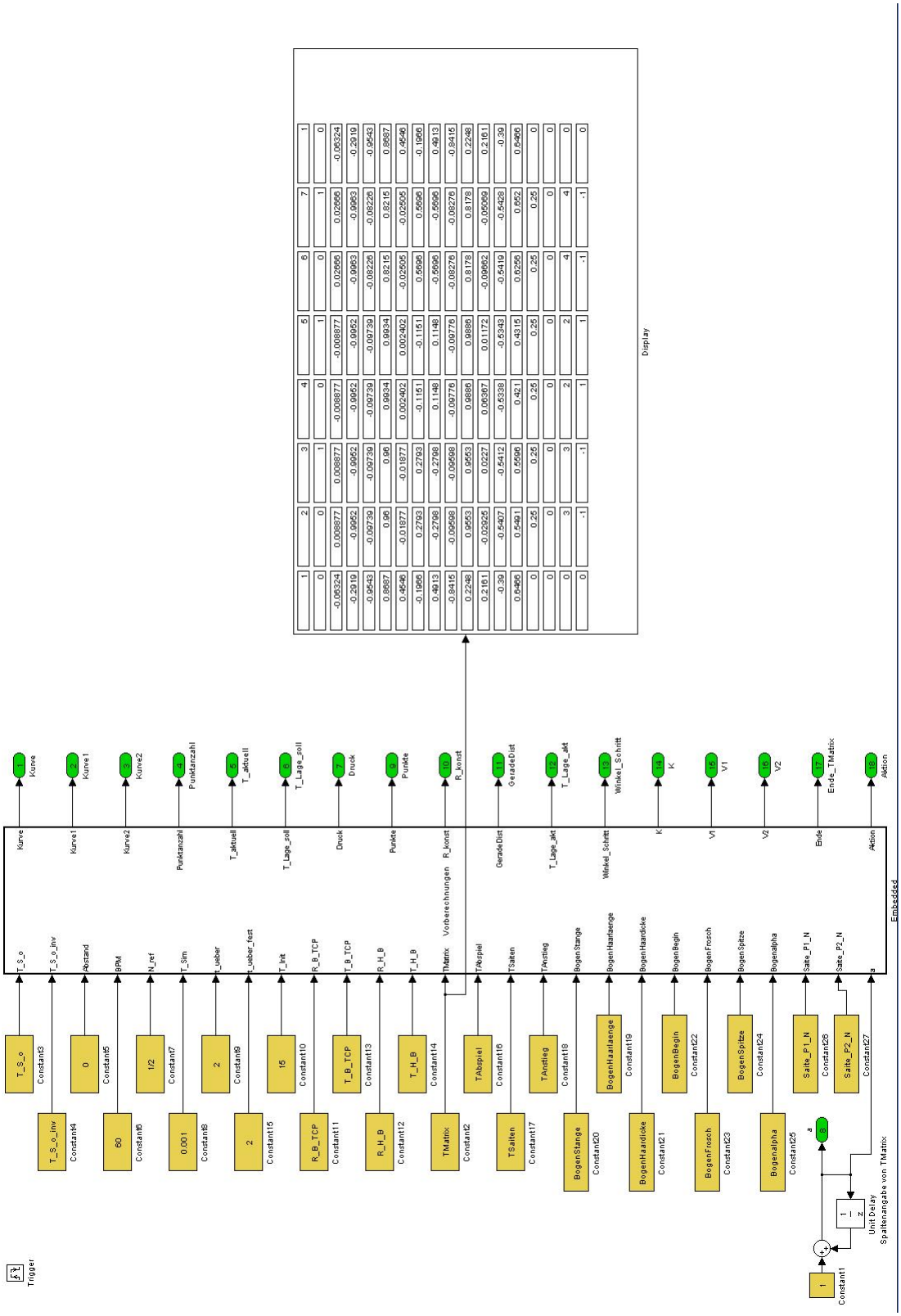


Abbildung 10.12.: Vorberechnungen

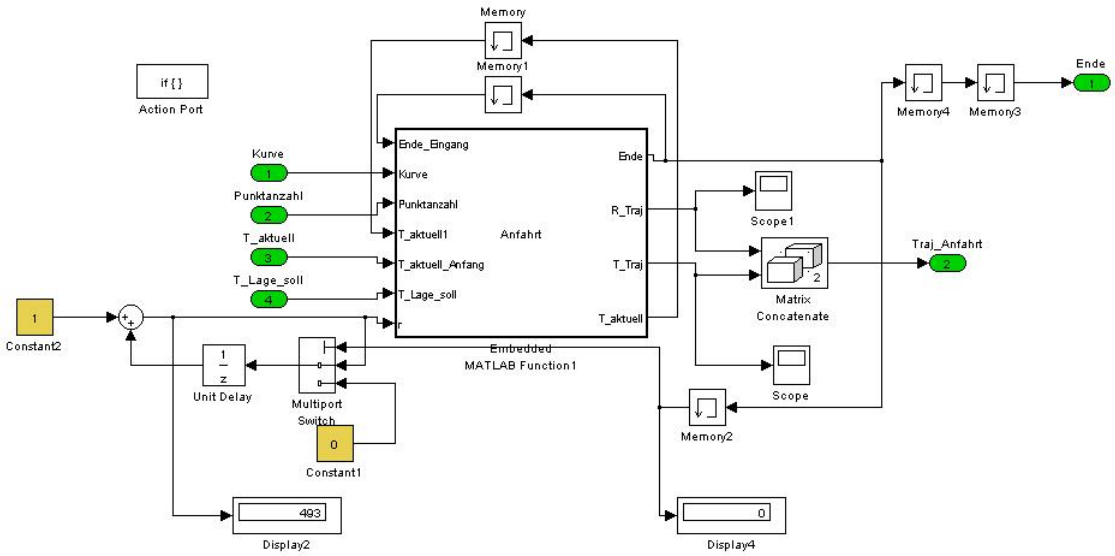


Abbildung 10.13.: Anfahrt

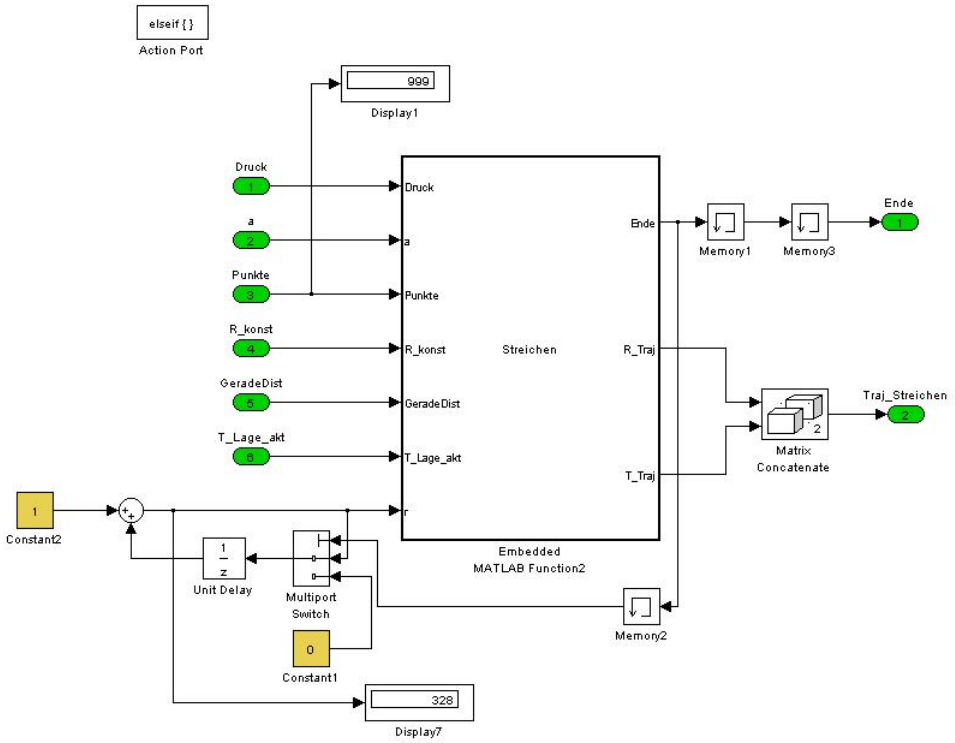


Abbildung 10.14.: Steichvorgang

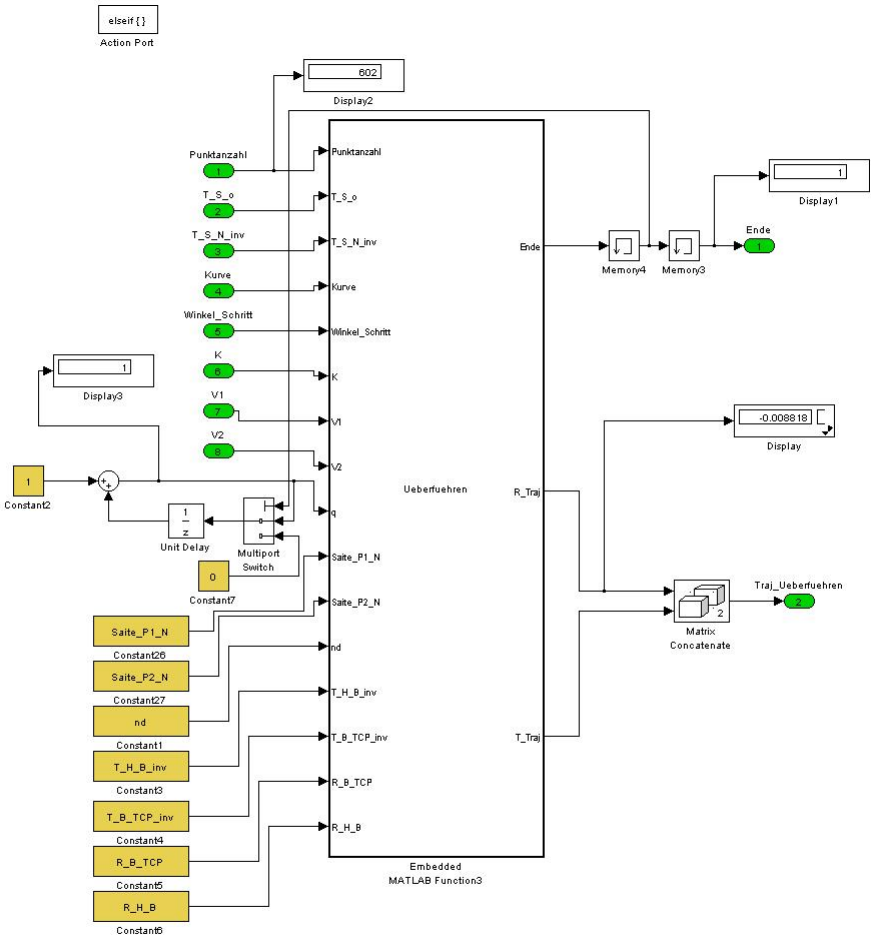


Abbildung 10.15.: Überführen

11. Bestimmung der Violinenkoordinaten

In diesem Kapitel sollen nun einige Möglichkeiten zur Bestimmung der Violinenkoordinaten aufgelistet werden. Tatsächlich konnte bisher noch keiner dieser Varianten in einem praktischen Versuch getestet werden. Es wurden allerdings schon einige Überlegungen angestellt, welche hier in kurzer Form erläutert werden sollen.

11.1. Geometrische Koordinatenbestimmung nach Haase

Das von *Haase* erstellte Programmfragment namens *lage* basiert auf einer Bestimmung der Violinenkoordinaten mithilfe einer Ausmessung der geometrischen Komponenten der Violine und des Bogens. Hierfür werden folgende Werte benötigt[1]:

Bogen (vgl. 11.1 & 8.1)

- Länge Bogenstange (63,7cm)
- Länge spielbarer Haarbereich (58cm) sowie die Haardicke (0,1cm)
- Abstand TCP $\leftrightarrow$ Bogenbeginn (4cm)
- Größe Bogenspitze (2,4cm) & Bogenfrosch (7,3cm)
- Winkel α zur Bestimmung der Neigung des Bogens (optional; 0,0°)

Violine

Am Steg (vgl. 11.2)

- Breite (3,4cm) und Höhe (3,8cm) des Steges (Maximalwert)
- Die Steghöhe an den Außensaiten (3,3cm)
- Stegdicke (0,5cm)

[1] Die in den Klammern angegebenen Werte wurden bereits messtechnisch ermittelt. Als Instrument diente hierbei die Violine von *Patrick van der Smagt*.

An zweiter Messstelle

- Länge der Saiten (vom Steg bis zweiter Messstelle; 32,7cm)
- Abstand der Saiten voneinander (0,55cm)
- Höhenunterschied zur Basis (3,2cm)

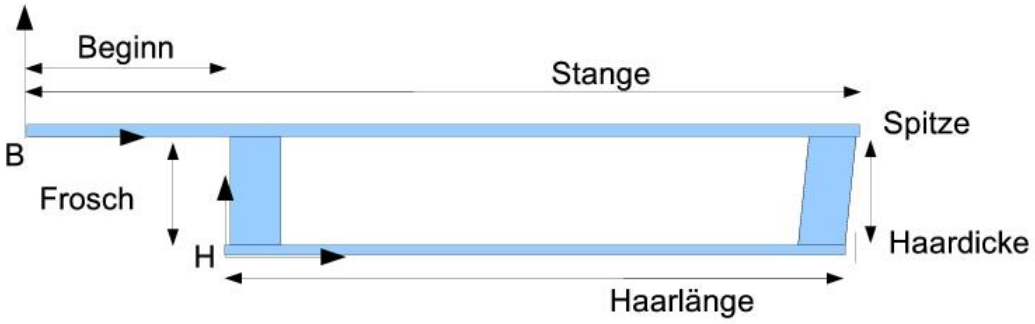


Abbildung 11.1.: Bogen [29]

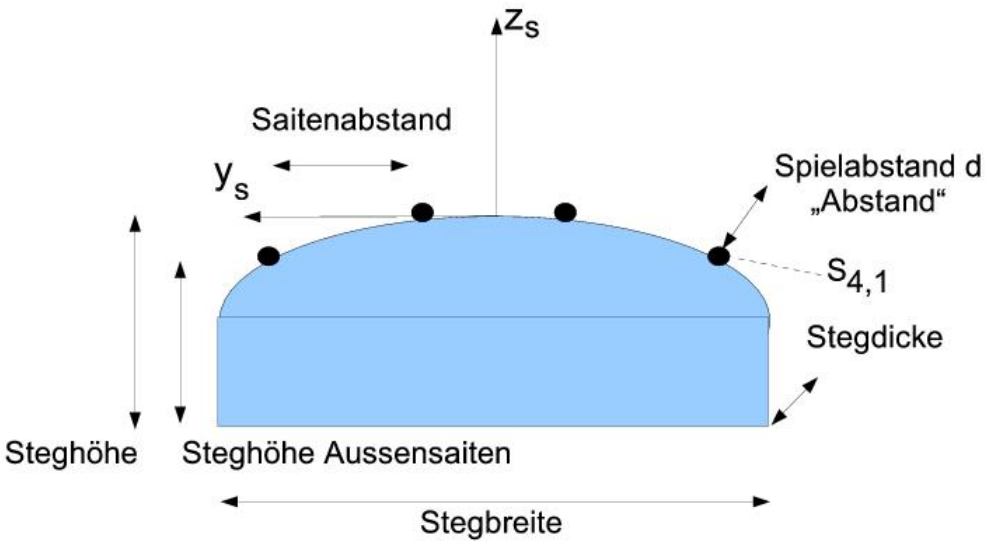


Abbildung 11.2.: Steg [29]

Zu beachten sei hier, dass die zweite Messstelle zwar beliebig gewählt werden kann, aber die größte Genauigkeit bei einer möglichst großen Entfernung zum Steg erzielt wird.

Fazit

Da diese Methode im Programm bereits vorgesehen und entsprechen programmiert wurde, stellt dies mit Sicherheit die einfachste Methode zur Bestimmung der Koordinaten dar. Ein praktischer Versuch zur Ausmessung der Koordinaten hatte allerdings gezeigt, dass dies mit herkömmlichen Hilfsmitteln nur sehr schwer und in hohem Maße Fehleranfällig ist, da schon kleine Messungenauigkeiten aller Voraussicht nach zu große Positionsabweichungen in den später transformierten Umweltkoordinaten hervorrufen würden. Die wirkliche Effektivität dieser Methode wäre zu untersuchen. Die bereits messtechnisch ermittelten Werte sind derzeit im Programm integriert.

11.2. Koordinatenbestimmung mithilfe der Vorwärtskinematik

Violine

Eine weitere Methode zur Bestimmung der Objektkoordinaten gebietet die Nutzung der Vorwärtskinematik. Hierbei werden keine geometrischen Kenntnisse über die Violine mehr benötigt. Wird die Violine im Raum geeignet montiert, so ist mit einem Anfahren des TCP eines geeigneten Manipulators jeder spielbare Anfangs- und Endpunkt jeder Saite in Roboterkoordinaten bekannt und durch Transformation in Umweltkoordinaten umrechenbar. Mir den so gewonnenen Informationen ist eine Abbildung der Saiten in Umweltkoordinaten möglich, ohne weitere Informationen über die Beschaffenheit der Violine besitzen zu müssen. Diese Methode reduziert durch die direkte Bestimmung der relevanten Punkte die Anzahl möglicher Fehlerquellen erheblich und ist darüber hinaus mit einen geeigneten Adapter am Manipulator sehr leicht durchzuführen. Mögliche Fehlerquellen liegen dabei allerdings in der Bestimmung der Gelenkwinkelkoordinaten (Motorpositionsmessung) sowie in fehlerhaften Vorwärtstransformationen, beispielsweise durch Elastizitätseigenschaften der Glieder des Manipulators (vgl. Abschnitt 4.1). Obwohl letztere Fehlerursache durch das geringe Gewicht der Extremitäten der LBRIII-Generation eher vernachlässigbar erscheint, sollten bei einer möglichen Fehlerursachenforschung solche Aspekte nicht ganz außer acht gelassen werden. Zu guter letzt bürgt natürlich auch noch ein ungenaues Heranfahren des TCP an den Saiten eine mögliche Fehlerquelle. Hierbei ist mit Sorgfalt zu arbeiten.

Entscheidet man sich nun mithilfe dieser Methode die Violinenkoordinaten, bzw. die eigentlich relevanten Saitenkoordinaten, zu bestimmen, so ist in Folge dessen dafür Sorge zu tragen, dass diese Informationen an geeigneter Stelle ins entsprechende Programm übertragen und richtig Verarbeitet werden. Ein möglicher Implizierungsort befindet sich im Programmfragment *lage*[2], Zeile 140 bis 148, in denen bereits aus den gegebenen Violinenkoordinaten für jede Saite zwei Punkte in Basiskoordinaten berechnet vorliegen.

[2] unter *violine/Wendt/Violine_ 02/trajektorie/lage.m*

Bogen

Genauso wie zur Koordinatenbestimmung der Violinensaiten, ist diese Methode auch zur Bestimmung der Bogenhaare anwendbar. Hierbei ist allerdings ein zweihändiger Manipulator (z. B. Justin) nötig, oder zwei Manipulatoren, welche im selben Basissystem arbeiten. Ziel ist es dabei, die relative Lage (und Orientierung) des bespielbaren Bogenhaarbereichs und des TCP des zu spielenden Gliedes zu erhalten. Vorgegangen wird folgend: Der Bogen wird an einem TCP_1 befestigt, während mit den anderen TCP_2 die Haarkoordinaten in Roboterkoord. und anschließend in Umweltkoord. bestimmt werden. Durch Differenzbildung der zwei (Lage-)Koordinaten P_1 und P_2 des TCP_2

$$\overrightarrow{h_B} = {}^0P_1 - {}^0P_2 \tag{11.1}$$

$$\overrightarrow{h_B} = \begin{bmatrix} P_{1_x} \\ P_{1_y} \\ P_{1_z} \end{bmatrix} - \begin{bmatrix} P_{2_x} \\ P_{2_y} \\ P_{2_z} \end{bmatrix} \tag{11.2}$$

lässt sich eindeutig die Orientierung der Haare des Bogens bestimmen und eine Transformationsmatrix ${}^B_H T$ von Haar- zu Basiskoordinaten erstellen.

$${}^B_H R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\arctan\frac{\triangle P_y}{\triangle P_z}) & -\sin(\arctan\frac{\triangle P_y}{\triangle P_z}) \\ 0 & \sin(\arctan\frac{\triangle P_y}{\triangle P_z}) & \cos(\arctan\frac{\triangle P_y}{\triangle P_z}) \end{bmatrix} \tag{11.3}$$

$${}^B_H R_y = \begin{bmatrix} \cos(\arctan\frac{\triangle P_x}{\triangle P_z}) & 0 & \sin(\arctan\frac{\triangle P_x}{\triangle P_z}) \\ 0 & 1 & 0 \\ -\sin(\arctan\frac{\triangle P_x}{\triangle P_z}) & 0 & \cos(\arctan\frac{\triangle P_x}{\triangle P_z}) \end{bmatrix} \tag{11.4}$$

$${}^B_H R_z = \begin{bmatrix} \cos(\arctan\frac{\triangle P_x}{\triangle P_y}) & -\sin(\arctan\frac{\triangle P_x}{\triangle P_y}) & 0 \\ \sin(\arctan\frac{\triangle P_x}{\triangle P_y}) & \cos(\arctan\frac{\triangle P_x}{\triangle P_y}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11.5}$$

$${}^B_H R = {}^B_H R_x \cdot {}^B_H R_y \cdot {}^B_H R_y \tag{11.6}$$

$${}^B_H T = \begin{bmatrix} & {}^B_H R & & \overrightarrow{h_B} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.7}$$

Multipliziert man diese Matrix mir der Inversen der Transformationsmatrix ${}^B_{TCP_1} T$, so erhält meine eine Transformationsmatrix, welche die relative Lage und Orientierung der Haarkoordinaten (von einem definierten Punkt aus) zum TCP_1 beschreibt.

$${}^{TCP_1}_H T = {}^B_H T \cdot {}^B_{TCP_1} T^{-1} \tag{11.8}$$

11.3. Verifikationsverfahren mithilfe virtueller Positionsbestimmung

Als dritte Methode zur Violinen-Koordinatenbestimmung möchte ich hier noch eine Verifikationsmethode vorstellen, falls die beiden in 11.2 und 11.3 erwähnten Methoden zu ungenaue Ergebnisse für ein sauberes Violinenspiel liefern sollten.

Dieses Verfahren basiert darauf lediglich die in einer Positionssteuerung für das Violinespielen sehr wichtigen Informationen für eine Trajektoriengenerierung mit hoher Präzision zu bestimmen. Für alle anderen mitunter vom Spielenden abhängigen Lageeigenschaften soll hier eine ausreichende Genauigkeit genüge tun.

Die Positionsdaten, welche für ein rein positionsgesteuertes Spiel große Wichtigkeit besitzen, sind die Orientierung der Bogenhaare relativ zum TCP des ausführenden Manipulators, sowie die genaue Lageposition der Saitenaufsetzpunkte. Folgend wird in der Verifikationsmethode vorgegangen:

Als erstes wird die relative Orientierung der Bogenhaare bestimmt. Eine sicherlich geeignete Methode dafür wurde bereits in Unterabschnitt 11.2 beschrieben. Eine andere noch einfachere Methode impliziert das Anfahren zweier ausgewählter Bogenhaarpunkte an einen starren Punkt im Raum bei konstanter TCP-Orientierung. Durch eine Differenzbildung der beiden Lagepunkte erhält man die Orientierung der Haare. Aus Haarorientierung und konstant gehaltener TCP-Orientierung kann analog zu 11.1 bis 11.8 eine Drehungsmatrix erstellt werden, welche die relative Orientierungsverschiebung beschreibt. Im nächsten Schritt werden unter Zuhilfenahme der ermittelten Haarorientierung (muss im Programm implementiert werden) alle benötigten Violinenkoordinaten mit einer der bereits bekannten Methoden ermittelt. Anschließen werden mithilfe der dabei gewonnenen Daten mit dem Beginn der spielbaren Haarspitze alle Anfangs- und Endpunkte jeder Saite angefahren, wobei ein Sicherheitsabstand in Z-Richtung zu halten ist[3]. Die genaue Positionsbestimmung soll nun manuell durch ein Absinken des Sicherheitsabstandes in Z-Richtung erfolgen, evtl. unter Zuhilfenahme der Drehmomentsensoren. Durch die somit gewonnenen Informationen und der genauen Bestimmung der relativen Orientierung von Bogenhaare und TCP ist nun äußerst präzise bekannt, in welche Position der Endeffektor zum Streichen einer Saite fahren muss und welche Orientierung er dabei zu halten hat, obwohl die eigentlich Violine- und Bogenkoordinaten nicht explizit berechnet wurden. Alle anderen mit dieser Verifikation nicht bestimmten Daten werden aus den bereits beschriebenen Koordinatenberechnungen genommen. Dies ist durchaus zulässig, da alle anderen Positionsdaten in einem gewissen Maße schwanken dürfen, ohne Einfluss auf die Qualität des Spielens haben zu können[4]

[3] Siehe Variable *Abstand* in *Start.m*

[4] Es ist hierbei nicht auszuschließen, dass solche Schwankungen eine Veränderung der Klangfarbe hervorrufen könnten. Ob dies einen Gewinn oder Verlust darstellt, ist allerdings eher eine subjektive Entscheidungsfrage.

Literaturverzeichnis

[1] http://de.wikipedia.org/wiki/Deutsches_Zentrum_f%C3%BCr_Luft-_und_Raumfahrt, Juni 2008.

[2] de.wikipedia.org/wiki/Ludwig_Prandtl, Juni 2008.

[3] http://www.dlr.de/DesktopDefault.aspx/tabid-71/453_read-637/gallery-1/gallery_read-Image.1.217, Juni 2008.

[4] http:// esamultimedia.esa.int/images/marsexpress/wallpapers/wallpaper_HRSC_tholus800.jpg, Juni 2008.

[5] http://www.astronomie.de/raumfahrt/mex/Februar05/HRSC_Funktionspr.jpg, Juni 2008.

[6] http://bilddb.rb.kp.dlr.de/deutsch/..%5CBilder%5Cjpeg%5C207.jpg, Juni 2008.

[7] http://www.nasa.gov/images/content/191019main_Columbus-EPF1.jpg, Juni 2008.

[8] http://www.bwb.org /02DB022000000001/CurrentBaseLink/W276BBHC902INFODE/$FILE/E61Oberpfaffenhofen_640.jpg, Juni 2008.

[9] http://www.dlr.de/en/Portaldata/1/Resources/standorte/oberpfaffenhofen/dlr_oberpfaffenhofen_ikonos_200.jpg, Juni 2008.

[10] http://www.dlr.de/Portaldata/1/Resources/portal_news/newsarchiv2006/justin_automatica.JPG, Juni 2008.

[11] http://www.dlr.de/Portaldata/1/Resources/portal_news/newsarchiv2004 /Hand-Tasse_hires.jpg, Juni 2008.

[12] http://www.euron.org/images/robots/180_kinemedic.png, Juni 2008.

[13] http://www.dlr.de/rm-neu/en/Portaldata/52/Resources/images/institute /robotersysteme/medical_robotics/herz1_200.jpg, Juni 2008.

[14] http://bilddb.rb.kp.dlr.de/deutsch/%5CBilder%5Cjpeg%5C551.jpg, Juni 2008.

[15] http://www.antriebspraxis.de/ai/resources/39ccbd8394c.pdf, Juni 2008.

[16] http://www.dlr.de/rm-neu/desktopdefault.aspx/tabid-3868/5903_read-8721/, Juni 2008.

[17] www.uni-karlsruhe.de/download/KinderUniRoboterOhneVideo.ppt, Juni 2008.

[18] http://www.dlr.de/DesktopDefault.aspx/tabid-3192/5066_read-3401/gallery-1/gallery_read-Image.1.1720/, Juni 2008.

[19] http://www.dlr.de/rm-neu/en/Portaldata/52/Resources/images/bildgalerie/hirzinger_hand_I.JPG, Juni 2008.

[20] http://de.wikipedia.org/wiki/Geschichte_der_Automaten, Juni 2008.

[21] http://de.wikipedia.org/wiki/Geschichte_der_Automaten, Juni 2008.

[22] http://de.wikipedia.org/wiki/Roboter, Juni 2008.

[23] http://www.sn.schule.de/ gyfloeha/rt/lex01/lex0102.html, Juni 2008.

[24] http://www.cs.cmu.edu/ rapidproto/mechanisms/figures/combinedrev.gif, Juli 2008.

[25] http://www.ee.unb.ca/tervo/ee4353/mycube.gif, Juli 2008.

[26] John J. Craig. *Introduction to Robotics - Mechanics and Control*. Pearson - Prentice Hall, 1989.

[27] Troch Desoyer, Kopacek. *Industrieroboter und Handhabungsgeräte*. Oldenbourg, 1985.

[28] J.-B. Lugtenburg A. Truckenbrodt E. J. Kreuzer, H.-G. Meißner. *Industrieroboter*. Springer-Verlag, 1994.

[29] Thomas Haase. Mathematische analyse, modellierung und simulation des violine spielens am lbr 3. Praktikumsarbeit, Februar 2007.

[30] Thomas Haase. Simulationsaufbau der m-files. Praktikumsarbeit, April 2008.