NTB
Interstaatliche Hochschule
für Technik Buchs
FHO Fachhochschule Ostschweiz

Kinematik Delta Roboter

Ergänzende Veranstaltung Vertiefungsprojekt 2

Manuel Ilg

August 2017

Inhaltsverzeichnis

1 Einleitung

In diesem Dokument wird ein Algorithmus für die Direkt-Kinematik eines Delta-Roboters, auch Parallel-Roboter genannt, vorgestellt. Zusätzlich zur Direkt-Kinematik kann der Algorithmus auch die Winkel von jedem Gelenk berechnen. Diese Winkel können dann in einem weiteren Schritt verwendet werden um den Zustand, später bezeichnet mit Pose, des Roboters zu visualisieren.

1.1 Motivation

Der Algorithmus soll als Zwischenglied in einem Roboter-Projekt eingesetzt werden. In diesem Roboter-Projekt muss ein Delta-Roboter visualisiert werden. Die Informationen für die Visualisierung stammen dabei von einem realen oder einem simulierten Roboter.

Genauer wird der Algorithmus zwischen die Simulations-Umgebung "Gazebo" und das Visualisierungs-Tool "rviz" geschaltet. Oder zwischen den realen Roboter und das Visualisierungs-Tool.

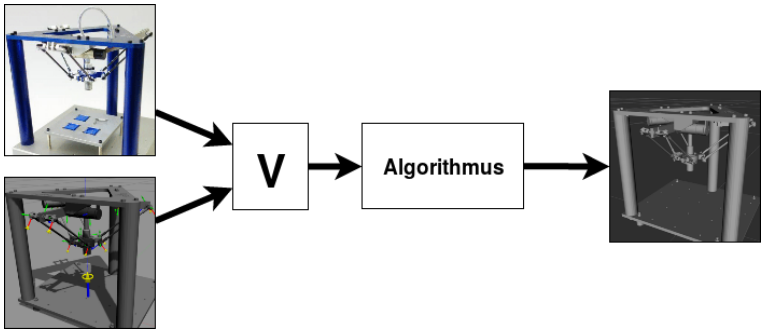

Abbildung 1.1: Informationsfluss

Was gibt es schon?

Für das Teilproblem Direkt-Kinematik gibt es schon ein Algorithmus für den EEDURO-Delta[1]. Jedoch hat dieser zwei grosse Schwächen. Die ganze Direkt-Kinematik wurde in einer einzigen Methode implementiert. Und es wurden keine sinnige Benennung von Variablen verwendet. Diese beiden Schwächen führen dazu, dass der Code sehr schwer verständlich ist.

Auch gibt es einen Ansatz, das Teilproblem Direkt-Kinematik mit einem Gleichungssystem zu lösen. Dieser Ansatz wird in der Masterarbeit *Education Robot Platform*[2] beschrieben. Um den Algorithmus mit diesem Ansatz umsetzen zu können, muss das Gleichungssystem symbolisch aufgelöst werden. So kann dann die symbolische Lösung in eine Programmiersprache wie C++ überführt werden. Die symbolische Lösung dieses Gleichungssystems war aber so riesige, dass es nicht Sinnvoll war diese umzusetzen. Der MATLAB-Code mit dem Gleichungssystem und der symbolischen Lösung ist im Repository *eeduro_delta* [1] zufinden.

Aufgrund der oben genannten Schwächen der schon vorhanden Lösungen bzw. Ansätzen wurde entschieden, auch das Teilproblem Direkt-Kinematik neu umzusetzen, zusammen mit dem anderen Teilproblem, der Berechnung der Pose des Roboters.

[1] `https://github.com/manuelilg/eeduro_delta/tree/master/eeduro_delta_joint_state_publisher/matlab`

2 Umsetzung

Die Umsetzung der Aufgabe wurde in zwei Schritten vollzogen. In einem ersten Schritt wurde das Konzept erarbeitet. In der Konzeptphase wurde entschieden das Problem mit eine geometrische Herangehensweise zu lösen. Das Konzept wurde in MATLAB umgesetzt und auch validiert.

In einem zweiten Schritt wurde dann das Konzept in ein C++ Programm überführt.

2.1 Vereinfachungen

Das Model vom Delta-Roboter wurde in zwei Stufen vereinfacht. In der ersten Vereinfachung wurden das Gestänge bestehen aus vier Links (Link 2, Link 3-1, Link 3-2 und Link 4) durch einen Link (nachfolgend im Dokument Link 3 genannt) ersetzt. In der zweiten Vereinfachung wurden die Arm nach innen verschoben, so das sich die Enden der Arme direkt treffen.

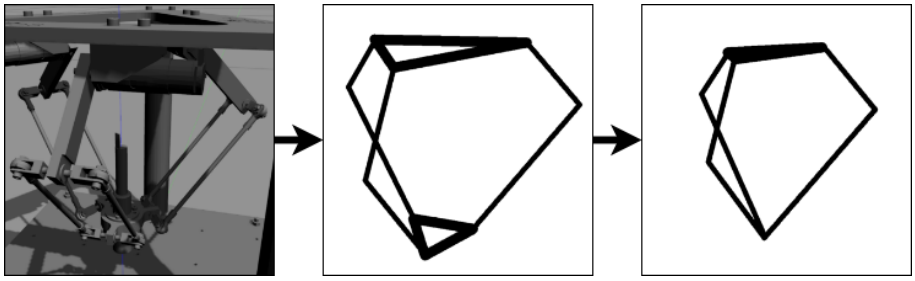

Abbildung 2.1: Vereinfachungsschritte

3 Konzept

In diesem Kapitel wird das Konzept des Algorithmus vorgestellt. Das Konzept basiert dabei auf einem geometrischen Ansatz.

Die erste Teilaufgabe, die Berechnung der Direkt-Kinematik eines Delta-Roboters, kann auf das Berechnen von Schnittpunkten für drei Sphären reduziert werden. Durch die geometrischen Gegebenheiten des Delta-Roboters schneiden sich die drei Sphären immer. Einer der beiden Schnittpunkte ist der Werkzeugmittelpunkt den wir suchen.

Die zweite Teilaufgabe kann auf das Berechnen von Winkeln zwischen Vektoren reduzierte werden.

3.1 Notation

Zur Klarstellung wird hier die Notation für die Variablen vorgestellt die in den Gleichungen verwendet wird:

- Skalar: a_x
  - Distanz X zu Y: d_{XtoY}
  - Radius von Sphäre: r_S
  - Radius von Kreis (Circle) 1: r_{C1}
  - Winkel α von Arm 1: α_1
  - Länge von Link 1: len_1
- Vektor: $\vec{a}_x$
  - Normale von Kreis (Circle) 1: $\vec{n}_{C1}$
  - Normale von Ebene (Plane) 1: $\vec{n}_{P1}$
  - Link 1 von Arm 2: $\vec{l}_{1_2}$
  - Link Projektion von Arm 1: $\vec{l}_{p_1}$
  - Vektor der X mit Y verbindet: $\vec{d}_{XtoY}$
- Punkt/Ortsvektor: $\vec{A}_x$
  - Zentrum (Center) Sphäre 1: $\vec{C}_{S1}$
  - Zentrum Kreis 2: $\vec{C}_{C2}$
  - Stützvektor von Ebene (Plane) 1: $\vec{P}_{P1}$

3.2 Direkt-Kinematik

Das Problem der drei sich schneidenden Sphären, sowie berechnen ihrer beiden Schnittpunkte, kann in folgende Teilschritte unterteilt werden.

1. Schnitt zwischen 1. und 2. Sphären -> Kreis 1
2. Kreis beschreibt Ebene -> Ebene
3. Schnitt zwischen Ebene und 3. Sphäre -> Kreis 2
4. Schnitt zwischen Kreis 1 und Kreis 2 -> 2 Schnittpunkte

Input für den Algorithmus sind die Stellungen der Oberarme, auch genannt Link 1, des Delta-Roboters. Die Stellungen werden über Winkel beschrieben. Somit erhält der Algorithmus drei Winkel als Input, für jeden Arm einen Winkel α_n.

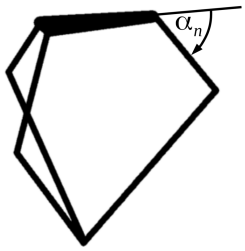

Abbildung 3.1: Input α

Position von den drei Sphären

Die Zentren der drei Sphären liegen jeweils am Ende des Link 1 von den drei Armen. Die Berechnungen für den Link 1 sind für all drei Arme gleich. Um dies zu verdeutlichen wird die Variable n in den Berechnungen verwendet. n steht dabei für die Nummer des Armes.

$$\vec{l}_{1_n} = Rot_z\left((n-1)\cdot\frac{2\cdot\pi}{3}\right)\cdot\begin{bmatrix}0\\-cos(\alpha_n)\cdot len_1\\sin(\alpha_n)\cdot len_1\end{bmatrix} \tag{3.1}$$

$$\vec{C}_{Sn} = \vec{l}_{1_n} \tag{3.2}$$

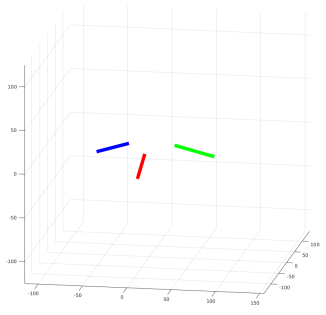

Abbildung 3.2: Link 1 von jedem Arm

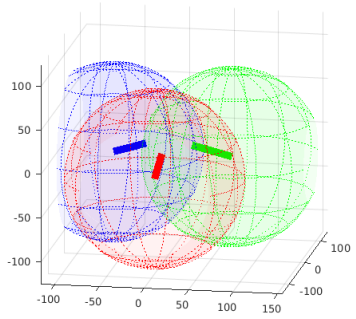

Abbildung 3.3: 3 Sphären

Schnitt zweier Sphären

Wenn sich zwei Sphären schneiden ist das Ergebnis eine Kreis. Da die beiden Sphären den gleiche grossen Radius haben liegt der Mittelpunkt des Kreises genau in der Mitte der beiden Mittelpunkte der Sphären. Somit kann der Mittelpunkt des Schnittkreises folgendermassen berechnet werden:

$$\vec{C}_{C1} = \vec{C}_{S1} + \frac{\vec{C}_{S2} - \vec{C}_{S1}}{2} = \frac{\vec{C}_{S1} + \vec{C}_{S2}}{2} \tag{3.3}$$

Der Radius des Schnittkreises:

$$r_{C1} = \sqrt{r_S^2 - \left(\frac{\left|\vec{C}_{S2} - \vec{C}_{S1}\right|}{2}\right)} \tag{3.4}$$

Um den Kreis im 3D-Raum definieren zu können fehlt noch eine Beschreibung für die Orientierung des Kreises im Raum. Gewählt wird einen Normalenvektor zur Beschreibung:

$$\vec{n}_{C1} = \frac{\vec{C}_{S2} - \vec{C}_{S1}}{\left|\vec{C}_{S2} - \vec{C}_{S1}\right|} \tag{3.5}$$

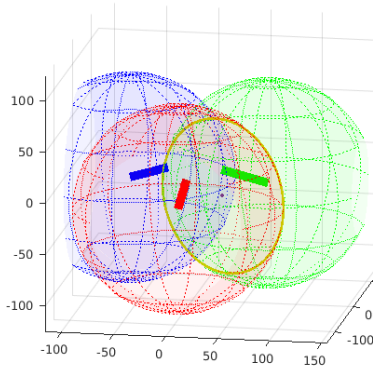

Abbildung 3.4: Schnitt 2 Sphären

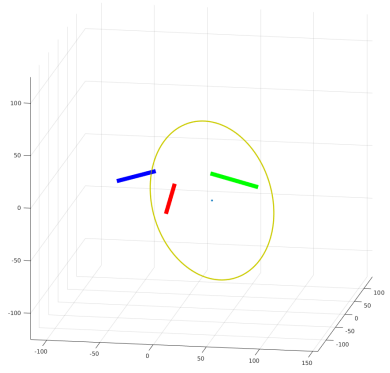

Abbildung 3.5: Schnittkreis 1

Kreis zu Ebene

Die Ebene, auf der der Kreis 1 liegt, kann einfach mit den Parametern des Kreises beschrieben werden.

$$\vec{P}_P = \vec{n}_{C1} \tag{3.6}$$
$$\vec{n}_P = \vec{n}_{C1} \tag{3.7}$$

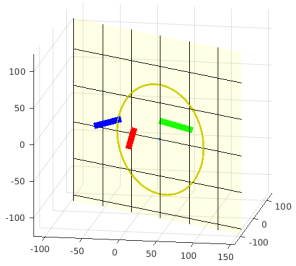

Abbildung 3.6: Ebene von Schnittkreis

Schnitt zwischen Ebene und Sphäre

Wenn sich eine Ebene und eine Sphäre schneiden gibt es einen Schnittkreis. Dieser Kreis kann wieder beschrieben werden mit folgenden drei Parametern: Kreismittelpunkt $\vec{C}_{C2}$, Radius r_{C2} und Normalenvektor $\vec{n}_{C2}$.

Für die Berechnung benötigen wir den kürzesten Abstand zwischen dem Sphärenmittelpunkt und der Ebene:

$$d_{PtoC} = \left(\vec{P}_P - \vec{C}_{S3}\right) \bullet |\vec{n}_P| \tag{3.8}$$

Nun können wir die Parameter des Schnittkreises berechnen.

$$\vec{C}_{C2} = \vec{C}_{S3} + d_{PtoC} \cdot \vec{n}_P \tag{3.9}$$
$$r_{C2} = \sqrt{r_S^2 - d_{PtoC}^2} \tag{3.10}$$
$$\vec{n}_{C2} = \vec{n}_P \tag{3.11}$$
$$\tag{3.12}$$

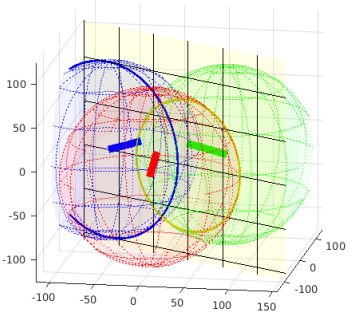

Abbildung 3.7: Schnitt: Ebene & Sphäre

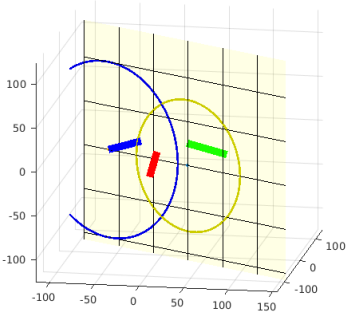

Abbildung 3.8: Schnittkreis 2

Schnitt zweier Kreise

Die Schnittpunkte der zwei Schnittkreise könne mit Hilfe der Potenzgeraden bestimmt werden. Denn wenn sich zwei Kreise unterschiedlicher Grösse schneiden, geht die Potenzgerade durch die beiden Schnittpunkte.

Der Vektor der die beiden Kreismittelpunkte verbindet lautet:

$$\vec{d}_{C1toC2} = \vec{C}_{C2} - \vec{C}_{C1} \tag{3.13}$$

Der Abstand der Potenzgeraden vom Kreismittelpunkt $\vec{C}_{C1}$ ist gegeben durch:

$$d_1 = \frac{\left|\vec{d}_{C1toC2}\right|^2 + {r_{C1}}^2 - {r_{C2}}^2}{2\left|\vec{d}_{C1toC2}\right|} \tag{3.14}$$

Der Abstand von der Strecke, die die beiden Kreismittelpunkte verbindet zu den Schnittpunkten, kann folgend bestimmt werden:

$$h = \sqrt{{r_1}^2 - {d_1}^2} \tag{3.15}$$

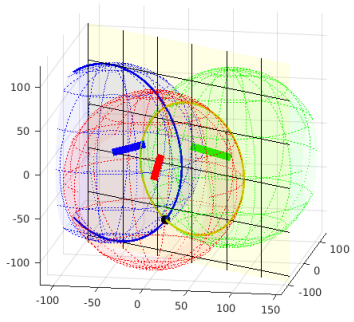

Abbildung 3.9: Schnitt 2 Kreise

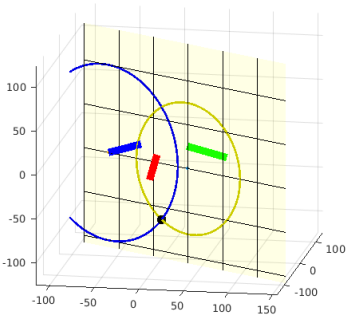

Abbildung 3.10: Schnittpunkt

Werkzeugmittelpunkt

Der Werkzeugmittelpunkt (auf englisch "tool center point"genannt und abgekürzt tcp) kann nun bestimmt werden. Wir müssen dazu denjenigen Schnittpunkt auswählen, der in z-Richtung gesehen weiter unten ist. Dafür benötigen wir einen Vektor der folgende Eigenschaften erfüllt:

- Orthogonal zum Vektor der die beiden Kreismittelpunkte verbindet
- Liegt auf der Ebene der beiden Kreise
- z-Komponente des Vektors ist negativ

Diesen Vektor erhalten wir mit folgender Berechnung:

$$\vec{v}_h = \vec{d}_{C1toC2} \times \vec{n}_P \tag{3.16}$$

Nun können wir den Werkzeugmittelpunkt berechnen:

$$\overrightarrow{TCP} = \vec{C}_{C1} + d_1 \cdot \frac{\vec{d}_{C1toC2}}{\left|\vec{d}_{C1toC2}\right|} + h \cdot \frac{\vec{v}_h}{|\vec{v}_h|} \tag{3.17}$$

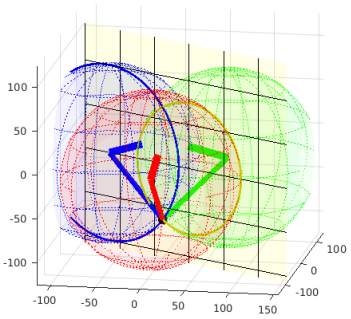

Abbildung 3.11: Lösung mit Sphären

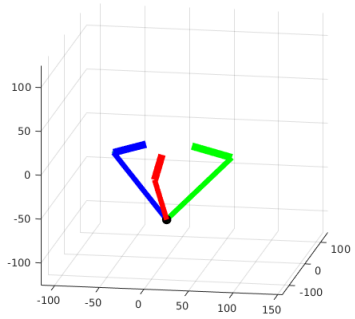

Abbildung 3.12: Lösung

3.3 Pose von Roboter

Um die Pose des Roboters beschreiben zu können, müssen die Winkel β und γ für jeden Arm bestimmt werden. Speziell für den 1.Arm muss noch der Winkel δ berechnet werden.

Die Berechnungen für die beiden Winkel β und γ sind für alle Arme gleich. Um dies zu verdeutlichen, wird der Index n in den Berechnungen verwendet.

Um α zu berechnen muss der Link 2 auf eine Ebene projiziert werden. Diese Ebene ist parallel zur z-Achse und der jeweilige Link 1 liegt auf ihr. Der Link 1 wird durch den Vektor $\vec{l}_{1_n}$ dargestellt.

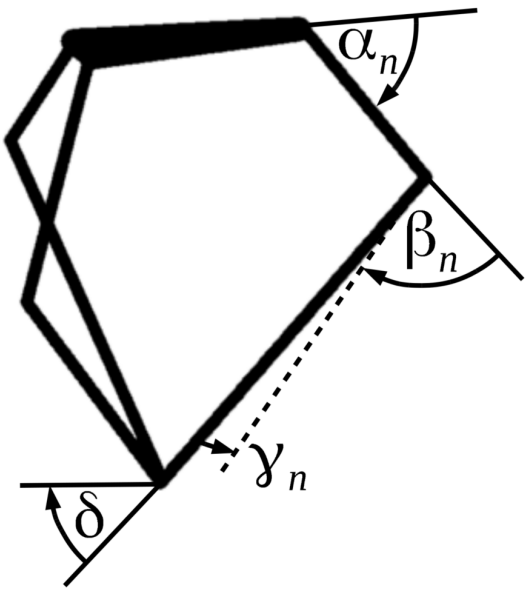

Abbildung 3.13: Winkeldefinitionen

Projektions-Ebene

Die Projektions-Ebene kann durch einen Punkt $\vec{P}_n$ und eine Normalenvektor $\vec{n}_n$ beschrieben werden.

$$\vec{P}_n = \vec{C}_{S_n} \tag{3.18}$$
$$R_{\alpha_n} = Rot_x(-\alpha_n) \tag{3.19}$$
$$R_{z_n} = Rot_z\left((n-1)\cdot\frac{2\cdot\pi}{3}\right) \tag{3.20}$$
$$\vec{n}_{P_n} = \vec{l}_{1_n} \times R_{z_n} \cdot R_{\alpha_n} \cdot \vec{e}_z \tag{3.21}$$

Projektion Link 2

Für jeden Arm kann der Link 2 mit folgender Berechnung auf die jeweilige Ebene projiziert werden.

$$\vec{l}_{p_n} = \vec{l}_{2_n} - \vec{n}_{P_n} \cdot \left(\vec{l}_{2_n} \bullet \vec{n}_{P_n}\right) \tag{3.22}$$

Berechnung Winkel β

Der Winkel β, ist der zwischen dem Link 1 und der Projektion von Link 2.

$$\beta_n = acos\left(\frac{\vec{l}_{1_n} \bullet \vec{l}_{p_n}}{\left|\vec{l}_{1_n}\right| \cdot \left|\vec{l}_{p_n}\right|}\right) \tag{3.23}$$

Berechnung Winkel γ

$$\gamma_n = acos\left(\frac{\vec{l}_{3_n} \bullet \vec{l}_{p_n}}{\left|\vec{l}_{3_n}\right| \cdot \left|\vec{l}_{p_n}\right|}\right) \tag{3.24}$$

Berechnung Winkel δ

Der Winkel zwischen dem Link 3 und dem Werkzeug ist δ. Da das Werkzeug immer die gleiche Orientierung hat, wie der Rahmen des Roboters, kann er aus den Winkeln α und β berechnet werden.

$$\delta = \pi - \alpha_1 - \beta_1 \tag{3.25}$$

3.4 Validation Konzept

Die Validation des Konzeptes wurde in zwei Schritten realisiert.

Der erste Schritt der Validation wurde während der Entwicklung des Konzeptes durchgeführt. Da das Konzept in MATLAB erstellt wurde, konnte jeder Teilschritt validiert werden. Denn für jeden Teilschritt wurden die berechneten Punkte und Vektoren, in einem gemeinsamen Plot, zusammen geführt. So konnte im Plot einfach die Korrektheit jedes Teilschrittes überprüft werden.

Im zweiten Schritt der Validation wurde das fertige Konzept getestet. Dabei wurde der Algorithmus mit verschiedenen Armstellungen getestet, um sicherzustellen, dass er im ganzen Arbeitsraum des Roboters stabil ist.

4 Umsetzung in C++

Damit der Algorithmus in einer ROS-Umgebung verwenden werden kann, musste dieser in C++ umgesetzt werden. Für die Umsetzung in C++ wurde die Library Ëigen"verwendet. Eigen ist für lineare Algebra, Matrix und Vektor Operationen geeignet.

Das fertige C++ Programm besteht aus einem main-Teil und zwei Klassen. Der main-Teil startet den ROS-Node, instantiert die Klasse EEDuroDeltaJointStatePublisher und startet die Kommunikation.

In der Klasse EEDuroDeltaJointStatePublisher werden die ROS-Kommunikations-Kanäle für Subscribe und Publish eröffnet. Auch wird eine Instanz der Klasse DeltaKinematic erstellt.

Die Klasse DeltaKinematic setzt den Algorithmus um, der im Kapitel 3 beschrieben wurde. Für die Berechnung des ToolCenterPoints muss die Methode calculateForwardKinematic angesprochen werden. Diese nimmt drei Winkel als Parameter entgegen. Die Winkel beschreiben die Stellung des Link 1 für jeden der drei Arme.

Alle Informationen für die Beschreibung der Pose des Roboters erhält man mit der Ausführung der Methode calculatePose.

Diese Methoden werden von der Klasse EEDuroDeltaJointStatePublisher mit den Informationen, die sie von dem Subscriber-Kanal erhät, aufgerufen. Die berechneten Werte werden dann über den Publish-Kanal gesendet. Somit werden die Daten allen Programmen zur Verfügung gestellt, die zu diesem Kanal subscriben.

Der Komplette Source-Code des Programms ist im Repository *eeduro_delta* [1] einsehbar.

4.1 Validation C++ Programm

Der in C++ umgesetzte Algorithmus konnte dann auch wieder mit einer visuellen Prüfung validiert werden.

Dabei wurde der Algorithmus zwischen die Simulations-Umgebung "Gazebo" und das Visualisierungstool "rviz" geschaltet. Danach wurde die Simulation gestartet und die beiden Anzeigen beobachtet. Die Beobachtung zeigte, dass beide Anzeigen zu jeder Zeit das Gleiche darstellten. Somit konnte die Korrektheit der Umsetzung in C++ bestätigt werden.

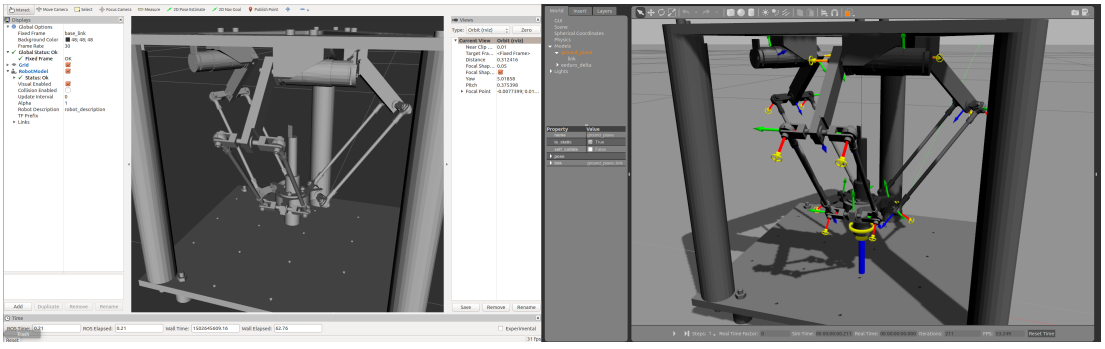

Abbildung 4.1: links rviz, rechts Gazebo

[1] https://github.com/manuelilg/eeduro_delta/tree/master/eeduro_delta_joint_state_publisher

5 Ergebnisse, Fazit und Ausblick

5.1 Ergebnisse

Mit dem fertigen Programm kann nun einfach die Pose des Roboters mit nur drei Winkelangaben dargestellt werden. Diese Winkel können wahlweise von einem realen Roboter stammen, oder von einer Simulation. Somit kann die Pose des realen oder simulierten Roboters widergespiegelt werden.

5.2 Fazit

Mir machte die Umsetzung dieses Projektes Spass, da ich mein Gelerntes im Bereich Roboter Kinematik umsetzen konnte. Auch hat viel zur Motivation beigetragen, dass das Ergebnis in einem effektiven Projekt benutzt wird und nicht nur ein Übungsprojekt bleibt.

Auch habe ich gelernt, wie man einen schulischen Ansatz (MATLAB) in ein produktives Umfeld (C++) überführt.

5.3 Ausblick

Die Code-Basis bietet eine gute Grundlage um weitere Funktionen einzubauen. So z.B. könnte die Klasse DeltaKinematic um eine inverse Kinematik Berechnung erweitert werden. Diese Funktion wiederum könnte dann von einer Bahnplanung verwendet werden.

Eine weitere Möglichkeit ist, die Klasse DeltaKinematic in einem anderen Projekt zu verwenden, in dem eine Applikation für irgend ein Delta-Roboter mit drei Armen erstellt werden soll.

Quellenverzeichnis

[1] *Repository: EEDURO, https://github.com/eeduro/eeduro-platform*
Stand vom 13.08.2017

[2] *Masterarbeit: Education Robot Platform; Adam Bajric, Silvan Huber, Stefan Landis*