Kurzfassung

Ein mechatronisches System, wie ein Roboter, ist ein komplexes Zusammenspiel von Teilsysteme, die aus den verschiedenen Disziplinen Mechanik, Elektronik und Informatik stammen.

Dieser Umstand macht es schwierig die einzelnen Teilsystemen zu entwickeln und testen. Um die Entwicklung zu unterstützen und beschleunigen werden Simulations- und Visualiserungs-Werkzeuge eingesetzt. In dieser Arbeit sollen die Möglichkeiten vom *ROS*-Ökosystem als unterstützendes Werkzeug für *EEROS*-Applikationen evaluiert werden. Im speziellen sollen Lösungen und Vorlagen für die Entwickler von *EEROS*-Applikationen erarbeitet werden.

Dafür werden zwei Fallbeispiele umgesetzt. Die Umsetzung besteht zum einen aus einer Simulation und zum anderen aus einer Visualisierungs-Lösung. Mit der Visualisierung sollen Daten und Zustände gleichermassen vom realen Roboter oder vom simulierten Roboter dargestellt werden.

Das erste Fallbeispiel ist eine einfachen Motor-Baugruppe. Mit diesem Beispiel konnte erfolgreich aufgezeigt werden, wie die Entwicklung eines Reglers in *EEROS* unterstützt werden kann. Ebenfalls wurde eine Entwicklungsumgebung mit Simulation und Visualisierung für den *EEDURO-Delta* Roboter erstellt. Diese kann dann eingesetzt werden bei der Entwicklung für die *EEROS*-Applikation vom Delta-Roboter. Auch kann mit der Entwicklungsumgebung vom *EEDURO-Delta* Roboter das Framework *EEROS* kennen gelernt werden ohne zuerst Hardware anzuschaffen.

Mit den beiden in dieser Arbeit gezeigten Fallbeispielen erhalten *ROS*- und *EEROS*-Entwickler eine Vorlage für das Erstellen von Simulationen und Visualisierungen von Robotern.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.2 Aufgabenstellung

Das Ziel dieser Arbeit ist zu evaluieren wie *ROS* als unterstützendes Werkzeug für *EEROS*-Applikationen eingesetzt werden kann. Dafür soll zuerst ein Konzept ausgearbeitet werden, in dem unter anderem geklärt wird wie die Schnittstellen zwischen *ROS* und *EEROS* aussehen. Das Konzept soll anschliessend für zwei Fallbeispiele umgesetzt werden. Eine einfache Motor-Baugruppe dient als erstes Fallbeispiel. Anhand dieses Beispiels soll die Korrektheit von dem Konzept gezeigt werden. Ausserdem soll Das zweite Fallbeispiel ist der EEDURO-Deltaroboter. Mit einem Deltaroboter soll gezeigt werden dass das Konzept auch funktionieren mit komplizierteren Robotern.

In dieser Arbeit wird EEROS als externe Software verwendet. Jedoch soll hier angemerkt werden dass jede Software verwendet werden kann, insofern die ROS-Library in diese integriert werden kann. Ein Beispiel wie diese Integration aussehen kann ist, in der Arbeit

1.3 Begriffe

In diesem Abschnitt werden in Kürze die Programme und Begriffe vorgestellt die in dieser Arbeit verwendet werden.

1.3.1 ROS

ROS (Robot Operating System) ist ein Software-Framework für die Programmierung von Roboteranwendungen. Mit *ROS* werden vor allem übergeordnete Aufgaben in einer Roboteranwendung umgesetzt. Es besteht aus einem Set von Softwarebibliotheken und Werkzeugen. Die einzelnen Teile von *ROS* sind organisiert als Packages. Das *ROS*-Netzwerk besteht aus Knoten die über Peer-to-Peer miteinander kommunizieren.

Für das Verständnis dieser Arbeit ist ein Grundwissen über *ROS* essentiell. Im Abschnitt 1.3.1.1 werden die Begriffe aus *ROS* nochmals kurz aufgefrischt. Darum wird *ROS*-Neulinge empfohlen, sich einen Überblick über *ROS* zu verschaffen. Gute Quellen für dies sind:

- Core Componets[1]
- ROS Wiki[2]
- Arbeit "*MME Simulationsprojekt: ROS und Gazebo*" [3]

[1] `http://www.ros.org/core-components/`
[2] `http://wiki.ros.org/`
[3] siehe Anhang xx

1.3.1.1 ROS-Komponenten

In diesem Kapitel werden die wichtigsten *ROS*-Komponenten aufgelistet. Es werden nur die Begriffe der Komponenten kurz vorgestellt. Die Begriffe werden bewusst nicht ins Deutsche übersetzt damit die Begriffe geläufig sind, wenn man die offizielle *ROS*-Dokumentation konsultiert.

Master
Der Master ist die Kernkomponente vom *ROS*-Netzwerk. Er verwaltet die Kommunikation zwischen den Knoten (Nodes). Das Kommunikationsverfahren das verwendet wird ist das Publish & Subscribe[4] Prinzip.

Nodes
Knoten sind Prozesse in denen Berechnungen oder Treiber ausgeführt werden. Somit kann ein Knoten eine Datenverarbeitung, ein Aktor oder ein Sensor darstellen.

Messages
Die Messages sind eine Konvention für den Austausch von Daten. Darum gibt es für die unterschiedlichen Anforderungen der Kommunikation, verschiedene Nachricht-Typen.

Topics
Der Nachrichtenaustausch mit dem Publish & Subscribe Mechanismus ist anonym. Darum werden die Nachrichten unter einem Thema (Topic) ausgetauscht.

1.3.2 EEROS

EEROS[5] ist eine Roboter-Framework mit Fokus auf die Ausführung von Echtzeit-Aufgaben. Somit ist es geeignet für die Umsetzung von untergeordneten Aufgaben in einer Roboteranwendung, wie das Regeln von einem Motor.

1.4 Zusammenarbeit

Bei der Entwicklung der Kommunikation und Synchronisation mit EEROS hat Marcel Gehrig aktiv mit geholfen. In der Arbeit "Eine ROS Anbindung für EEROS" von Marcel Gehrig wird beschrieben wie *EEROS* in das *ROS*-Netzwerk eingebunden wird.

[4] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
[5] http://eeros.org

2 Konzept

Eine Skizze des Konzeptes ist in der Abbildung 2.1 zusehen. Die zwei Hauptkomponenten sind *EEROS* und *ROS*. *EEROS* übernimmt in diesem Szenario die Aufgabe des Reglers. Und *ROS* die Aufgaben der Simulation und der Visualisierung.

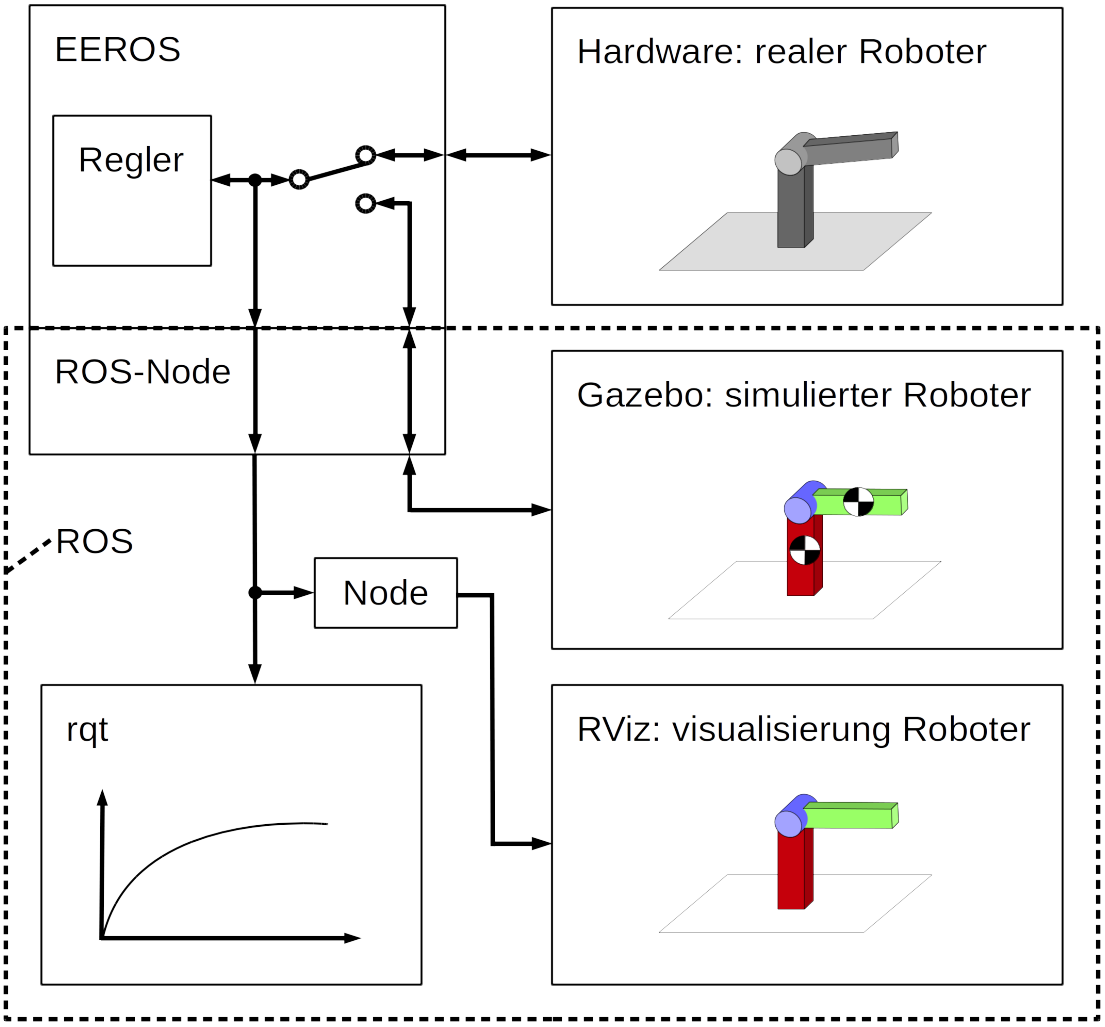

Abbildung 2.1: Konzept

Damit *EEROS* mit *ROS*-Komponenten kommunizieren kann, wurde einen *ROS*-Node ins *EEROS* integriert. Diese Integration wird in der Arbeit xx erklärt. Somit hat *EEROS* die Fähigkeit über das *ROS*-Kommunikationsprinzip *Publish & Subscribe* Daten auszutauschen. Das Konzept sieht vor, das *EEROS* wahlweise einen echten oder simulierten Roboter steuert und regelt. Der Roboter in der Skizze ist aus zwei Gliedern und einem Gelenk aufgebaut. Die Stellgrösse die vom *EEROS* vorgegeben wird ist das Drehmoment für das Gelenk. Und die Regelgrösse ist die Winkelposition des Gelenks.

Die Visualisierung mit *RQt* und *RViz* erfolgt gleichermassen ob ein echter oder simulierter Roboter betrieben wird. Mit *RQt* können die Grössen in Plots dargestellt werden. Und der Zustand des Roboters kann mit *RViz* visualisiert werden. Da *RViz* nicht direkt die Gelenkwinkel interpretieren kann müssen diese von einem *ROS*-Knoten in Koordinatensystem-Transformationen [1] umgerechnet werden.

[1] siehe Kapitel 2.5

2.1 Gazebo

Gazebo ist eine Starrkörper-Simulationsumgebung für Mehrkörper-Systeme. Vom System das man simulieren möchte muss eine System-Beschreibung in Form von einer *SDF*-Datei erstellt werden. *Gazebo* baut dann die Simulation anhand dieser Datei auf. In unserem Fall wird im *SDF*[2] das Roboter-Modell beschrieben.

Um *Gazebo* mit Funktionen zu erweitern können Plugins verwendet werden. Dabei kann man schon fertige Plugins verwenden oder selber eines programmieren.

2.2 RViz

Mit dem Visualisierungs-Tool *RViz* kann der Zustand vom Roboter visualisiert werden. Der Zustand ist die momentane Stellung von den Gelenken des Roboters, oder einfacher ausgedrückt die Pose vom Roboter. Auch wird *RViz* eingesetzt für die Visualisierung von anderen Daten wie z.B. die Sensor-Daten einer 3D-Kamera

Damit RViz den Zustand vom Roboter darstellen kann braucht es folgende Informationen:

- Aussehen jedes Glieds des Roboters, wie Form und Farbe
- laufend für jedes Glied: Position und Orientierung

Mit einer *URDF*-Datei[3] müssen dem *RViz* die Information über das Aussehen zur Verfügung gestellt werden. *URDF* ist ein Format für die Modell-Beschreibung eines Roboters. In *RViz* muss immer ein Koordinatensystem als fixe Referenz angegeben werden. Normalerweise wird das Welt-Koordinatensystem gewählt. Um ein Körper im Raum darzustellen braucht *RViz* eine Angabe zur Position und Orientierung relative zum Referenz-Koordinatensystem. Diese Angabe wird als Transformation[4] bezeichnet. Diese Transformations-Daten erhält *RViz* laufend und kann immer den aktuellen Zustand des Roboters darstellen.

2.3 RQt

RQt ist Framework für die GUI Entwicklung in *ROS*. Diese GUI's werden als Plugins implementiert. Somit können mehre GUI's in einem *RQt*-Fenster verwendet werden. Für fast jedes gängige *ROS*-Command-line-Tool gibt es schon Plugins[5]. Es können aber auch selbst programmierte Plugins verwendet werden. Für die Visualisierung von zeitabhängigen Grössen wird das *RQt*-Plugin *multiplot*[6] eingesetzt.

2.4 Roboter-Modell

Die Programme *Gazebo* und *rviz* verwenden zwei unterschiedliche Datei-Formate für die Modell-Beschreibung. Beide sind im XML-Stil gehalten und repräsentieren kinematische Strukturen. Diese Strukturen sind aus Gliedern (Links) und Gelenken (Joints) aufgebaut.

[2] siehe Kapitel 2.4.2
[3] siehe Kapitel 2.4.1
[4] siehe Kapitel 2.5
[5] `http://wiki.ros.org/rqt/Plugins`
[6] `http://wiki.ros.org/rqt_multiplot`

2.4.1 URDF-Unified Robot Description Format

Das *URDF*-Format[7] wird standardmässig in *ROS* verwendet für die Beschreibung von Robotern. Es kann nur kinematische Strukturen abbilden die ein Baum-Form haben. Somit können keine geschlossenen kinematischen Ketten beschrieben werden. Im Kapitel 2.4.1 wird beschrieben wie ein *URDF*-Model aufgebaut werden kann.

2.4.2 SDF-Simulation Description Format

Das *SDF*-Format wird bis jetzt ausschliesslich im *Gazebo* verwendet. Es kann kinematische Graph-Strukturen abbilden. Deshalb können auch geschlossene kinematische Ketten beschrieben werden.

2.4.3 Konvertierung URDF zu SDF

Damit für ein Roboter nicht zwei Dateien erstellt und instand gehalten werden müssen gibt es eine Lösung. Die Lösung besteht darin die *URDF*-Datei in eine *SDF*-Datei zu konvertieren. Das *URDF*-Format ist jedoch nicht so mächtig wie das *SDF*-Format. Um jedoch den ganzen Umfang der Möglichkeiten vom *SDF*-Format zu nutzen kann die *URDF*-Datei mit speziellen XML-Elementen erweitert werden. In welchen Fällen es solche Spezial-Elemente braucht und wie sie eingesetzt werden wird im Kapitel xx gezeigt.

2.4.4 URDF-Modell

Ein Modell besteht aus den zwei Teilen: Glied (Link) und Gelenk (Joint). Für jedes gib es ein entsprechendes XML-Element. Die Beziehung zwischen den beiden Komponenten ist in der Abbildung 2.2 zusehen. Speziell zu beachten ist das die Abstände zwischen den Gelenken nicht vom Link-Element sondern vom Joint-Element definiert wird. Somit wird die Kinematische Struktur des Roboters nur über die Joint-Elemente beschrieben. Auch ist das Koordinatensystem vom Gelenk gleichzeitig der Ursprung des Koordinatensystems vom Kind-Glied des jeweiligen Gelenks.

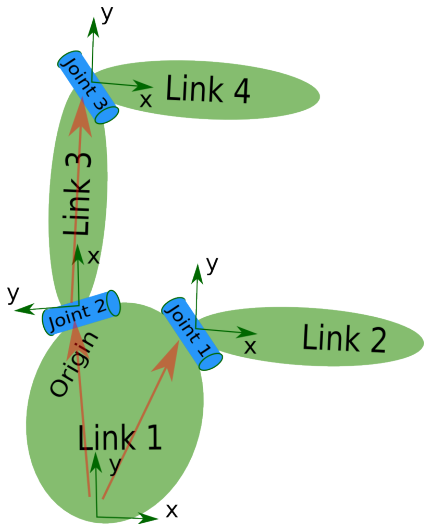

Abbildung 2.2: Aufbau URDF [1]

[7] http://wiki.ros.org/urdf

Link

Das Link-Element beschreibt ein einzelnes Glied vom Roboter. Folgende Informationen sind als Sub-Elemente enthalten:

- Massenträgheit
- Geometrie und Material für Aussehen
- Geometrie für Kollisionsberechnung

Für jedes dieser aufgelisteten Sub-Element muss der Ursprung bezüglich des Link-Koordinatensystem angegeben werden. Wie die verschiedenen Sub-Elemente in Verbindung stehen, ist in der Abbildung 2.3 dargestellt.

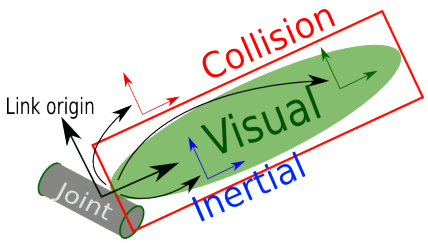

Abbildung 2.3: Schematische Darstellung Link-Element [1]

Joint

Mit dem Joint-Element können Gelenke unterschiedlichster Art beschrieben werden. Folgende Informationen braucht es zur Beschreibung eines Gelenks:

- Gelenk-Type
- Gelenk-Ursprung
- Eltern Link-Element
- Kind Link-Element

Die zwei häufigsten Gelenk-Typ die verwendet werden sind:

- "kontinuierliche" Typ, freie Rotation um eine Achse (continuous)
- "fixe" Type, beide Links fest verbunden (fixed)

Die Beziehung dieser Angaben zueinander ist in Abbildung 2.4 dargestellt.

2.5 Transformationen

Eine Transformation beschreibt wie ein Koordinatensystem relative zu einem anderen steht. Koordinatensysteme beschreiben eine Position und Orientierung im Raum für zum Beispiel ein Glied. Aufbau einer Transformation:

- Eltern-Koordinatensystem
- Kind-Koordinatensystem
- Translation

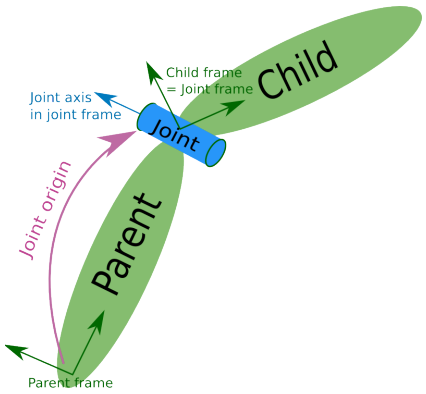

Abbildung 2.4: Schematische Darstellung Joint-Element [1]

- Rotation

Durch die Eltern/Kind Relation bilden die Transformationen eine Baumstruktur. Das Wurzel-Element dieses Baumes ist meistens das Welt-Koordinatensystem.

Zur Veranschaulichung ein Beispiel:

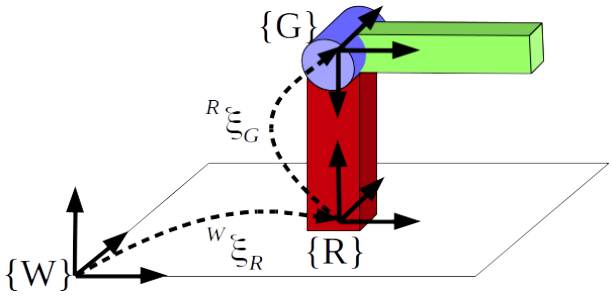

Abbildung 2.5: Transformation

Man berechnet das Koordinatensystem des grünen Gliedes in Bezug zu der Welt wie folgt:

$$^{W}\xi_{G} = {}^{W}\xi_{R} * {}^{R}\xi_{G} \tag{2.1}$$

Die Angabe für das Koordinatensystem von Objekt {B}, in Bezug zu einem Koordinatensystem {A}, ist gleich zeitig auch die Transformation ($^{A}\xi_{B}$) von Koordinatensystem {A} zu {B}.

2.5.1 Umrechnung Gelenkwinkel zu Transformationen

Für die Umrechnung von den Winkelpositionen der Gelenke zu den Koordinaten-Transformationen gibt es in *ROS* den *robot_state_publisher*. Dieser *ROS*-Node benötigt die Beschreibung des Roboter-Modells als *URDF*-Datei. Er setzt dann die Gelenkwinkel im Modell ein und rechnet anschliessend die einzelnen Transformationen zwischen den Gelenken aus. Da der Ursprung jedes Gliedes im Ursprung vom Gelenk liegt, sind die berechneten Transformationen gleichzeitig die zwischen den Gliedern. Mit diesen Transformationen kann dann *RViz* die Glieder im Raum so Positionieren, dass es den Roboter korrekt darstellen kann.

3 Motor

In diesem Kapitel wird anhand eines einfachen Fallbeispiels gezeigt, wie mit *ROS* die Entwicklung einer *EEROS* Applikation unterstützt werden kann. Das Fallbeispiel ist eine Motor-Baugruppe (Abbildung 3.1).

Für dieses System muss im *EEROS* ein Regler entwickelt werden. Um die Entwicklung zu unterstützen und den Regler zu testen, wird eine Simulation erstellt. Und mit Plots von den Stell- und Regel-Grössen kann die Leistung des Reglers besser beurteilt werden.

Durch das einfache Beispiel können alle Grundlagen vorgestellt werden, die es braucht um eine Simulation und Visualisierungen erstellen zu können.

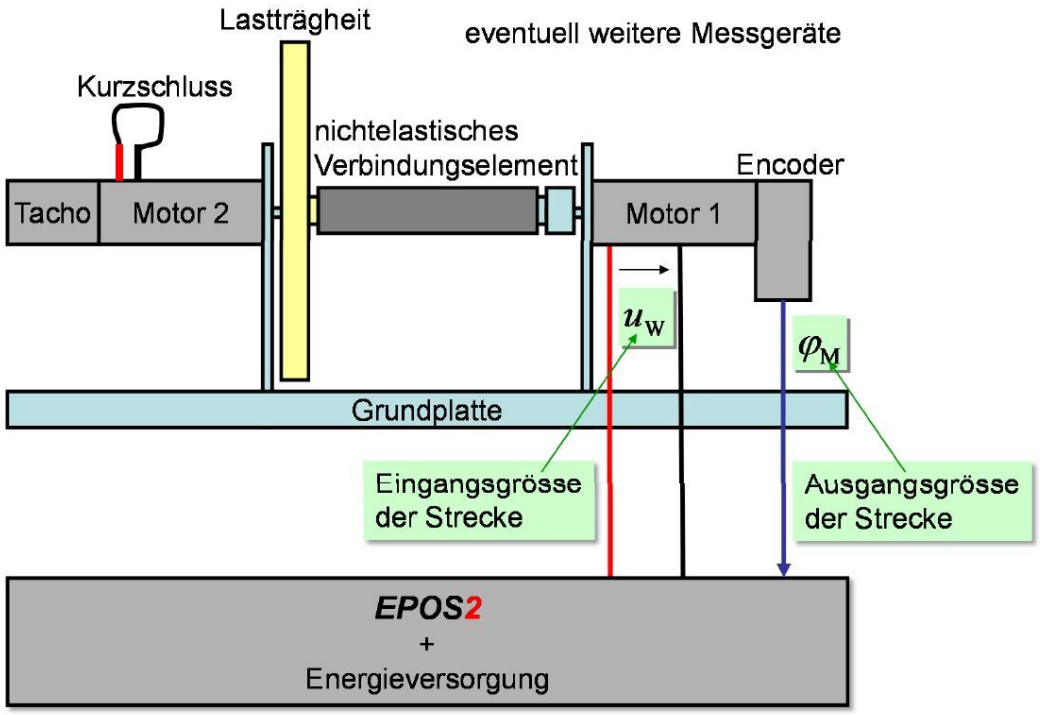

Abbildung 3.1: schematische Darstellung von Motor-Baugruppe

3.1 Motor-Baugruppe

Die Baugruppe besteht aus einem Motor, Schwungrad und linearen Dämpfer. Diese drei Komponenten sind mit einander verbunden. Der lineare Dämpfer ist realisiert durch einen zweiten kurzgeschlossenen Motor.

Die Parameter für das Simulations-Model konnten grössten teils aus den Datenblättern der jeweiligen Komponenten entnommen werden. Die Dämpfungskonstante und die Rotationsträgheit der Schwungscheibe mussten berechnet werden. Die Datenblätter und Berechnungen sind im Repository "*motor_sim*" [1] zu finden.

Für Parameter die nicht im Datenblatt stehen oder nicht genügend Informationen für eine Berechnung vorhanden ist, wurden geschätzt. Jedoch haben die geschätzten Parameter in diesem Fallbeispiel keinen nennenswerten Einfluss auf die Simulation.

[1] `https://github.com/manuelilg/motor_sim/tree/master/motor_description/datasheets`

3.2 Modell

Für die Motorbaugruppe wurde eine Modell-Beschreibung im *URDF*-Format erstellt. Die Kinematische Struktur des Modells ist in der Abbildung xx dargestellt. Wie zu sehen ist, das der Dämpfer noch nicht mit dem Schwungrad verbunden ist, dieser Umstand wird im Kapitel **??** erläutert.

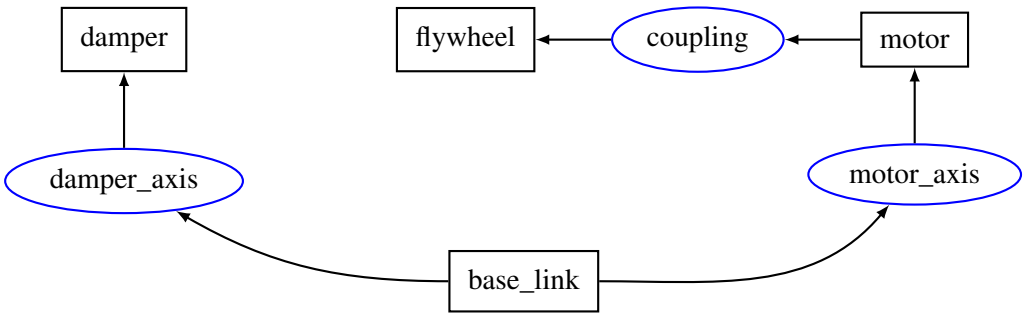

Abbildung 3.2: kinematische Struktur Motor

Die komplette Datei ist im Repositiory "*motor_sim*" [2] abgelegt.

3.2.1 Kinematische Kette schliessen

Das *URDF*-Format kann keine geschlossenen kinematischen Strukturen abbilden. Dieser Mangel kann aber behoben werden mit: einem *SDF*-Joint Eintrag im *URDF* und mit dem *URDF*-Spezial-Element "Mimic".

3.2.1.1 SDF-Joint

Die in diesem Abschnitt erklärte Anpassung wird benötigt, damit in der Simulation das Schwungrad und der Dämpfer miteinander gekoppelt sind.

Im *URDF* können Informationen hinterlegt werden, die nur *Gazebo* interpretiert. Dafür müssen die Informationen mit dem XML-Element "gazebo" umschlossen werden.

Wenn man jetzt ein Joint-Element im *SDF*-Format einsetzt kann man die kinematische Struktur schliessen. Denn die URDF-Datei wird, bevor es ins *Gazebo* geladen wird, ins *SDF*-Format konvertiert. Während der Konvertierung werden die Zeilen die sich im XML-Element "gazebo" befinden einfach ins *SDF* übernommen.

Wie der Eintrag für die Motor-Baugruppe lauten muss, ist in Auflistung xx gezeigt.

3.2.1.2 Mimic

Der *ROS*-Knoten *robot_state_publisher* berechnet die Transformations-Daten für die Darstellung der Körper im *RViz*. Dafür braucht er alle Gelenkwinkel. Das Modell vom *URDF* hat zwei Gelenke aber das in der Simulation nur eines. Da in der Simulation die beiden Gelenke verbunden wurden.

Deshalb muss im Gelenk folgender Eintrag eingefügt werden:

[2] `https://github.com/manuelilg/motor_sim/tree/master/motor_description/urdf`

Dieser Eintrag wird vom *ROS*-Knoten *joint_state_publisher* interpretiert. Er bewirkt das das Gelenk xx der gleiche Winkel hat wie das Gelenk xx. Somit erhalten wir vom *joint_state_publisher* für die Eingabe von einem Gelenkwinkel xx, die Ausgabe für die Gelenkwinkel xx und xx.

Diese beiden werden dann vom *robot_state_publisher* als Eingabe benötigt.

3.2.2 Gazebo Plugins

Damit das Modell in der Simulation mit dem *ROS*-Netzwerk interagieren kann, müssen Plugins eingesetzt werden.

Mit Folgenden Zeilen werden zwei *Gazebo*-Plugins im Motor-Modell eingebettet:

Das Plugin xx publisht den Winkel von Gelenk xx. Achtung das Gazebo-Plugin xx ist nicht mit dem ROS-Knoten xx zu verwechseln. Es ist aus dem Package xx, ein Standard *ROS*-Package.

Das zweite Plugin xx wird im nächsten Abschnitt genauer erläutert.

3.2.2.1 Joint Force Plugin

Das Plugin xx appliziert ein Drehmoment auf das Gelenk xx. Die Grösse des Drehmoments erhält das Plugin über das Topic xx. Dieses Plugin würde im Rahmen dieser Arbeit erstellt und ist im Repository xx zu finden.

joint force vorstellen

auf tutorial verweisen

es wird vor jedem Simulations-Schritt aufgerufen

sync erklären

Fälle von Einsatz: 2 oder 3. auch noch

wie benutzen

3.3 •

3.4 rviz

system wird geladen mit robot description braucht auch tf

3.4.1 Joint State Publisher

erklären für was gebraucht wird für mimic tag Achtung ist eine eigenständiger Knoten

3.5 rqt

keine speziellen anpassungen an urdf

3.6 Starten

einzelnen programme gazebo, rviz, rqt

4 EEDURO-Delta-Roboter

In diesem Kapitel wird das zweite Fallbeispiel mit dem EEDURO-Delta-Roboter vorgestellt. Der EEDURO-Delta ist ein kleiner Delta-Roboter, der für Schulungszwecke gedacht ist. Die Modell-Beschreibung des Roboters ist im *URDF*-Format gehalten. Speziell ist aber das zusätzlich noch die Skriptsprache *XACRO* eingesetzt worden.

4.1 Model

Das Modell des Delta-Roboter kann grob in drei Teile aufgeteilt werden:

- Rahmen (base)
- drei Arme (arm 1-3)
- Werkzeug (tool)

Der Arm selber besteht aus einem Oberarm auch Link 1 genannt und aus einem Unterarm. Der Unterarm selber besteht selber wider aus:

- Doppelgabel (Link 2)
- zwei Stangen (Link 3.1 und Link 3.2)
- 2.Doppelgabel (Link 4)

Wie beim Motor muss die kinematisch Kette geschlossen werden mit dem *SDF*-Joint Element. Alle grünen Ovale sind *SDF*-Joint Elemente.

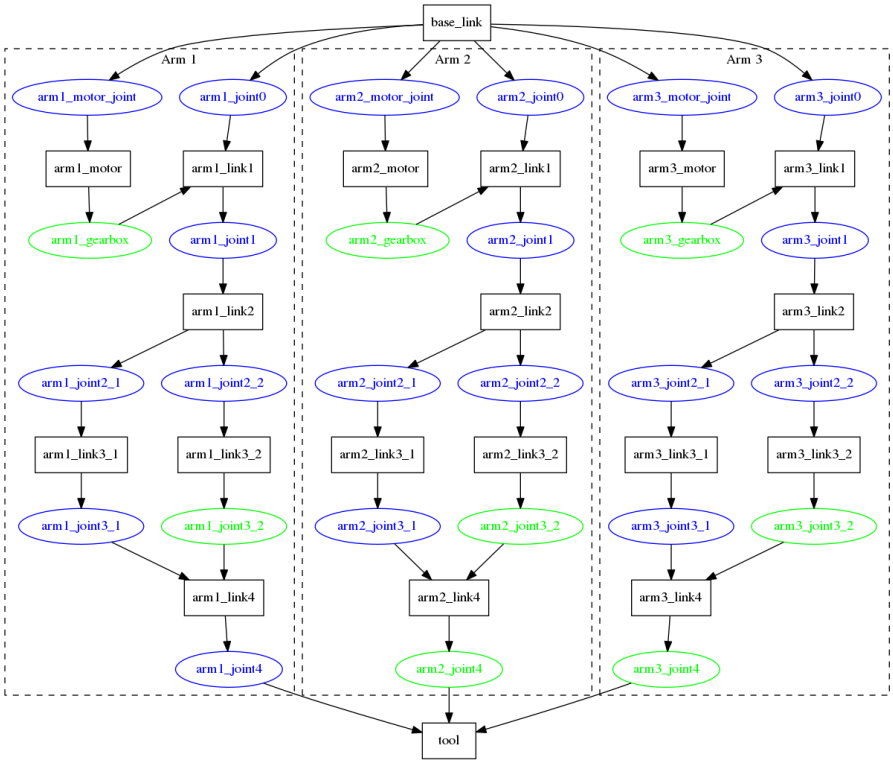

Abbildung 4.1: kinematische Struktur EEDURO-Delta

4.1.1 Berechnung Masse und Rotationsträgheit

Für jedes Glied musste die Masse und Rotationsträgheit berechnet werden, damit das Modell des Roboters sich in der Simulation wie der reale Roboter verhält. Die Glieder bestehen jedoch aus mehreren Bauteilen. Dieser Umstand verkompliziert die Berechnung sehr. Auch erschwerend kam hinzu das die CAD-Daten nur im *STEP*-Austauschformat vorliegend waren.

Das führt zu folgendem Arbeitsablauf:

- für jedes Glied/Baugruppe eine CAD-Datei erstellen
- für jede CAD-Datei:
  - für jedes Bauteil folgende Daten exportieren
    - Volumen von Körper
    - Schwerpunkt von Körper
    - Rotationsträgheit-Tensor von Volumen
  - exportierte Daten in Matlab eintragen
  - für jedes Bauteil Dichte in Matlab hinterlegen
  - für jedes Bauteil mit Dichte: Masse und Rotationsträgheit ausrechnen
- Schwerpunkt und Masse von Baugruppe bestimmen
- Rotationsträgheit von Baugruppe ausrechnen (Steinerscher Satz für jedes Bauteil anwenden)

Die CAD-Daten und Matlab-Dateien sind im Repository xx zufinden.

4.2 EEDURO-Delta Joint State Publisher

Wie schon im mehrmals erwähnt braucht

Somit gibt es folgenden Prozess Graph.

4.2.1 Vereinfachung

Eine Vereinfachung für das erstellen und unterhalten von den Dateien für die System-Beschreibung wurde schon im Kapitel xx Konvertierung vorgestellt. Mit dem Programm *XACRO* kommt eine weitere hinzu.

4.2.1.1 XACRO

XACRO ist eine Makro Sprache für XML-Dateien. Mit *XACRO* können kürzere und einfacher zu unterhaltende XML-Dateien erstellt werden. Denn mit Hilfe der Makros können Wiederholungen vermieden werden. Auch kann Dank *XACRO* eine XML-Datei in mehrere Unterdateien aufgeteilt werden. Somit kann ein komplexes System logisch in Untersysteme aufgeteilt werden, die dann jeweils in einer eigenen XML-Datei beschrieben werden. Im Kapitel XX werden die Vorteile von *XACRO* in der *URDF* -Datei des Delta-Roboters gezeigt.

4.3 Installation und Ausführung

Verweis auf Repository readme

5 Ergebnisse, Fazit und Ausblick

5.1 Ergebnisse

- zeigen von vollständigem Zusammenspiel eeros <-> ros
- zeigen erstellen Simulation komplexer Systeme
- Vorlagen und einzeln Komponenten die wiederverwendet werden können

5.2 Fazit

- möglichkeit EEROS auszuprobieren
- ausprobieren ohne realtime kernel
- ausprobieren mit eeduro-Delta

5.3 Ausblick

- wiederverwenden von einzelnen Komponenten wie plugins
- EEROS-Applikation für Delta-roboter auf neuste EEROS-Version updaten
- erweitern von Delta-Model mit Rotation Werkzeug
-

Quellenverzeichnis

[1] *Web: ROS, https://ros.org/*
Stand vom 20.08.2017

A Anhang

- Datenblätter Motor-Baugruppe
- Datenblätter Bauteile von EEDUO-Delta
- Source code