

Praktikum Messtechnik

V4: Grundlagen der Medizinrobotik

Fachgebiet Mess- und Sensortechnik
Sommersemester 2023



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Motivation

Seit dem ersten Einsatz eines Medizinroboters im Jahre 1991 bei einer transurethralen Resektion der Prostata haben sich Robotersysteme in der Medizintechnik etabliert. Den Siegeszug verdanken sie ihren Vorteilen wie Präzision, höherer Effizienz, erhöhter Belastbarkeit und intelligenter Bauweise, sodass die Akzeptanz bei Ärzten und Patienten immer weiter steigt. Die hohe gesellschaftliche Relevanz und der Innovationswille ermöglichen ein stetiges Wachstum der zugelassenen Medizinroboter. Heutzutage sind Medizinroboter sehr vielseitig aufgestellt, sei es in der Service- und Pflegerobotik, in der Rehabilitation in Form von Orthesen, bzw. Exoskeletten oder in der Chirurgie als Assistenzroboter. Allen ist gemein, dass die medizinische Unterstützung des Arztes im Vordergrund steht, ohne ihn ersetzen zu wollen.

Die Medizinrobotik ist jedoch ein komplexes Themengebiet und gerade die Ansteuerung sowie das Vokabular in der Robotik müssen erst erlernt werden. In diesem Versuch soll daher, anhand eines Beispiels der Nadel-einführung im Bereich der interventionellen Radiologie und Onkologie, sowohl das Vokabular als auch die Grundlagen der Robotik vermittelt werden. Konkrete Anwendungsfälle finden sich in der Seedimplantation (Brachytherapie), Biopsie- und Stanzsystemen sowie interstitiellen Applikationen zur Erkennung und Behandlung von Tumoren. Das Ziel dabei ist, einen simulierten Tumor in einem Gewebephantom möglichst genau mit einer von einem Roboter geführten, sensorintegrierten Biopsienadel zu punktieren. Das zu untersuchende Szenario ist an eine Leberbiopsie angelehnt.

Wir wünschen Ihnen viel Erfolg und einen spannenden Versuch!

Sven Suppelt M.Sc.

Prof. Dr. mont. Mario Kupnik

An diesem Versuch oder bei der Entwicklung des Roboterarms haben mitgewirkt: Felix Herbst, Matthias Rutsch, Konstantin Fey, Seyfettin Devrim, Yannick Chatelais, Magnus Gärtner, Dennis Roth, Eric Pohl, Esan Sundaralingam, Philipp Witulla, Jan Hinrichs, Romol Chadda, Niklas Schäfer, Markus Hessinger

Inhaltsverzeichnis

1 Theoriteil	4
1.1 Grundlagen der Robotik	4
1.1.1 Koordinatentransformationen und Pose	4
1.1.2 Arbeitsräume (Joint Space und Task Space)	6
1.1.3 Kartesische Koordinatensysteme von Roboterarmen	6
1.1.4 Direkte und Inverse Kinematik	7
1.1.5 Bewegungstypen	9
1.1.6 Singularitäten	11
1.2 Der verwendete Roboterarm	14
1.3 Die verwendete Software	16
1.3.1 Das Terminal	16
1.3.2 ROS	16
1.3.3 Verknüpfungen zum Starten des Roboterarms	21
1.4 Inhalt des Versuchs	23
2 Vorbereitung	24
2.1 Aufgabe 1.1: Singularität	24
2.2 Aufgabe 1.2: Umgang mit Singularitäten	24
2.3 Aufgabe 1.3: ROS	24
2.4 Aufgabe 1.4: Python	24
3 Versuchsdurchführung	26
3.1 Aufgabe 2.1: Starten und Bewegen des Roboterarms	26
3.2 Aufgabe 2.2: Bewegung über Joint-Winkel	27
3.3 Aufgabe 2.3: Kartesische und Nicht Kartesische Bewegungen	27
3.4 Aufgabe 2.4: Bewegungen über Singularitäten	27
3.5 Aufgabe 2.5: Anbringen des Endeffektors	27
3.6 Aufgabe 2.6: Vorbereitungen und Planung des Nadeleinstichs	28
3.7 Aufgabe 2.7: Nadeleinstich	29
3.8 Aufgabe 2.8: Interpretation der Messwerte	31

1 Theorieteil

Der in diesem Versuch eingesetzte Roboterarm ist ein 6-Achs-Knickarmroboter mit dem Namen Helene, der am Fachgebiet Mess- und Sensortechnik im Rahmen von einer Abschlussarbeit [1] sowie mehrerer Projektseminare entwickelt wurde. Der Roboterarm ist hauptsächlich aus 3D-gedruckten Teilen hergestellt.

1.1 Grundlagen der Robotik

Um diesen Roboterarm im Praktikumsversuch erfolgreich einzusetzen, wird im Folgenden auf einige Grundlagen der Robotik eingegangen [2]. Im Anschluss wird der vorliegende Roboter und das Messzenario vorgestellt, sodass Sie für den Praktikumsversuch optimal vorbereitet sind.

1.1.1 Koordinatentransformationen und Pose

Ein Roboterarm besteht aus einer definierten Anzahl Gliedern, die beginnend von der Basis aus ansteigend hochgezählt werden (Abbildung 1.1).



Abbildung 1.1: Die Gelenke von Roboterarmen werden ausgehend von der Basis durchnummieriert. Sie dienen der Beschreibung von Posen, die durch die Einstellung der einzelnen Winkel der Gelenke erreicht werden. Zudem lassen sich die Elemente des Gesamtsystems durch die Nummerierung der Gelenke eindeutig beschreiben. Bildquelle [3].

Serielle Manipulatoren zeichnen sich dadurch aus, dass zwei aufeinanderfolgende Glieder stets durch exakt ein gemeinsames Gelenk miteinander in Verbindung stehen. Es lassen sich sowohl translatorische (Lineargelenk), wie auch rotatorische (Drehgelenk) Ausführungen unterscheiden, wobei sich beim Aufbau serieller Roboter das Drehgelenk etabliert hat. So handelt es sich auch bei der Verbindung der Glieder des genutzten Roboters ausschließlich um Drehgelenke, weshalb sich im Folgenden auf deren Beschreibung beschränkt wird. Das letzte Glied der kinematischen Kette wird als Endeffektor (oder auch Achse 6) bezeichnet. Diese Bezeichnung ist sowohl für das Endstück des Roboters, beispielsweise den Flansch, als auch für ein Werkzeug (Tool) gebräuchlich, welches gegebenenfalls zusätzlich an einem Flansch befestigt ist.

Für die Umsetzung vorgegebener Aufgaben und Bewegungsabläufe des Endeffektors ist das Wissen über dessen Pose von Bedeutung. Die Pose eines Körpers wird stets durch seine relative Lage zu einem anderen Körper beschrieben. Die Pose beinhaltet hierbei sowohl die relative Position als auch Orientierung des Körpers. Für den Anwendungsfall eines seriellen Manipulators bedeutet dies, dass die Lage des körperfesten Koordinatensystems S_B (engl. frame) eines Gliedes B durch die beiden genannten Komponenten bezüglich eines Referenzsystems, des Koordinatensystems S_A des vorherigen Körpers A , festgelegt ist. Die Position von S_B in Bezug zu S_A ist charakterisiert über eine Translation, darstellbar durch den Vektor p (Element \mathbb{R}^3), wobei die Indizes A und B für die zwei Koordinatensysteme stehen:

$${}^A p_B = \begin{bmatrix} {}^A p_B^x \\ {}^A p_B^y \\ {}^A p_B^z \end{bmatrix}. \quad (1.1)$$

Die Orientierung des körperfesten gegenüber des Referenzkoordinatensystems wird durch die (3x3) Rotationsmatrix R beschrieben:

$${}^A R_B = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \quad (1.2)$$

Ist ein Punkt d bezüglich S_B gegeben, lässt sich dieser relativ zu Bezugssystem S_A mithilfe der Transformationsbeziehung ausdrücken (Abbildung 1.2):

$${}^A d = {}^A p_B + {}^A R_B {}^B d. \quad (1.3)$$

Eine weitere Möglichkeit zur Angabe der Orientierung bietet die Euler-Winkel-Darstellung. Mit dieser lässt sich die 3x3 Rotationsmatrix aus dem vorherigen Abschnitt auf eine Drehung Theta (Element \mathbb{R}^3) mit insgesamt drei Rotationen darstellen. Dabei wird als erstes das Koordinatensystem um die z-Achse um den Winkel alpha gedreht. Danach wird um die y'-Achse (resultierend aus der ersten Drehung) um den Winkel beta gedreht. Schließlich wird noch um die z''-Achse (resultierend aus der zweiten Drehung) um den Winkel gamma gedreht. Dies wird auch als ZYZ-Euler-Winkel-Transformation bezeichnet und wird im EMS-Praktikum (Modul im Masterstudium) genauer beleuchtet.

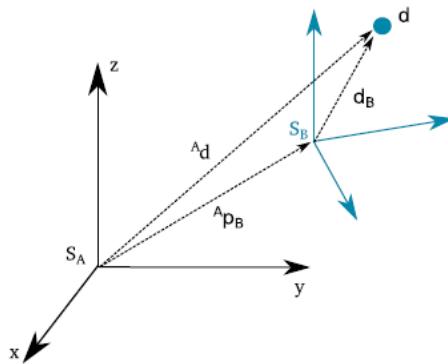


Abbildung 1.2: Repräsentation eines Punktes in zwei verschiedenen Bezugssystemen. Ist die Transformationsbeziehung zwischen den beiden Koordinatensystemen bekannt, kann die eine Repräsentation aus der anderen berechnet werden.

1.1.2 Arbeitsräume (Joint Space und Task Space)

Der Joint Space beschreibt die Konfiguration der gesamten kinematischen Kette des Roboters anhand seiner Gelenkparameter q_i . Diese enthalten im Falle rotatorischer Gelenke ihre jeweiligen Drehwinkel. Seine Dimension entspricht der Zahl der n Freiheitsgrade des Roboters und wird durch den Vektor $q = (q_1 \dots q_n)^T$ ($T =$ transponiert) (Element \mathbb{R}^n) gebildet.

Der Taskspace, auch Operational Space oder Arbeitsraum genannt, dient hingegen der Beschreibung der Pose des Endeffektors bezüglich der Basis des Roboters in kartesischen Koordinaten. Die Dimension des Arbeitsraumes entspricht der Anzahl der m DOF, welche zur Erledigung der vorgegebenen Aufgabe zur Verfügung stehen. Bei einer Aufgabe im dreidimensionalen, wie z.B. dem Führen und Ausrichten einer Nadel, sind dies $m = 6$ Freiheitsgrade, je drei für Position und Orientierung. Die Konfiguration des Endeffektors x_{ee} , wird somit anhand seiner generalisierten Koordinaten bezüglich Position p_{ee} und Orientierung θ_{ee} beschrieben.

Wird ein Punkt im Taskspace angegeben, wird dieser als frame bezeichnet und besteht aus Position p_{ee} und Orientierung θ_{ee} . Beispielsweise gibt der Frame:

$$\text{Frame_Start} = [0.2, 0.05, 0.3, *(\pi, -\pi/2, 0)] \quad (1.4)$$

einen Punkt bei $x = 0.2$, $y = 0.05$ und $z = 0.3$ an, der mittels der eulerschen Winkelarstellung um $R = \pi$, $P = -\pi/2$ und $Y = 0$ gekippt ist.

1.1.3 Kartesische Koordinatensysteme von Roboterarmen

Wird eine Bewegung eines Roboterarms im Taskspace gewünscht, so ist es notwendig, dass die Translation bzw. die Rotation angegeben werden muss, bevor diese im Anschluss ausgeführt werden kann. Generell ist es möglich, diese entweder relativ (Verschiebung um xx bezogen auf die aktuelle Position) oder absolut (fahre an diesen Punkt) anzugeben. Neben dieser Unterscheidung ist es bei Roboterarmen üblich, dass sich auf das Weltkoordinatensystem, auf das Basiskoordinatensystem oder auf das Werkzeugkoordinatensystem bezogen werden kann:

- **Weltkoordinatensystem:** Das Weltkoordinatensystem (WORLD) hat seinen Ursprung üblicherweise im rotatorischen Zentrum der ersten Achse (Abbildung 1.3(a)). Es ist das Hauptkoordinatensystem, welches unveränderlich im Raum liegt. Auf dieses Koordinatensystem sind alle anderen bezogen.
- **Basiskoordinatensystem:** Das Basiskoordinatensystem (BASE) wird meistens auf dem Werkstück oder der Werkstückaufnahme verwendet, um Punktkoordinaten im Bezug zum Werkstück oder zur Werkstückaufnahme einzuspeichern. Somit kann der Ursprungspunkt des Basiskoordinatensystems verschoben werden und die dazugehörigen Punktkoordinaten wandern mit.
- **Werkzeugkoordinatensystem:** Der Ursprung des Werkzeugkoordinatensystems (TOOL) befindet sich am TCP des Roboterarms (Abbildung 1.3(b)). Seine Orientierung nach der Konvention von Denavit-Hartenberg definiert. Dadurch, dass sich das Werkzeugkoordinatensystem mit dem Werkzeug bewegt, liegt es relativ zu diesem immer gleich, sodass nur relative Bewegungen angegeben werden können.

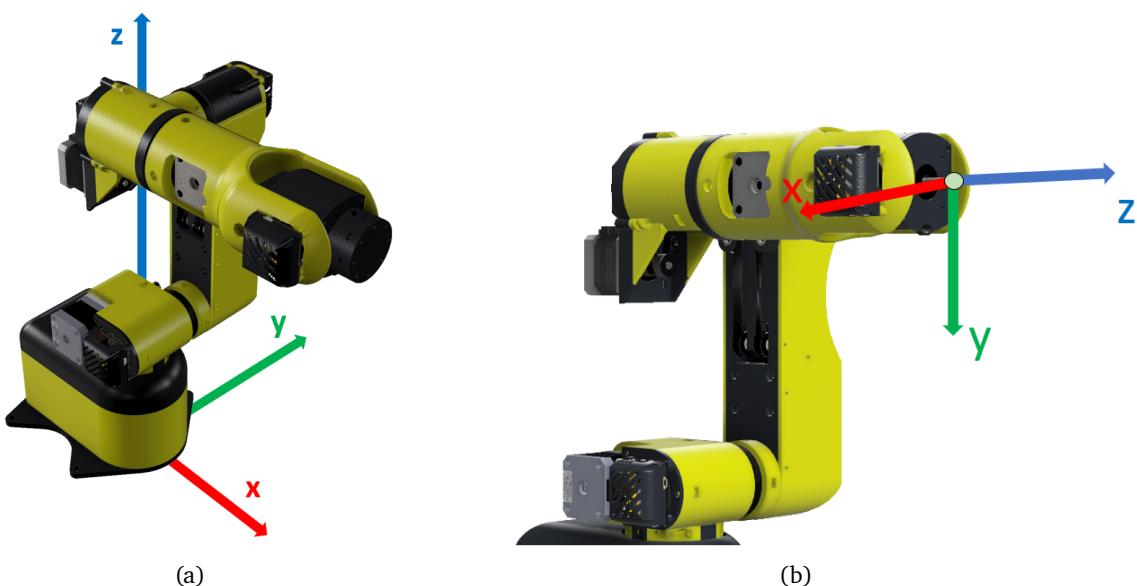


Abbildung 1.3: Bewegungen des Endeffektors können auf das Weltkoordinatensystem (a), auf den Endeffektor selbst (b) oder auf das Basiskoordinatensystem bezogen werden. Bildquelle [3].

In diesem Versuch wird ausschließlich im Weltkoordinatensystem gearbeitet.

1.1.4 Direkte und Inverse Kinematik

Die Berechnung der Kinematik führt zu zwei Ansätzen. Die direkte Kinematik (auch Vorwärtsskinematik oder Vorwärtstransformation genannt) dient dazu, aus dem Joint Space (Gelenkwinkel der Armelemente) eines Roboters den Taskspace (Position und Orientierung) des Endeffektors in Bezug auf das Basiskoordinatensystem zu bestimmen. Wird die Ausrichtung der einzelnen Segmente durch je ein festes Koordinatensystem dargestellt, stellt der Winkel des verbindenden Gelenks eine Rotationstransformation um eben jenen Winkel dar. Die Lage des Endes der offenen Kette in Relation zur Basis entspricht dann dem Produkt aller Transformationen.

Schwieriger gestaltet sich die inverse Kinematik. Dabei soll bei gegebener Position des Endeffektors die notwendige Ausrichtung der einzelnen Gelenke bestimmt werden (Abbildung 1.4).

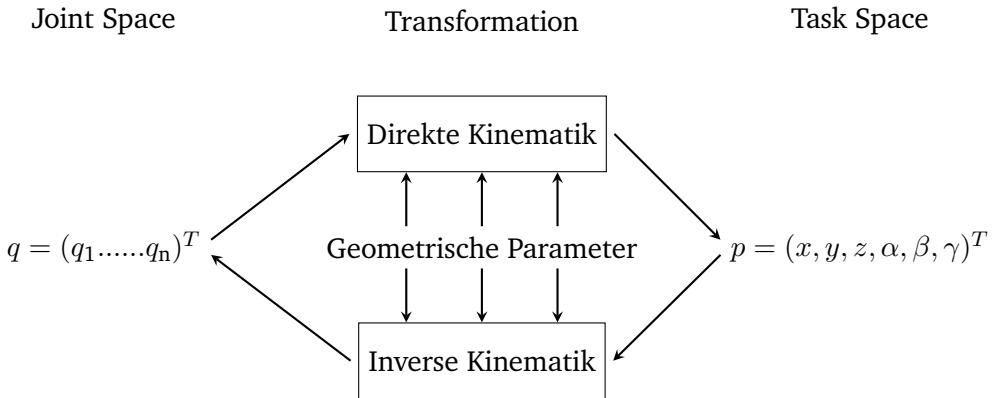


Abbildung 1.4: Die direkte Kinematik dient dazu, die Gelenkwinkel (Joint Space) in die Position des Endeffektors in Bezug auf das Basiskoordinatensystem (Task Space) zu transformieren. Die inverse Kinematik ist das logische Gegenstück dazu, der die Position des Endeffektors auf die Gelenkwinkel rückrechnet. Zur Bestimmung der Direkten und inversen Kinematik sind geometrische Parameter notwendig.

Diese Berechnung ist beispielsweise notwendig, wenn der Roboter einen definierten Punkt im Raum erreichen soll und die dazu benötigten Gelenkwinkel nicht bekannt sind. Die Gelenkwinkel werden jedoch zur Ansteuerung eines Roboterarms zwingend benötigt.

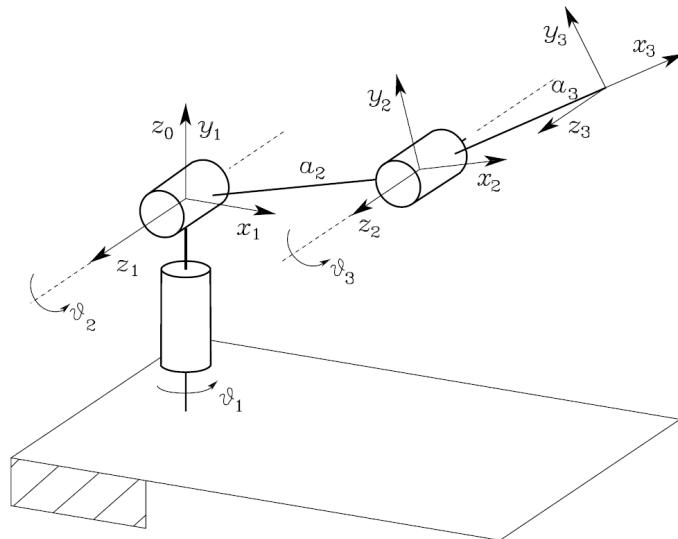


Abbildung 1.5: Die Objektkoordinatensysteme der einzelnen Gelenke sind nach Denavit-Hartenberg festgelegt.
Bildquelle [4].

Bei seriellen Anordnungen von Strukturauteilen und Gelenken kann die direkte Kinematik durch eine einfache Matrizenmultiplikation der Denavit-Hartenberg-Matrizen berechnet werden. Diese Matritzen sind ein mathematisches Verfahren, das auf der Basis von homogenen Matrizen und der Denavit-Hartenberg-Konvention (DH-Konvention) die Überführung von Ortskoordinatensystemen innerhalb von kinematischen Ketten beschreibt. Dabei wird jedes Objektkoordinatensystem einheitlich positioniert (Abbildung 1.5). Die z_{n-1} -Achse zeigt in Richtung der Rotationsachse. x_n steht senkrecht auf z_{n-1} und z_n und ist somit das Kreuzprodukt.

Die y_n -Achse wird so gewählt, dass das System ein rechtshändiges Koordinatensystem darstellt.

Zum Aufstellen der Denavit-Hartenberg-Matrizen sind geometrische Parameter notwendig, die in diesem Kontext Denavit-Hartenberg-Parameter genannt werden und tabellarisch niedergeschrieben werden. Die Tabelle enthält eine Zeile je Gelenk (Tabelle 1.1). Die Transformationen des Objektkoordinatensystems T_{n-1} in das nächste T_n in der seriellen Kinematik kann dann durch vier Parameter angegeben werden. θ_n gibt die Rotation um die z_{n-1} -Achse an, der Abstand d_n ist eine Verschiebung entlang der z_{n-1} -Achse, der Abstand a_n entlang der x_n -Achse. Schließlich sorgt eine Rotation α_n um x_n für die Ausrichtung von z_n mit der neuen Gelenkkachse.

Tabelle 1.1: Die Geometrie von Abbildung 1.5 wird durch eine Tabelle dargestellt und beschreibt die Transformation der Objektkoordinatensysteme von Gelenk i_{n-1} zu i_n nach Denavit Hartenberg. Diese Tabelle beschreibt dabei die Denavit-Hartenberg-Parameter und ist damit nicht identisch zur direkten Kinematik.

Gelenk	θ	α_i	a_i	d_i
1	θ_1	90°	0	0
2	θ_2	0°	a_2	0
3	θ_3	0°	a_3	0

Mithilfe dieser Tabelle kann die Gesamte Transformationsmatrix aufgestellt werden, welche die Lage und Orientierung des TCP-Koordinatensystems TCP relativ zum Basiskoordinatensystem BKS ausdrückt. Sie ist gleichbedeutend mit der Lösung des direkten kinematischen Problems. Um aus der direkten Kinematik auf die inverse Kinematik zu schließen, gibt es mehrere Möglichkeiten. So kann die inverse Kinematik mathematisch bestimmt werden, indem die Matrix der direkten Kinematik invertiert wird, bzw. im Falle der Denavit Hartenberg Transformationsmatrizen können auch die einzelnen Gelenksmatrizen einzeln invertiert und aufmultipliziert werden. Hierzu kommen Jacobi-Matrizen zum Einsatz. Weiterhin existieren geometrische und numerische Methoden, welche jedoch meist nur vereinfacht sind.

Diese Berechnung der direkten und inversen Kinematik kann analytisch und damit auch in Echtzeit auf Maschinensteuerungen erfolgen. Bei Maschinen oder Robotern mit parallelkinematischer Struktur, die nicht durch Denavit-Hartenberg-Parameter beschrieben werden können, ist eine analytische Lösung der direkten Kinematik im Allgemeinen nicht möglich. Analytische Lösungen existieren hier nur unter strengen geometrischen Voraussetzungen.

1.1.5 Bewegungstypen

Mithilfe der inversen Kinematik ist es nun möglich, einen Roboterarm gezielt an gewünschte kartesische Koordinaten zu bewegen. Die Bewegung zu diesem Punkt wird Pfad genannt und wird im Allgemeinen zuerst geplant, geprüft und anschließend ausgeführt. Eine Bewegung zu einem Punkt muss nicht immer geradlinig sein, sodass sich verschiedene Bewegungstypen etabliert haben. Etabliert hat sich eine Untergliederung in vier Bewegungstypen:

- **PTP-Bewegung:** Bei der Point-to-Point-Bewegung (PTP) wird der gewünschte Zielpunkt des Roboters direkt in die inverse Kinematik eingegeben und der sich daraus ergebende Joint-Space direkt dem Roboterarm übergeben. Der jeweilige Regelkreis der einzelnen Gelenke sorgt dann dafür, dass der Zielpunkt erreicht wird. Die Achsen sind jedoch nicht miteinander synchronisiert, d.h. eine Achse kann noch in Bewegung sein, während die restlichen Achsen bereits ihr Ziel erreicht haben. Der

sich ergebende Pfad ist nicht der kürzeste Weg im Raum und somit keine gerade Linie (Abbildung 1.6(a)). Der genaue Pfad der Bewegung ist nicht vorhersehbar, bleibt allerdings immer gleich, solange die Rahmenbedingungen nicht geändert werden. Diese Bewegung wird genutzt, wenn der Roboter genügend Platz hat oder um über Singularitäten (siehe nächstes Kapitel) zu fahren. Es sollte beachtet werden, dass die Bewegung nicht geradlinig ist und somit nicht optimal für präzise Aufgaben geeignet ist (z.B.: Nadeleinstich).

- **LIN-Bewegung:** Die Linearbewegung (LIN) beschreibt eine geradlinige Bewegung im kartesischen Raum. Ausgehend von der aktuellen Roboterposition werden mittels Linearinterpolation viele einzelne Zwischenpunkte errechnet, die anschließend mit einem festen zeitlichen Abstand nacheinander durch eine PTP-Bewegung abgefahrene werden. Je mehr Zwischenpunkte errechnet werden, desto geradliniger ist die Bewegung des Roboterarms. Bei Helene ist der Interpolationsabstand auf 1 mm gesetzt. Durch diesen Bewegungstyp verfährt der TCP des Roboterarms auf einer Geraden zu der vorgegebenen Zielposition (Abbildung 1.6(b)). Diese Bewegung wird genutzt, wenn der Roboterarm präzise Arbeiten ausführen muss. Die vorgegebenen Punkte werden auf direktem Weg angefahren, ohne von der vorgegebenen linearen Bewegungsbahn abzuweichen.
- **CIRC-Bewegung:** Die kreisförmige (circular oder CIRC) Bewegung beschreibt eine kreisförmige Bahnbewegung, die vom Startpunkt aus über einen Hilfspunkt zum Ziel ausgeführt wird. Berechnet wird sie genauso wie die LIN-Bewegung und ergänzt hauptsächlich die anderen Bewegungsaarten.
- **SPL-Bewegung:** Die Spline (SPL) Bewegung ist ein Bewegungstyp, der sich besonders für komplexe, gekrümmte Bahnen eignet. Mit einer Spline-Bewegung kann ein Roboterarm komplexe Bahnen in einer kontinuierlichen Bewegung ausführen. Zwar könnten solche Bahnen auch mit mehreren LIN- und CIRC-Bewegungen erzeugt werden, eine SPL-Bewegung muss aber zwischenzeitlich keine neuen Pfade berechnen. Die sich ergebende Bewegung des Roboterarms ist somit flüssiger.

In diesem Versuch werden die Bewegungstypen PTP und LIN genutzt (Abbildung 1.6).

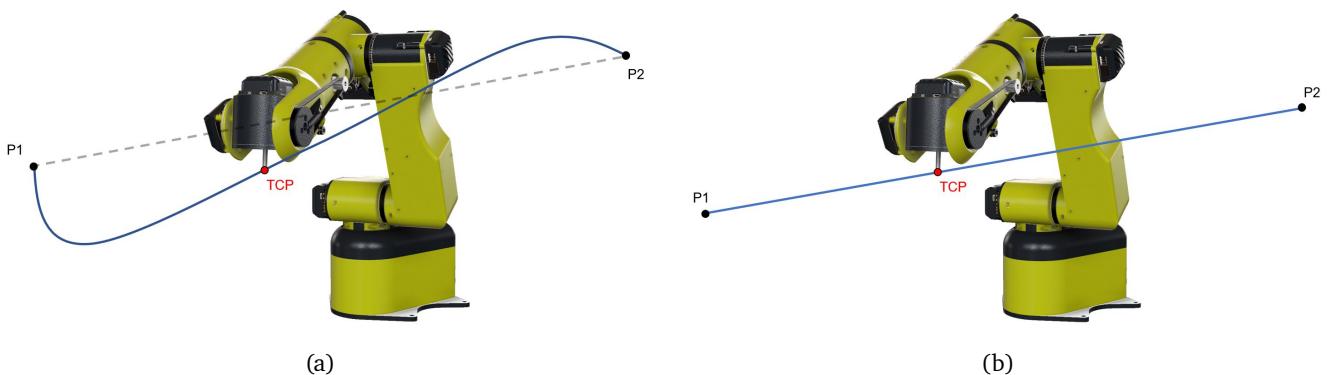


Abbildung 1.6: Bei der PTP-Bewegung (a) wird die inverse Kinematik nur für den Zielpunkt (P2) gelöst und die sich ergebenden Gelenkwinkel direkt an den Roboterarm übergeben. Die sich ergebende Endeffektorbewegung von Start- (P1) zum Zielpunkt ist nicht geradlinig. Bei der LIN-Bewegung (b) werden viele Punkte auf einer Geraden zum Zielpunkt interpoliert und für diese Punkte jeweils die inverse Kinematik gelöst. Zwar ist der Rechenaufwand höher, allerdings ergibt sich eine nahezu geradlinige Endeffektorbewegung.

1.1.6 Singularitäten

Bei der Berechnung der inversen Kinematik treten primär zwei Probleme auf. Zum einen ist die Lage der einzelnen Glieder bzw. Armelemente nicht unbedingt eindeutig. Es kann und wird im Allgemeinen mehrere valide Gelenkwinkelanordnungen geben, die zur gewünschten Lage des Endeffektors führen. Beispielsweise ist ein Mensch in der Lage, dass trotz festgehaltener Hand der Ellenbogen kreisförmig bewegt werden kann. In diesem Fall muss aus den sich ergebenden Gelenkwinkelanordnungen die sinnvollste Konfiguration ausgewählt werden. Zum anderen können unzulässige Konfigurationen entstehen, die mathematisch zwar korrekt sein können, allerdings von den einzelnen Gelenken nicht erreicht werden können. Das entsteht beispielsweise, wenn sich ein Gelenk über seinen Endanschlag hinaus drehen müsste. Wohingegen der zweite Fall relativ einfach zu umgehen ist, so ist der Umgang mit ersterem weitaus weniger trivial.

Wenn Sie versuchen, einen 6-Achs-Knickarmroboter im Gelenkkraum zu bewegen, stoppt der Roboterarm nur dann, wenn ein Gelenk an eine Winkelbegrenzung (Endanschlag) stößt. Versuch Sie hingegen, denselben Roboter linear im kartesischen Raum zu bewegen, wird er häufig in der Bewegung blockieren und nicht in der Lage sein sich weiter zu bewegen, obwohl er sich innerhalb seines Arbeitsraums befindet. Dies wird Singularität genannt und beschreibt eine Konfiguration, bei der ein Endeffektor des Roboterarms in bestimmte Raumrichtungen blockiert wird.

Bei einer Singularität verliert ein Roboterarm einen oder mehrere Freiheitsgrade. Singularitäten von 6-Achs-Knickarmrobotern können mit der inversen Kinematik erklärt werden:

$$q = J^{-1} \cdot v. \quad (1.5)$$

mit

$$p = [x, y, z, \omega_x, \omega_y, \omega_z]^T. \quad (1.6)$$

J ist dabei die Jacobimatrix und hat eine Dimension von 6x6 (Erklärung der Variablen siehe Abbildung 1.4). Die Jacobimatrix ist eine Funktion der Gelenkpositionen (q) und der Robotergeometrie. Wenn diese Matrix singulär wird (an bestimmten Gelenkpositionen), ist die obige Gleichung der inversen Kinematik nicht definiert und es existieren beliebig viele Lösungen. Dadurch wird der Roboterarm in bestimmte Richtungen blockiert und es wird davon gesprochen, dass sich der Roboterarm in einer Singularität befindet.

Das Problem mit Singularitäten ist nicht nur die Schwierigkeit, diese zu überqueren, sondern auch die hohen Gelenkgeschwindigkeiten, die sich aus der Annäherung an sie ergeben. Die Gelenkgeschwindigkeiten sind dabei extrem hoch, auch wenn sich der Endeffektor im kartesischen Raum kaum bewegt. Ein Roboter befindet sich in der Nähe einer Singularität, wenn die Determinante seiner Geschwindigkeits-Jacobimatrix nahe bei Null liegt, was den Effekt einer Division durch eine Zahl nahe Null hat. Derart hohe Gelenkgeschwindigkeiten können unerwartet sein und ein Sicherheitsrisiko darstellen [5]. Singularitäten lassen sich am besten durch Bilder (Abbildung 1.7) und Videos visualisieren, wie beispielsweise <https://www.youtube.com/watch?v=1D2HQcxeNoA>. Vergleichen Sie idealerweise die Beschreibungen der Singularitäten mit dem Video und vollziehen Sie die Singularitäten nach. Achten Sie auf die hohen Gelenkgeschwindigkeiten im Vergleich zur Bewegungsgeschwindigkeit des Endeffektors.

Bei einem 6-Achs Roboterarm treten drei Typen von Singularitäten auf:

- **Handgelenkssingularität:** Die am häufigsten anzutreffende Singularität bei einem 6-Achs Roboterarmen ist die Handgelenkssingularität. Sie tritt auf, wenn die Gelenke 4 und 6 auf einer Achse liegen. Bei einer Singularität im Handgelenk kann sich der Roboter nicht in Richtung der Achse von Gelenk 5 bewegen. Damit der *TCP* auf einer Linie durch die Singularität folgen kann, müssen sich die Gelenke 4 und 6 an der Singularität gleichzeitig um 90° in entgegengesetzte Richtungen drehen (rote Linie in Abbildung 1.7(a)). Das Überqueren einer Handgelenks-Singularität beim Verfolgen einer Linie ist also physikalisch möglich, aber an der Singularität bleibt der Endeffektor unbeweglich, während sich die Gelenke 4 und 6 drehen. Mit anderen Worten, es ist unmöglich, dass der Endeffektor die Singularität durchquert, ohne anzuhalten. Dazu kommt, dass einige Solver aufgrund der Singularität generell keine Lösung für die inverse Kinematik finden und so keine Bewegung des Roboterarms möglich ist.
- **Ellbogensingularität:** Die Ellbogensingularität tritt auf, wenn der Mittelpunkt des Handgelenks des Roboters auf der gleichen Ebene liegt wie die Gelenke 2 und 3. Ellbogen-Singularitäten sehen so aus, als wäre der Roboterarm zu weit gestreckt, wodurch der Ellbogen in der Position einrastet (Siehe Abbildung 1.7(b)). Wird der Roboterarm nun weiter bewegt, gibt es zwei mögliche Lösungen der inversen Kinematik. Diese Singularität ist die am wenigsten unerwartete und lässt sich leicht vermeiden.
- **Schultersingularität:** Die dritte und letzte Art der Singularität bei vertikal gelenkigen Roboterarmen mit geraden Handgelenken ist die Schultersingularität. Sie tritt auf, wenn der Mittelpunkt des Roboterhandgelenks in der Ebene liegt, die durch die Achsen der Gelenke 1 und 2 verläuft (oder durch die Achse von Gelenk 1 und parallel zur Achse von Gelenk 2). Diese Singularität ist die komplexeste, da sie nicht von einer einzigen Gelenkposition abhängt, wie die beiden anderen.

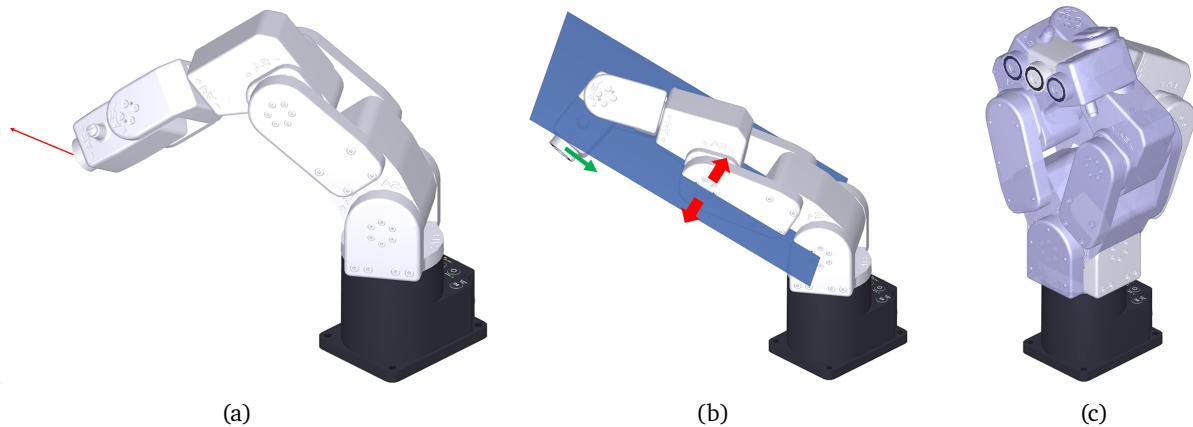


Abbildung 1.7: Bei einer Handgelenkssingularität (a) liegen die Achsen 4 und 6 auf einer Linie, sodass eine Bewegung (rote Linie) dazu führt, dass sich beide Achsen entgegen zueinander drehen müssen. Eine Elbogensingularität (b) tritt hingegen auf, wenn sich der Mittelpunkt des Handgelenks aus der gleichen Ebene wie Gelenk 2 und 3 befindet. Wird der Roboterarm entlang der grünen Linie bewegt, so ergibt die Inverskinematik zwei Lösungen (rote Pfeile). Die Schultersingularität tritt auf, wenn der Mittelpunkt des Roboterhandgelenks in der Ebene liegt, die durch die Achsen der Gelenke 1 und 2 verläuft. Bildquelle [5].

Die Vermeidung von Singularitäten ist seit vielen Jahren ein wichtiges Thema. Es wurden verschiedene Lösungen vorgeschlagen, von denen sich einige allmählich in Industrierobotern durchsetzen. Je mehr Achsen ein Roboter hat, desto mehr Möglichkeiten gibt es für Singularitäten. Das liegt daran, dass es mehr Achsen gibt, die sich aneinanderreihen können. Zusätzliche Achsen können jedoch auch die Auswirkungen von

Singularitäten verringern, indem sie alternative Positionen ermöglichen, um denselben Punkt zu erreichen. Das wird beispielsweise auch von Kollaborativen Roboterarmen genutzt, die meistens auf einer 7-Achs Kinematik basieren. Durch diesen Freiheitsgrad ist es möglich, dass der Roboterarm um Dinge herum greifen kann.

Singularitäten treten in der Regel auf, wenn die Glieder des Roboters gerade ausgerichtet sind und/oder wenn sich ein Gelenk dem Null-Bereich nähert. Aus diesem Grund können Offsets in die Werkzeuge eingebaut werden, um die Wahrscheinlichkeit zu verringern, dass der Roboter in eine Singularität gerät. Eine weitere Methode besteht darin, die Aufgabe in einen Teil des Arbeitsbereichs zu verlegen, in dem es keine Singularitäten gibt. Dies ist nicht immer möglich, kann aber sehr effektiv sein. Zuletzt hilft die PTP-Bewegung, da sich so problemlos über eine Singularität bewegen kann, da in diesem Fall die mathematische Instabilität der inversen Kinematik einfach übergangen wird. Empfohlen wird, dass der Roboterarm mit einer PTP-Bewegung zu dem untersuchenden Bereich bewegt wird. Dieser Bereich sollte frei von Singularitäten sein, sodass innerhalb auf LIN- oder CIRC- Bewegungen zurückgegriffen werden kann.

Singularitäten sind ein integraler Bestandteil dieses Versuchs. Sofern Sie Schwierigkeiten bei der Erklärung mit dem Zusammenhang zwischen hohen Gelenkgeschwindigkeiten bei einer geringen Endeffektorgeschwindigkeit in der Nähe von Singularitäten haben, hilft evtl. dieses Erklärvideo: <https://www.youtube.com/watch?v=n2nqs8ZW-4c>.

1.2 Der verwendete Roboterarm

Durch die meist sehr hohen Anschaffungskosten von Roboterarmen, die meist zwischen 10.000 € - 100.000 € kosten, werden die Möglichkeiten zur Nutzung in Forschung und Lehre stark begrenzt. Gerade in Praktika werden viele dieser Roboterarmen benötigt. Die Anforderungen an Präzision und Maximalkraft sind dabei weniger wichtig als in der Industrie.

Im Rahmen dieses Praktikumsversuchs wird der 6-Achs-Knickarmroboter Helene (Abbildung 1.8(a)) verwendet, der in einer Bachelorarbeit initial entwickelt und innerhalb von mehreren PEM-Veranstaltungen sukzessive verbessert wurde. Der Roboterarm ist mit Normteilen und handelsüblichen 3D-Druckern aufgebaut. Die Gelenke erhalten dabei eine modulare Elektronik, die für diese Anwendung speziell konstruiert wurde. Angetrieben werden die Gelenke durch Schrittmotoren und eine kombinierte Übersetzung mit Planetengetrieben und Zahnriemen.

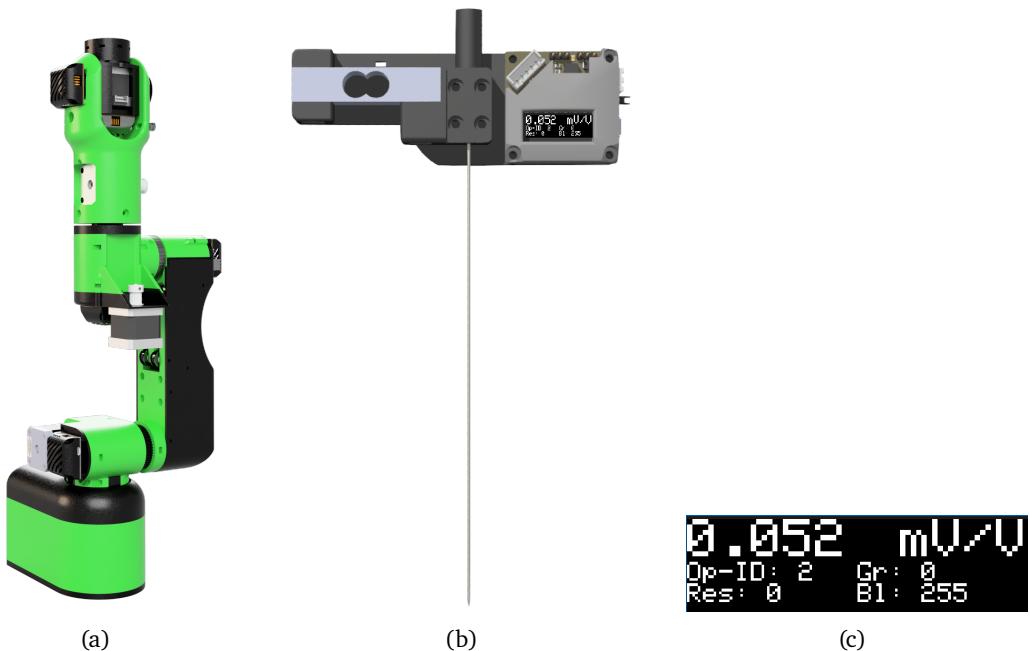


Abbildung 1.8: In diesem Praktikum wird der Roboterarm Helene verwendet (a), an dem als Endeffektor eine sensorische Nadel angebracht wird (b). Der Roboterarm ist somit in der Lage, bei einem Nadeleinstich zu jeder Tiefe die Einstichkraft zu messen. Die Kraft wird dabei in mV/V gemessen, wobei 1 mV/V 50 N entspricht. Die gemessene Kraft wird, neben Statusinformationen, direkt auf dem Bildschirm des Endeffektors angezeigt (c).

Helene kann bei einer maximalen Spannweite von 46 cm (Tabelle 1.2) eine Kraft von mindestens 10 N in alle Bewegungsrichtungen ausüben, je nach Ausrichtung sind auch höhere Kräfte möglich. Die Position wird mit absoluten Magnetencodern überprüft, die auch eine schnelle Inbetriebnahme mit automatischer Ausrichtung während dem Startvorgang ermöglichen [1].

An den Roboterarmen wird während des Versuchs ein Endeffektor angebracht sein. Dieser Endeffektor besteht aus einer Nadel und einer Sensorik, welche die aktuelle Einstichkraft misst und direkt an Roboterarm übergibt (Abbildung 1.8(b)). Die Sensorik ist zweiteilig und besteht sowohl aus der Wägezelle (CZL635-5kg, Tinkerforge,

Tabelle 1.2: Denavit-Hartenberg-Tabelle von Helene.

Gelenk	θ	α_i	a_i	d_i
1	θ_1	-90°	0	135 mm
2	$-90^\circ - \theta_2$	0°	250 mm	0
3	θ_3	-90°	0	0
4	θ_4	90°	0	177 mm
5	θ_5	-90°	0	0
6	θ_6	0°	0	58 mm

Deutschland) als auch aus der Elektronik. Die Elektronik basiert auf einem Microcontroller (ESP32, Espressif Systems, China), welcher die Wägezelle durch das Nutzen eines ADCs (ADS1220, Texas Instruments, USA) ausmisst, die Messdaten filtert, und die gefilterten Messdaten sowohl auf dem integrierten Bildschirm anzeigt (Abbildung 1.8(c)), als auch an den Rest des Roboterarms per CAN mitteilt.

Beim Anschluss der Helene an eine Stromversorgung verfahren die einzelnen Glieder in eine voreingestellte Position (Abbildung 1.9). Diese Ausrichtung wird folgt als Home-Position bezeichnet. Diese Position entspricht dem Frame:

$$Frame_Home = [0.247, 0, 0.345, *(pi, -pi/2, 0)]. \quad (1.7)$$

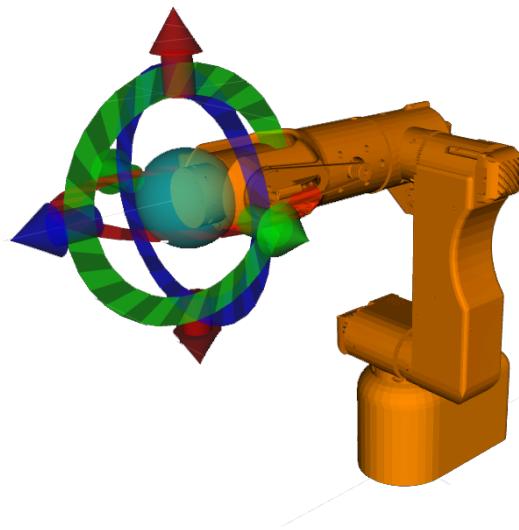


Abbildung 1.9: Die Home-Position wird von Helene beim initialen Verbinden mit der Stromversorgung selbstständig, ohne einen Computer, angefahren.

Sowohl der Roboterarm, als auch der Endeffektor und die Software dazu sind Open-Source und sind auf mehrere Repositories aufgeteilt:

- Messtechnik-Versuch: <https://github.com/SrSupp/MesstechnikPraktikumV4>
- Roboterarm: https://github.com/felixherbst/helene_hardware
- Motorelektronik: https://github.com/SrSupp/Helene_electronics

1.3 Die verwendete Software

Zur Ansteuerung des Roboterarms wird die Linux-Umgebung Ubuntu genutzt, in der das Robot Operating System (ROS) installiert ist. In diesem Kapitel wird nicht nur auf ROS, sondern auch auf die genutzten Bibliotheken eingegangen, die zum Teil eigens für diesen Praktikumsversuch entwickelt wurden. Auch wird auf das Programmierinterface des Roboterarms eingegangen. Vorher werden jedoch einige Linux-Grundlagen thematisiert.

1.3.1 Das Terminal

Ein Terminal, auch Shell oder Kommandozeile genannt, stellt eine textbasierte Ein-/Ausgabe-Schnittstelle für ein Computersystem dar. Auch der Begriff Konsole ist geläufig, bezeichnet aber kein Fenster, sondern einen Bildschirm im Textmodus. Im Terminalfenster können Befehle eingegeben und so das System gesteuert oder Dateien bearbeitet werden (Abbildung 1.10).

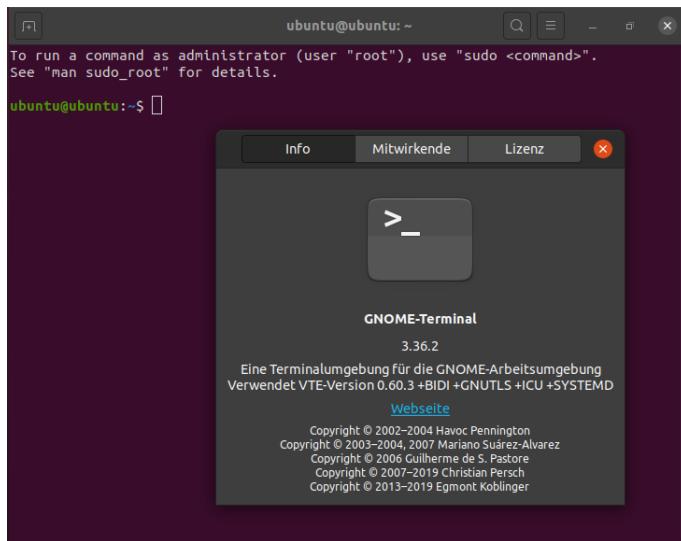


Abbildung 1.10: In Ubuntu kommt das GNOME-Terminal als Terminal zum Einsatz.

Wenngleich Sie in diesem Versuch das Terminal nicht aktiv bedienen müssen, so spielt es zur Bedienung von ROS eine zentrale Rolle. Möchten Sie eine Verbindung zum Roboterarm aufbauen oder die Simulationsumgebung starten, so kommen im Praktikumsversuch Verknüpfungen für Befehle innerhalb des Terminals zum Einsatz. Daher öffnet sich nach dem Klicken auf eine Verknüpfung unmittelbar ein Terminal, das die weiteren Anwendungen automatisiert startet. Das Terminal ist also der Host-Prozess und darf während dem Nutzen der Anwendungen nicht geschlossen werden. Sofern Sie die Verbindung zum Roboterarm ändern wollen, Sie die Simulationsumgebung schließen wollen oder auch das Speichern der Messdaten beenden wollen, so müssen sie nur das zugehörige Terminal-Fenster schließen.

1.3.2 ROS

ROS ist ein Open-Source Software-Framework für Roboteranwendungen. Es ermöglicht durch die Nutzung seiner Vielzahl an Software-Bibliotheken, Werkzeugen und Konventionen die Erstellung robuster und komplexer

plattformunabhängiger Anwendungen für die Robotik. Durch seine freie Verfügbarkeit und Modularität können einzelne Programmteile von seinen Nutzern hinzugefügt, erweitert und flexibel für ihre Anwendungsfälle genutzt und kombiniert werden, wodurch die kollaborative Weiterentwicklung von Robotiksoftware gefördert wird. Im vorliegenden Fall wird ROS Noetic verwendet, das Ubuntu 20.04 benötigt.

Innerhalb von ROS können einzelne Prozesse bestimmte Aufgaben, wie Hardwareabstraktion, reine Datenverarbeitung (Verarbeitung von Sensordaten) oder Bereitstellung von Statusdaten des Roboters, übernehmen und sich über Nachrichten austauschen. Die einzelnen Prozesse/Aufgaben sind in Paketen organisiert. Der Start der Prozesse selbst erfolgt vorwiegend mithilfe sogenannter Launch-Files. Diese ermöglichen sowohl das gleichzeitige Starten mehrerer Prozesse also auch die Übergabe von Parametern beim Start. In diesem Praktikumsveruch wurden die Launch-Files mit den Verknüpfungen in der Startleiste verknüpft, welche dann ein Terminal-Fenster startet.

Den Grundbaustein des Dateisystems stellen die sogenannten Packages dar, in welchen die in ROS genutzten Softwarebausteine organisiert sind. Sie können diverse Elemente wie Datensätze, Bibliotheken, Konfigurationsdateien oder Prozesse enthalten, welche als gemeinsamer Baustein zur Verfügung gestellt werden sollen. Ihnen verdankt ROS somit seine Modularität.

Verbindung zum Roboterarm: Real und per Gazebo

Die Verbindung zum Roboterarm wird automatisiert über die Launch-Files gestartet und ist je nach Aufgabenstellung unterschiedlich.

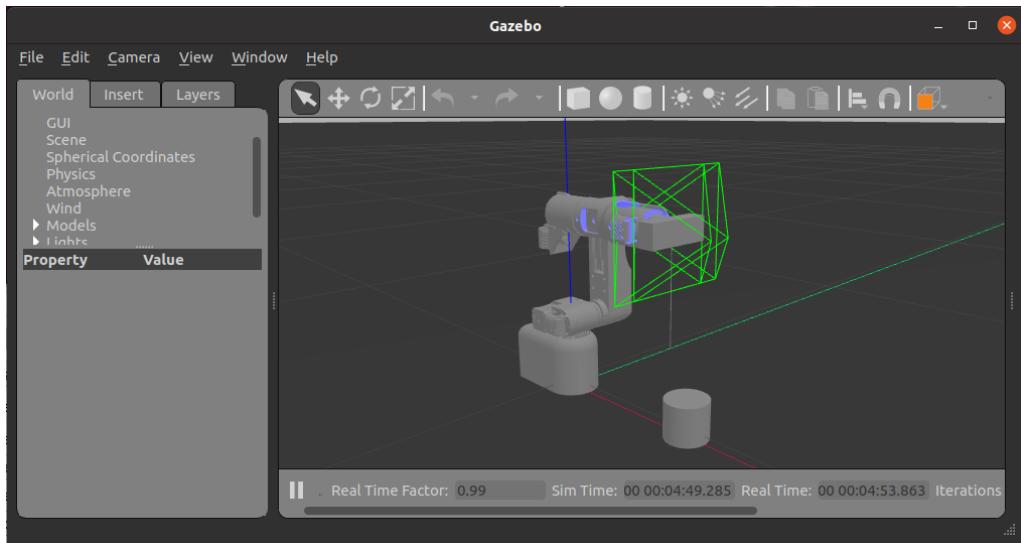


Abbildung 1.11: In Gazebo wird der Roboterarm inklusive Umgebung simuliert.

Für die späteren Aufgaben wird der Roboterarm per USB mit Ihrem Computer verbunden. Sofern Sie sich mit dem Roboterarm und dem zugehörigen Computer verbinden möchten, starten Sie die zugehörige Verknüpfung. Anschließend verbindet sich ihr Computer mit dem Roboterarm und die Messdaten werden in Echtzeit ausgetauscht. Von dem Roboterarm erhält ihr Computer nun die aktuelle Position jedes Gliedes, die aktuelle Winkelgeschwindigkeit jedes Glieder und aktuell gemessene Nadelkraft. Umgekehrt erwartet der Roboterarm eine Soll-Vorgabe der Gelenkgeschwindigkeit, die ihr Computer errechnet und ihn schickt.

Während das für die letzten Aufgaben alles reale Messdaten sind, so wird für die ersten Aufgaben der Roboterarm komplett simuliert, wofür die Software Gazebo zum Einsatz kommt. Gazebo dient also dafür, dass Helene und der Arbeitsraum (bspw. das Tumorgewebe) komplett abgebildet und simuliert wird. Aus dieser Simulationsumgebung errechnet Gazebo physikalisch echt wirkende Messdaten, sodass ihr Computer "denkt", dass eine richtige Helene angeschlossen ist (Abbildung 1.11). Gazebo dient nur zur Visualisierung und Simulation des Roboterarms, kann jedoch keine Steuerbefehle an den Roboterarm senden.

Bewegungsplanung mit MoveIt! und Visualisierung mit RViz

Zur Bahnplanung des Roboters wird das MoveIt! Motion Planning Framework, welches aufbauend auf dem Kommunikationskonzept und Buildsystems von ROS eine Sammlung von Paketen zur Bewegungsplanung und Objektmanipulation bereitstellt, verwendet. Diese enthalten Software zur Pfad- und Trajektorienplanung und der damit einhergehenden Berechnung von Vorwärts- und Inverskinematik, sowie der Kollisionserkennung, 3D Objektwahrnehmung und Navigation.

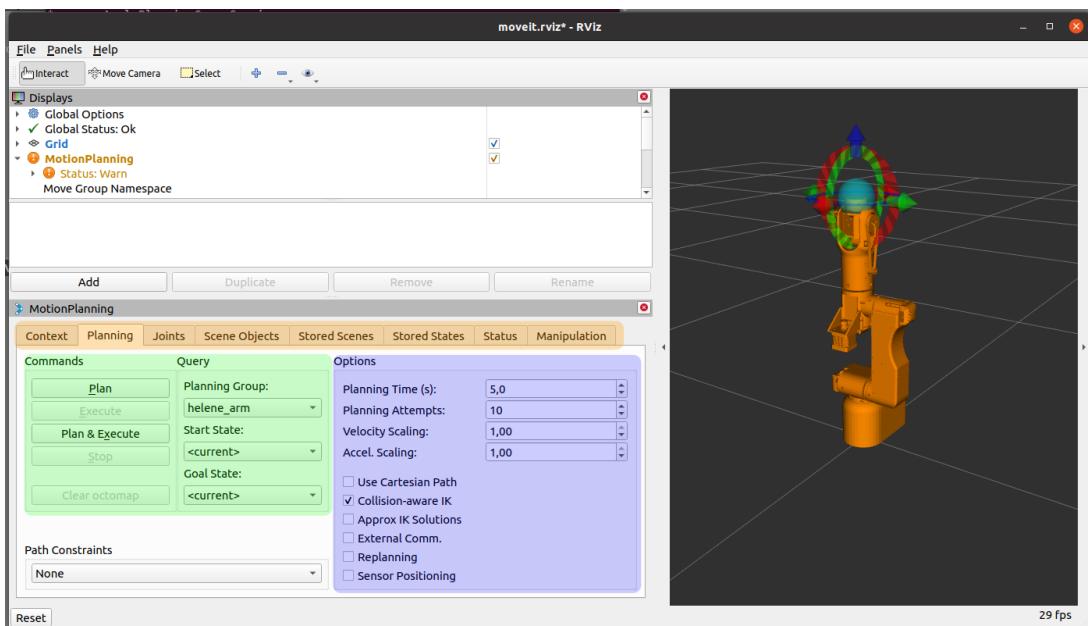


Abbildung 1.12: Beim Start von RViz stehen über das MotionPlanning Plugin Steuermöglichkeiten für Helene zur Verfügung. Im orangenen Bereich kann bei » Joints « die Zielposition des Roboterarms über das Bewegen einzelner Achsen festgelegt werden. Unter dem Reiter » Planning « kann anschließend eine Bewegung an die Zielposition geplant und ausgeführt werden (Plan and Execute, Grün). Unter » Options « (Blau) kann die Bewegung parametrisiert werden, beispielsweise kann die Geschwindigkeit skaliert werden (Velocity Scaling) und es kann verlangt werden, dass die Bewegung des Roboterarms linear ausgeführt wird. Dazu muss bei » Use Cartesian Path « der Haken gesetzt werden. Die Zielposition des Roboterarms kann auch über die Maus im rechten Teil des Bildschirms bewegt werden.

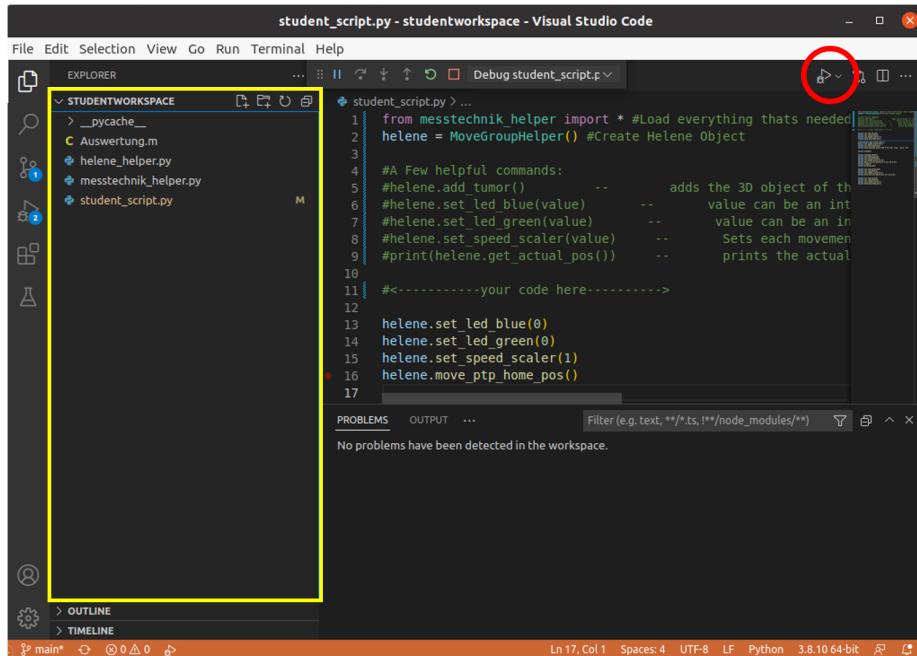
Zentrales Element der MoveIt! Architektur ist der 'movegroup' Prozess, welcher als Bewegungssteuerungsprozess des Roboterarms verstanden werden kann. Dieser dient der Integration verschiedener Informationszweige. Zu diesen gehören unter anderem Informationen bezüglich der aktuellen Lage, Geschwindigkeit und Be-

schleunigung der einzelnen Gelenke des Roboters, Daten externer Sensoren zur Umgebungserfassung, sowie Angaben über vorhandene Transformationen.

Für die Interaktion mit dem Roboter mittels MoveIt! stehen verschiedene Interfaces zur Auswahl. Darunter fallen Anwendungsprogrammierschnittstellen (API) zur Programmierung sowie eine grafische Benutzeroberfläche, das ROS Visualisierungstool (RViz). Mithilfe dieser Interfaces lassen sich Anfragen zur Planung und Ausführung von Bewegungen des Roboterarms an den Bewegungssteuerungsprozess von Helene übergeben. RViz ist ein 3D-Visualisierungstool für Roboter, Sensoren und Algorithmen. Es ermöglicht, die Wahrnehmung des Roboters von seiner (realen oder simulierten) Welt zu sehen und weiterhin über das MotionPlanning Plugin Steuerbefehle an den Roboter zu übergeben (Abbildung 1.12).

Programmierung durch Python

Um die Einarbeitungszeit zu verkürzen und verschiedene Ansteuerungsarten des Roboters vereinheitlicht zu verwenden, wurden die benötigten Funktionen noch weiter abstrahiert und stehen über Python zur Verfügung. Python ist eine Programmiersprache, die dank ihrer klaren Syntax und einfachen Lesbarkeit leicht zu erlernen ist und sich vielseitig einsetzen lässt. In Python Skripten ist am Ende einer Zeile kein Semikolon notwendig. Im Gegensatz zu vielen anderen Sprachen sind die verschiedenen Blöcke nicht durch bestimmte Schlüsselwörter oder Klammern markiert (Bspw. in C bei Schleifen), sondern durch das Einrücken der einzelnen Code-Zeilen. Daher ist unbedingt darauf zu achten, dass keine Zeile Code versehentlich eingerückt ist. Kommentare können im Code per # eingeleitet werden.



```
student_script.py - studentworkspace - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
STUDENTWORKSPACE
> __pycache__
C Auswertung.m
helene_helper.py
messtechnik_helper.py
student_script.py
M
PROBLEMS OUTPUT ...
Filter (e.g. text, **/*.ts, **/node_modules/**)
No problems have been detected in the workspace.

Ln 17, Col 1 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit ⚡ ⌂
```

Abbildung 1.13: Als Entwicklungsumgebung wird Visual Studio Code von Microsoft verwendet. Über den Rot markierten RUN Befehl kann der geschriebene Code ausgeführt werden. In Gelb markiert ist der Workspace, in welchem Sie das Skript student_script.py bearbeiten dürfen.

In diesem Versuch wird als Entwicklungsumgebung Visual Studio Code (VS Code) verwendet. VS Code ist ein kostenloser Quelltext-Editor von Microsoft, der Syntaxhervorhebung, Debugging, Auto vervollständigen

digung und Versionsverwaltung bietet. Durch das Python Plugin sind all diese Funktionen auch für diese Programmiersprache vorhanden (Abbildung 1.13).

Der Workspace, in dem Sie arbeiten werden, liegt unter `/ros_ws/src/MesstechnikPraktikumV6/student-workspace` und beinhaltet das Skript `student_script.py`, das Sie bearbeiten dürfen. Dieses Skript soll am Ende des Versuchs den kompletten Bewegungsablauf für einen automatisierten Nadeleinstich beinhalten. Um dieses Ziel zu erreichen, bietet das Python Interface von Helene folgende Kommandos:

- `helene.set_led_blue(value)`
Setzt die Blaue LED von Helene auf den Wert value. Value muss Ganzzahlig sein und zwischen 0-255 sein. Bei 0 ist die LED ganz aus, bei 255 ganz an. 128 setzt die LED auf eine Helligkeit von 50 %.
- `helene.set_led_green(value)`
Verhält sich wie der Befehl für die Blaue LED. Es wird nur statt der Blauen die Grüne LED gesetzt.
- `helene.set_speed_scaler(speedvalue)`
Der Befehl setzt Global die Bewegungsgeschwindigkeit für Helene fest. Speedvalue kann dabei als Fließkommazahl gesetzt werden (mit einem Punkt als Dezimaltrennzeichen), muss zwischen 0 und 1 sein und gibt relativ, skaliert an der maximalen Geschwindigkeit von Helene, die Bewegungsgeschwindigkeit an. Bspw. wird mit dem Befehl: `helene.set_speed_scaler(0.01)` die maximale Bewegungsgeschwindigkeit von Helene auf 1 % gesetzt.
- `print(helene.get_actual_pos())`
Nach dem ausführen des Befehls wird die aktuelle Position von Helene im Kommandofenster von Visual Studio Code angegeben.
- `helene.sleep(time_in_s)`
Mit diesem Befehl wird das Ausführen der darauffolgenden Zeile für die übergebene Zeit in Sekunden pausiert.
- `helene.probing_start()`
Bitte verwenden Sie diesen Befehl vor dem Einstich in den Tumor. Er signalisiert dem Datenlogger, wann die Messdaten gespeichert werden sollen.
- `helene.probing_end()`
Nach dem Nadeleinstich verwenden Sie bitte diesen Befehl. Er signalisiert dem Datenlogger, wenn die Messdaten nicht mehr gespeichert werden sollen.

Zusätzlich stehen folgende Bewegungsbefehle, die alle auf die vorher gesetzte Geschwindigkeit (`helene.set_speed_scaler(speedvalue)`) achten, für Helene zur Verfügung:

- `helene.move_ptp_home_pos()`
Helene wird über eine PTP-Bewegung an ihren Home-Punkt fahren. Der Home-Punkt wird von Helene selbstständig beim Initialen Hochfahren angefahren und spiegelt eine gute Ausgangslage wieder. Empfohlen ist, sowohl beim Start als auch am Ende eines Programms Helene in diese Home Position zu fahren.
- `helene.move_ptp_abs(Frame_Input)`
Mit dem Befehl wird Helene über eine PTP-Bewegung an die über den Frame_Input) bestimmte Position fahren. Bezogen wird sich dabei auf das Weltkoordinatensystem und die Einheiten sind Meter und Bogenmaß. Mit einer Orientierung von `*(pi, -pi/2, 0)` befindet sich die Nadel in der richtigen Rotation für den Nadeleinstich.

- `helene.move_lin_abs(Frame_Input)`

Wie `helene.move_ptp_abs(Frame_Input)`, jedoch wird eine LIN-Bewegung ausgeführt. In der Nähe von Singularitäten nicht zuverlässig!

Frames können in Python als Array angelegt werden:

$$Frame_Input = [x, y, z, *(pi, -pi/2, 0)]. \quad (1.8)$$

Die Einheiten sind dabei in Meter und Bogenmaß. Der Frame um die home-Position sieht beispielsweise so aus:

$$Frame_Home = [0.247, 0, 0.345, *(pi, -pi/2, 0)]. \quad (1.9)$$

Er beschreibt eine Position bei $x = 0.247$, $y = 0$, $z = 0.345$ mit Orientierung von $*(\pi, -\pi/2, 0)$.

Ein Programm für Helene könnte beispielsweise so aussehen:

```

0 helene.set_led_blue(255) #Blaue LED volle Helligkeit
1 helene.set_led_green(255) #Grüne LED volle Helligkeit
2 helene.set_speed_scaler(0.5) #Geschwindigkeit auf 50%
3 helene.move_ptp_home_pos() #Fahre zur Home-Positon
4 print(helene.get_actual_pos()) #Gib die aktuelle Position aus
5 Frame_Start = [0.25, 0, 0.30, *(pi, pi/2,0)]
6 helene.set_led_green(0) #Grüne LED aus
7 helene.move_ptp_abs(Frame_Start); #Fahre PTP zu Frame_Start
8 Frame_Zwei = [0.25, 0.1, 0.30, *(pi, pi/2,0)]
9 helene.move_lin_abs(Frame_Zwei); #Fahre LIN zu Frame_Zwei, der 10cm neben Frame_Start ist
10 helene.move_ptp_home_pos() #Fahre zur Home-Positon
11 helene.set_led_blue(0) #Blaue LED aus

```

1.3.3 Verknüpfungen zum Starten des Roboterarms

Zum Starten von Programmen innerhalb von ROS kommen Launch-Files zum Einsatz, die als Verknüpfungen in der Startleiste von Ubuntu hinterlegt sind (Abbildung 1.14).

Die einzelnen Launch Verknüpfungen sind diese:

- **Launch: Simulationsumgebung Roboter ohne Nadel**
Startet ein Terminal, welches RViz und Gazebo startet. Helene ist ohne Endeffektor konfiguriert.
- **Launch: Simulationsumgebung Roboter mit Nadel**
Startet ein Terminal, welches RViz und Gazebo startet. Helene hat als Endeffektor die Nadel mit Kraftsensor konfiguriert.
- **Launch: Verbindung zum echten Roboterarm**
Startet ein Terminal, welches zuerst nach dem Namen der zu verbindenden echten Helene fragt. Nach Bestätigung mit der Eingabetaste wird RViz gestartet. Es ist erforderlich beim Wechsel zwischen Simulation und Realität alle Programme neu zu starten. Das inkludiert Visual Studio Code.
- **Launch: Loggen der aktuellen Messdaten**
Startet ein Terminal, welches den Datenlogger startet. Gespeichert wird die aktuelle Position des Roboterarms und die aktuell gemessene Einstichkraft.

Somit öffnet sich nach einem Druck auf eines der Verknüpfungen ein Terminal-Fenster. Schließen Sie dieses Fenster, sofern Sie alle zugehörigen Programme schließen wollen. Je nach Verknüpfung schlägt das Starten der Launch-Files hin und wieder fehl. Versuchen Sie es in diesem Fall einfach erneut.

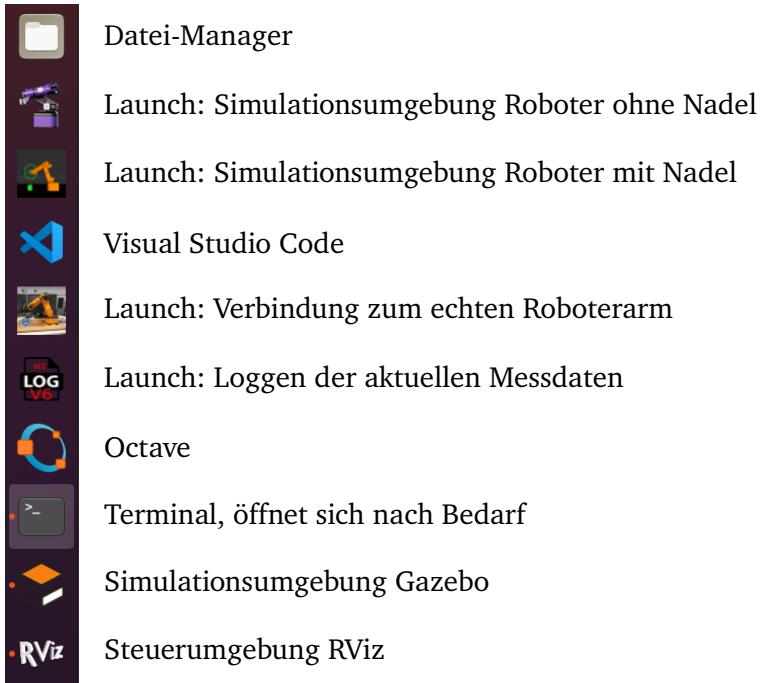


Abbildung 1.14: Screenshot der Verknüpfungen in der Startleiste von Ubuntu. Die Verknüpfungen mit Launch am Anfang starten ein Terminal, welches wiederum die Launch-Files von ROS starten.

1.4 Inhalt des Versuchs

Für diesen Versuch wurden mehrere Gewebephantome aufgebaut, in denen sich ein simulierter Tumor befindet. Das Gewebephantom besteht dabei aus einer Hautschicht, Fettgewebe, Muskelgewebe sowie Lebergewebe. Weiterhin findet sich inmitten des Lebergewebes der Tumor (Abbildung 1.15).

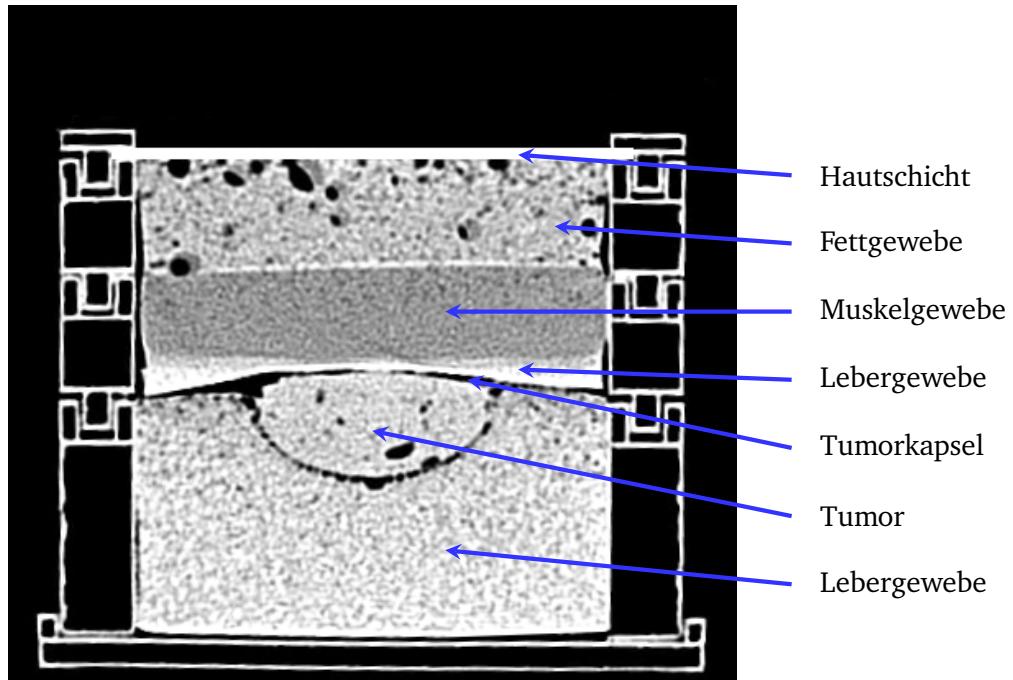


Abbildung 1.15: Ein Schnittbild durch das Gewebephantom für diesen Versuch. Der Tumor befindet sich im Lebergewebe und ist, in Realität, von der Tumorkapsel umhüllt.

Das Gewebephantom befindet sich in unbekanntem Abstand zum Roboterarm. Ziel ist, die Nadel über dem Gewebe zu positionieren und einen Einstich vorzunehmen. Mithilfe der Kraftsensorik soll erkannt werden, in welcher Tiefe sich der Tumor befindet.

2 Vorbereitung

Die Vorbereitungsaufgaben sollen das Verständnis weiter fördern und erfordern eigenständige Recherche. Eine gründliche Vorbereitung ist für diesen Versuch unbedingt notwendig.

2.1 Aufgabe 1.1: Singularität

Was bedeutet Singularität in Bezug auf Roboterarme?

2.2 Aufgabe 1.2: Umgang mit Singularitäten

Sie haben die Aufgabe, einen Roboterarm für das automatisierte Setzen einer Schweißnaht einzurichten. Ihnen fällt jedoch auf, dass bei einer, durch den Schweißvorgang bedingten, linearen Bewegung eine Singularität des Roboterarms durchquert wird. Ist das ein Problem? Wie würden Sie vorgehen, um das Problem zu lösen?

2.3 Aufgabe 1.3: ROS

Was ist der Unterschied zwischen RViz und Gazebo?

2.4 Aufgabe 1.4: Python

Sie werden während der Versuchsdurchführung mit der Programmiersprache Python arbeiten. Um den Umgang zu trainieren können können Sie beispielsweise diesen Online-Interpreter nutzen: <https://www.programiz.com/python-programming/online-compiler/>. Vervollständigen Sie folgenden Code-Ausschnitt, der zwei übergebene Frames addieren kann:

```
0 #Please finish the method "sum_frames", in which two frames are added together.
1 def sum_frames(Input_Frame_a, Input_Frame_b):
2     ausgabe = []
3     #Your code here:
4     #
5     #
6     #
7     return (ausgabe)
8
```

```
9 pi = 3.14159265359
10 Frame_Start = [0.2,0.05,0.3,* (pi,-pi/2,0)]
11 Frame_Offset = [0,0.1,-0.15,* (0,0,0)]
12 Frame_Ziel = sum_frames(Frame_Start,Frame_Offset)
13 print(Frame_Ziel)
```

Sofern Sie den Code kopieren möchten, finden Sie ihn hier: <https://github.com/SrSupp/MesstechnikPraktikum/blob/main/studentworkspace/Vorbereitungsaufgabe.py>

Bei einer korrekten Vervollständigung des Codes erscheint in der Shell:

$$[0.2, 0.15000000000000002, 0.15, 3.14159265359, -1.570796326795, 0]. \quad (2.1)$$

3 Versuchsdurchführung

Im folgenden Abschnitt wird die Versuchsdurchführung beschrieben. Die Aufgaben sind dabei aufeinander aufbauend, sodass diese nacheinander bearbeitet werden sollen und müssen. In Ihrer Dokumentation müssen die fett markierten Fragen schriftlich beantworten werden, ansonsten werden die Teilaufgaben als nicht beantwortet bewertet. Die Fragen finden Sie dabei auch in der Protokollvorlage wieder. In den kommenden Aufgaben werden Sie eine Messungen durchführen, die Sie in der Ausarbeitung zeigen müssen. Zum Speichern der Dateien wird ein USB-Stick benötigt, **den jede Gruppe mitbringen muss**.

Bei diesem Versuch ist es wichtiger denn je, die Grundlagen verinnerlicht zu haben. Lesen Sie den Grundlagen Teil vor der Durchführung gewissenhaft und setzen Sie sich mit den Vorbereitungsaufgaben auseinander. Die Aufgaben sind nicht in Durchführung und Nachbereitung unterteilt. Bitte lesen Sie sich die Aufgaben einmal komplett durch und planen Sie Ihre Vorgehensweise im Team. Ansicht ist der Versuch so ausgelegt, dass keine Nacharbeitung nötig ist.

3.1 Aufgabe 2.1: Starten und Bewegen des Roboterarms

Beginnen Sie damit, die Simulation um den Roboterarm zu starten, indem Sie die Launchdatei für den Roboterarm ohne Endeffektor öffnen (Abbildung 3.1).

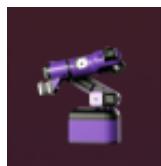


Abbildung 3.1: Die Verknüpfung startet die Simulationsumgebung für den Roboterarm ohne Endeffektor.

Danach startet ein Terminal, welches wiederum RViz und Gazebo startet. Sofern um den Endeffektor in RViz Kreise und Pfeile erscheinen, ist der Startvorgang abgeschlossen. Versuchen Sie, mit der Maus den Endeffektor in RViz zu bewegen und einen Pfad zu dem gewünschten Punkt zu planen und auszuführen (Siehe Abbildung 1.12). Beantworten Sie die folgenden Fragen:

- Ist es möglich, alle Punkte im Arbeitsraum des Roboterarms anzufahren? Welche Punkte sind nicht erreichbar?
- In RViz ist es möglich, den Endeffektor durch Verschieben der Pfeile und der Kreise zu bewegen. Was kann mit den Pfeilen sowie der Kreise verändert werden?

3.2 Aufgabe 2.2: Bewegung über Joint-Winkel

Innerhalb von RViz ist es möglich, die einzelnen Gelenke des Roboterarms zu bewegen. Wechseln Sie dazu auf den Reiter „Joints“ innerhalb von RViz (Siehe Abbildung 1.12, in Orange markiert) und bewegen Sie einzelne Gelenke. Wechseln Sie anschließend wieder auf den Reiter „Planning“ und führen Sie eine Bewegung an Ihre eingebene Position aus.

- Welchem Arbeitsraum entspricht eine Bewegung im Reiter „Joints“ ?

3.3 Aufgabe 2.3: Kartesische und Nicht Kartesische Bewegungen

Weiterhin ist es innerhalb von RViz möglich, den geplanten Pfad zu parametrisieren. Verändern Sie unter dem Options Reiter innerhalb von RViz (Siehe Abbildung 1.12, in Blau markiert) die Optionen „Velocity Scaling“, „Accel. Scaling“ und probieren Sie die Option „Use Cartesian Path“ aus.

- Welchem Bewegungstyp entspricht es, wenn der Haken bei „Use Cartesian Path“ gesetzt ist?
- Welchem Bewegungstyp entspricht es, wenn der Haken bei „Use Cartesian Path“ nicht gesetzt ist?

3.4 Aufgabe 2.4: Bewegungen über Singularitäten

Lassen Sie den simulierten Roboterarm über den „Query“ Reiter innerhalb von RViz (Siehe Abbildung 1.12, in Grün markiert) an die Home-Position fahren.

- Befindet Sich der Roboterarm in der Home-Position innerhalb einer Singularität? Wenn ja, welche?

Bewegen Sie nun den Robterarm von der Home-Postion weg und planen Sie mehrere Bewegungen über die Home-Position (bspw. von Links über die Home Position nach Rechts). Führen Sie diese Bewegungen sowohl kartesisch als auch nicht kartesisch aus.

- Was fällt Ihnen bei kartesischen Bewegungen über die Home-Position auf?
- Warum tritt das Verhalten bei nicht kartesischen Bewegungen nicht auf?

3.5 Aufgabe 2.5: Anbringen des Endeffektors

Schließen Sie das in Aufgabe 2.1 geöffnete Terminal und warten Sie, bis sowohl RViz als auch Gazebo geschlossen wird. Anschließend öffnen Sie die Verknüpfung, welche die Simulationsumgebung um den Roboterarm mit Endeffektor startet (Abbildung 3.2).

Sie werden bemerken, dass der Roboterarm jetzt mit der sensorischen Nadel ausgestattet ist.

- Hat sich durch das Anbringen des Endeffektors der Arbeitsraum des Roboterarms verändert? Wenn ja, wie?

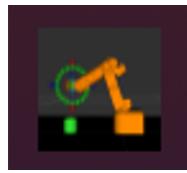


Abbildung 3.2: Die Verknüpfung startet die Simulationsumgebung für den Roboterarm mit Endeffektor.

- Ist es möglich, dass sich der Roboterarm durch geplante Bewegungen mit der Nadel selbst verletzen kann?

3.6 Aufgabe 2.6: Vorbereitungen und Planung des Nadeleinstichs

Nachdem Sie nun die Grundlagen von Roboterarmen verinnerlicht haben und auch die Software um Helene ausgiebig untersucht haben, werden in dieser Aufgabe alle Vorbereitungen um den Nadeleinstich vorgenommen. Da der Nadeleinstich automatisiert vorgenommen werden soll, muss sich jedoch vorher mit der Programmierumgebung auseinandersetzt werden. Öffnen Sie Visual Studio Code (Abbildung 3.3). Sie werden die Datei `student_script.py` editieren.



Abbildung 3.3: Startverknüpfung für Visual Studio Code.

Machen Sie sich mit der Programmierumgebung vertraut, in dem Sie beispielsweise einige Licht-Animation einprogrammieren. Probieren Sie auch einige Bewegungen aus und testen Sie weiterhin die Ausgabe der aktuellen Position des Roboterarms. Die Beleuchtung wird nur in Gazebo angezeigt.

In RViz wird nach einmaligen Ausführen des Codes auch das Gewebephantom abgebildet. Ziel dieser Aufgabe ist, die Koordinaten von diesem Phantom herauszufinden. Dazu gibt es mehrere Möglichkeiten. Entweder kann in der Simulation der Roboterarm so lange bewegt werden, bis er richtig steht und dann kann mit dem Programmierbefehl die Position abgefragt werden. Alternativ kann durch das Nutzen eines Maßbandes die Position vom echten Roboterarm aus gemessen werden. Wie Sie vorgehen wollen ist Ihnen überlassen! Reminder: Das Frame der Home Position ist

$$Frame_Home = [0.247, 0, 0.345, *(pi, -pi/2, 0)]. \quad (3.1)$$

- Für den Nadeleinstich benötigen Sie eine Position, in der der Roboterarm frei von einer Singulatität mit der Nadel über dem Tumor steht. **Ermitteln Sie das Frame für diesen Startpunkt!** (Abbildung 3.4)
- **Weiterhin ermitteln Sie den Frame, der einem vollkommenen Einstich der Nadel entspricht.** Der Frame sollte dabei eine lineare Verschiebung in z-Richtung von Startframe sein (Abbildung 3.4). Auch hier können Sie wieder die Simulation als auch ein Maßband nutzen.

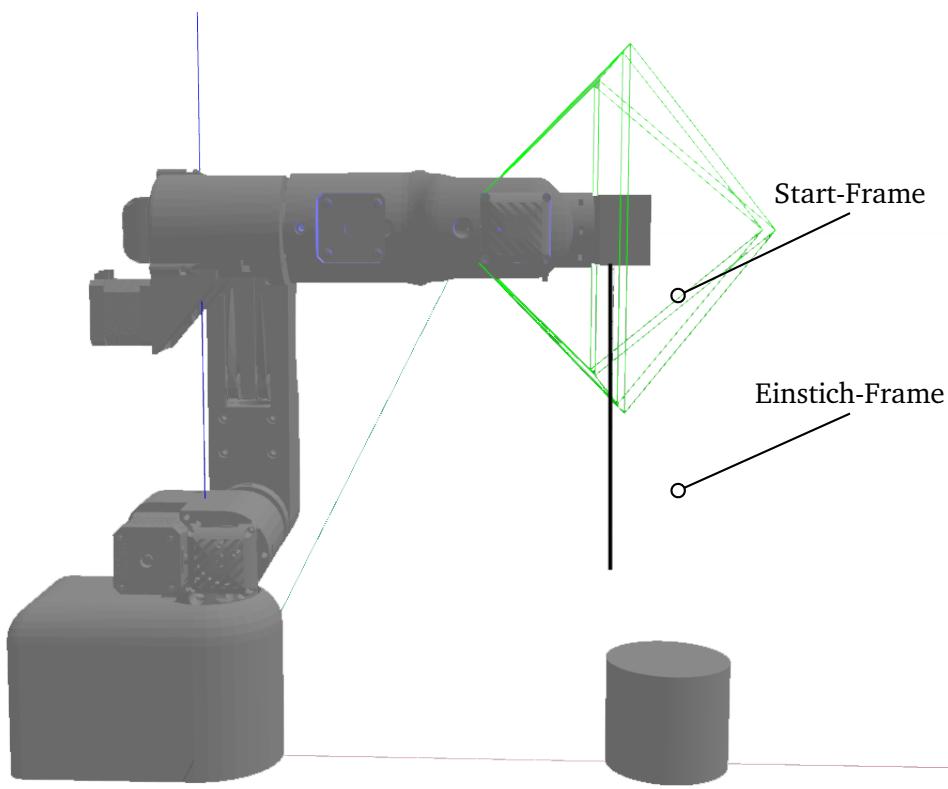


Abbildung 3.4: Skizze der einzelnen Frames (ungefähre Position), die für den Einstich bestimmt werden müssen.

3.7 Aufgabe 2.7: Nadeleinstich

Nach der Vorbereitung für den Einstich wird dieser nun programmiert und anschließend ausgeführt.

- In der vorigen Aufgabe wurde der Frame für den Start und für den Endpunkt des Nadeleinstichs festgestellt. Helene beginnt beim Anfang des Programms immer in der Home Position. **Mit welchem Bewegungstyp würden Sie von der Homeposition zum Startpunkt fahren? Begründen Sie!**
- **Mit welchem Bewegungstyp würden Sie anschließend vom Startpunkt zum Endpunkt des Nadeleinstichs fahren? Begründen Sie!**

Pflegen Sie die Frames in ihr Python Skript ein und definieren Sie die Bewegungen dazwischen. Zu Beginn und zum Ende Ihres Skripts soll Helene die Home-Position anfahren. Der Ablauf sieht also so aus:

- Fahre zu Home-Position.
- Fahre zu Start-Position.
- Vergessen Sie nicht, den Probing_Start Befehl zu setzen! (Siehe Kapitel 1.3.2 und vergessen Sie nicht Klammer auf und zu).
- Fahre zu Einstich-Position. Die Geschwindigkeit des Roboters sollte auf 0.0025 gesetzt werden.

- Vergessen Sie nicht, den Probing_End Befehl zu setzen! (Siehe Kapitel 1.3.2 und vergessen Sie nicht Klammer auf und zu).
- Fahre zu Start-Position. Sie können die Geschwindigkeit dabei gerne erhöhen.
- Fahre zu Home-Position.

Eventuell sind Pausen zwischen den Bewegungen (mit der Sleep-Funktion) sinnvoll. Nutzen Sie auch die LEDs von Helene, um zu signalisieren, in welchem Bewegungsvorgang Helene gerade ist. Das hilft Ihnen nicht nur beim Debuggen, sondern sieht auch cool aus. Führen Sie diesen Code iterativ so häufig aus, bis Helene einen fehlerfreien Nadeleinstich innerhalb der Simulation ausführt. Vergessen Sie nicht, den Source Code auf Ihren USB-Sticks zu speichern. **In der Auswertung müssen Sie diesen darstellen.**

- Zeigen Sie ihren erstellten Python-Code.

Sofern Sie mit dieser Aufgabe fertig sind, zeigen Sie einem Tutor die Simulation. Sie erhalten anschließend eine echte Helene, mit der Sie den von Ihnen geschriebenen Code ausführen können. Beiliegend finden Sie weiterhin Schutzbrillen, die Sie unbedingt anziehen müssen, da Helene mit der medizinischen Nadel ausgestattet ist. **Bei Missachtung der Schutzbrillenpflicht werden Sie umgehend des Raumes verwiesen.**

Schließen Sie alle geöffneten Terminals, um die Simulation zu beenden. Anschließend verbinden Sie Ihre Helene mit dem Netzteil und warten Sie, bis sie sich in der Home-Position befindet. Daraufhin können Sie Helene mit dem Computer über ein Micro-USB verbinden und die Verknüpfung starten, welche RViz öffnet und die Verbindung zu Helene herstellt (Abbildung 3.5).



Abbildung 3.5: Startverknüpfung für das Verbinden mit einer Helene über USB.

Sobald RViz vollständig gestartet ist (Ersichtlich an dem bunten Kreis um Achse 6), haben Sie einige Zeit sich mit dem Roboterarm vertraut zu machen. Fahren Sie beispielsweise über eine Singularität und beobachten Sie, wie sich der reale Roboterarm verhält. Weiterhin geben Sie manuell eine Kraft über die Nadel an die Kraftsensorik und beobachten Sie die gemessenen Kräfte am Bildschirm des Endeffektors. Nachdem Sie ein Gefühl für den Roboterarm haben, starten Sie den Datalogger, der die Messdaten während des Nadeleinstichs sichert (Abbildung 3.6)

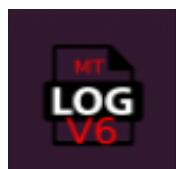


Abbildung 3.6: Startverknüpfung für das Starten des Datenloggers.

Sobald im gestarteten Terminalfenster signalisiert wird, dass das Loggen begonnen hat, starten Sie ihr vorher vorbereitetes Skript über Visual Studio Code und führen Sie einen Nadeleinstich durch. Nach Vollendigung des Skripts schließen Sie das für den Datenlogger nötige Terminalfenster.

- Fiel Ihnen bei dem realen Nadeleinstich eine Abweichung des Verhaltens vom Roboterarm gegenüber der Simulation auf?

3.8 Aufgabe 2.8: Interpretation der Messwerte

Nachdem Sie die Messdaten aufgenommen haben, trennen Sie die Verbindung mit dem Roboterarm, indem Sie zuerst das USB Kabel und anschließend die Stromversorgung trennen. So haben auch andere Gruppen die Chance mit diesem Roboterarm zu arbeiten.

Zur Interpretation der Messdaten öffnen Sie Octave (Abbildung 3.7) und öffnen Sie das Skript Auswertung.m.



Abbildung 3.7: Startverknüpfung für das Starten von Octave.

Zum Starten des Skripts drücken Sie F5. Es erscheint ein Graph, der die aktuelle Nadeleinstichtiefe gegenüber der gemessenen Kraft anzeigt. Speichern Sie den Graph sowie die aufgenommenen .csv Rohdaten aus der vorigen Aufgabe auf Ihrem USB-Stick.

Sofern Sie den Versuch Online bewältigen, können Sie die aufgenommenen Messdaten entweder über das Github des Versuchs oder über das Moodle Forum herunterladen. Das Auswerteskript wurde zusätzlich für Matlab erstellt und liegt, genau so wie das Octave Skript, auch im Github des Versuchs. <https://github.com/SrSupp/MesstechnikPraktikumV4/blob/main/studentworkspace/>. Erstellen Sie den Plot in jedem Fall selbst! Wenn Sie das Auswerteskript starten, müssen die Rohmessdaten im gleichen Ordner wie das Skript liegen. Im Skript wird die gemessene Position des Roboterarms tiefpassgefiltert. Die Cutoff-Frequenz beträgt dabei 5 Hz und wird über $fc = 5$; gesetzt. Variieren und experimentieren Sie gerne auch mit dieser Frequenz.

- Erklären Sie mit eigenen Worten, was der Graph darstellt.
- Auf dem Graph müsste erkennbar sein, in welcher Einstichtiefe die Tumorkapsel durchquert wird. In welcher Einstichtiefe befindet sich die Nadel im Tumor? Gemeint ist die Distanz zwischen Oberfläche des Gewebephantoms und des Durchquerens der Tumorkapsel.
- Können Sie weitere Gewebebeschichten ausmachen?

Sie sind nun fertig mit dem 6. Versuch und können gerne die Zeit bis zum Ende für die Ausarbeitung oder weitere eigene Bewegungen mit dem Roboterarm nutzen. Sofern Sie den Computer herunterfahren möchten, nutzen Sie den **Ausschaltkopf in Ubuntu, der sich oben rechts am Bildschirm befindet**. Schalten die den Strom erst ab, wenn der Computer vollständig heruntergefahren ist. Andernfalls beschädigen Sie das Dateisystem der Ubuntu-Installation.

Literatur

- [1] F. Herbst, „Low-Cost Knickarmroboter für die Mensch-Maschine-Schnittstelle,“ Bachelorarbeit, Technische Universität Darmstadt, Darmstadt, 23. September 2019.
- [2] M. W. Spong, *Robot Modeling and Control, Second Edition*. Wiley-Blackwell, 2020, ISBN: 978-1-119-52399-4.
- [3] K. Fey, „Optimierung von Umkehrspiel und Steifigkeit eines additiv gefertigten Roboterarms,“ Bachelorarbeit, Technische Universität Darmstadt, Darmstadt, 8. April 2022.
- [4] L. V. u. G. O. Bruno Siciliano Lorenzo Sciavicco, *Robotics: Modelling, planning and control*. Springer, 2010, ISBN: 978-1-84628-641-4.
- [5] E. Ilian Bonev Ph.D. „What are Singularities in a Six-Axis Robot Arm?“ Mecademic. (), Adresse: <https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm> (besucht am 10.05.2022).