# Kurzfassung

Ein mechatronisches System, wie ein Roboter, ist ein komplexes Zusammenspiel von Teilsystemen, 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 Entwickler-Werkzeug 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 eine 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 dem Upgrade der bestehenden *EEROS*-Applikation für den Delta-Roboter auf die neuste *EEROS*-Version. Auch kann mit der Entwicklungsumgebung vom *EEDURO-Delta* Roboter das Framework *EEROS* kennen gelernt werden ohne zuerst Hardware anzuschaffen.

Mit den beider 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* eingesetzt werden kann, um die Entwicklung von *EEROS*-Applikationen zu unterstützen. Dafür wird zuerst ein Konzept ausgearbeitet. Das Konzept wird an einem einfachen Fallbeispiel, bestehen aus einer Motorbaugruppe, gezeigt. FUUUUUCK

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 Beispiele wie diese Integration aussehen kann ist in der Arbeit

## 1.3 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 xx 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
- ROS Wiki
- Arbeit Andreas Kalberer

### 1.3.1 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 Prinzip.

##### 1.3.1.1 Nodes

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

#### 1.3.1.2 Messages

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

#### 1.3.1.3 Topics

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

## 1.4 EEROS

*EEROS* 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.

# 2 Konzept

### Stichworte

Eine Skizze des Konzeptes ist in der Abbildung 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.

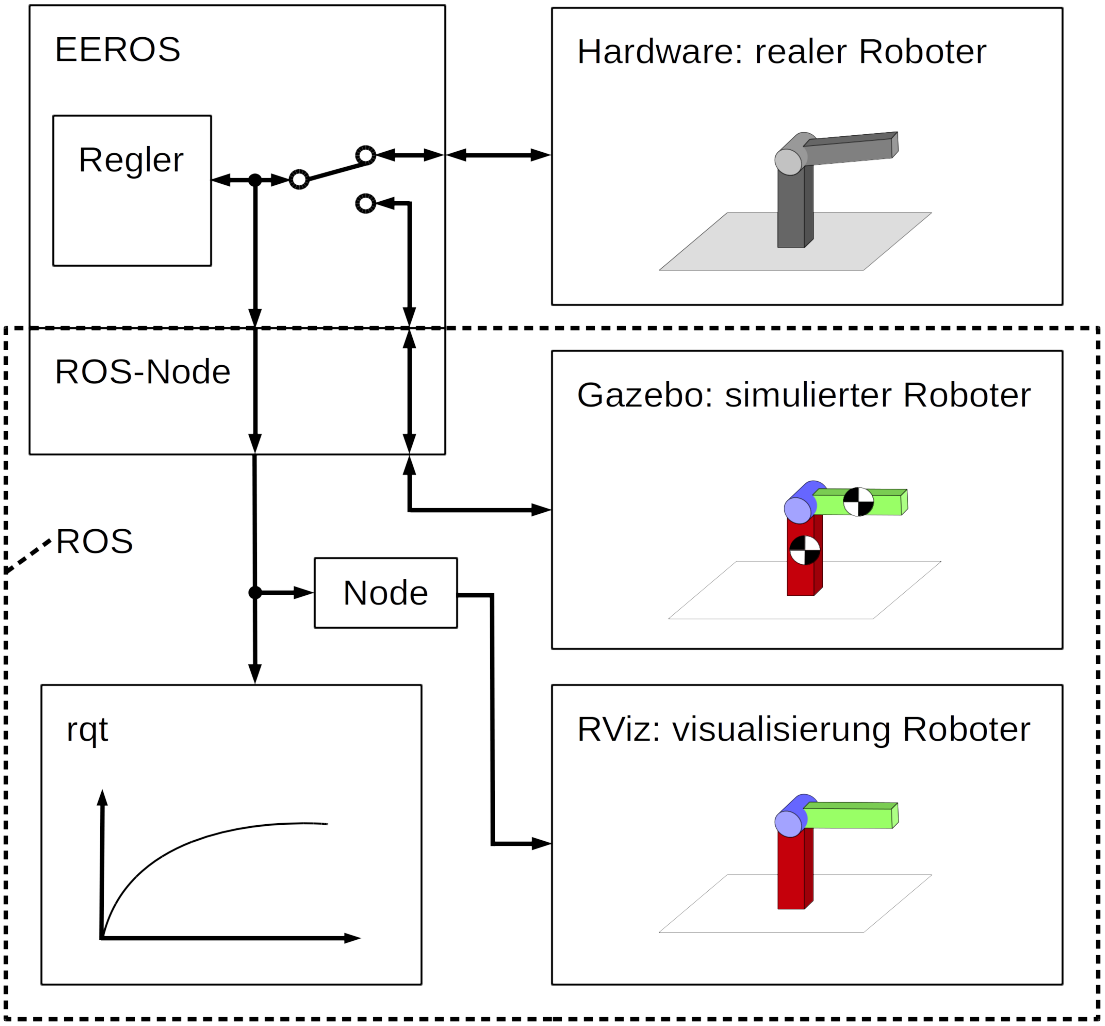


Abbildung 2.1: Konzept

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 umgerechnet werden.

## 2.1 Gazebo

*Gazebo* ist eine Starrkörper-Simulationsumgebung für Mehrkörper-Systeme. Vom System das man simuliert 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* 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.sie Auch wird *RViz* eingesetzt für die Visualisierung von anderen Daten wie z.B. die Sensor-Daten von 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 müssen dem *RViz* die Information über das Aussehen zur Verfügung gestellt werden. *URDF* ist ein Format für die Modell-Beschreibung eines Roboters. In *RViz* werden muss immer ein Koordinatensystem als fixe Referenz angegeben werden, normaler weise 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 als Transformation bezeichnet. Diese Transformations-Daten erhält *RViz* laufend und kann dann immer der aktuelle 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. Es können aber auch selbst programmiert Plugins verwendet werden. Für die Visualisierung von zeitabhängigen Grössen wird das *RQt*-Plugin *multiplot* 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.

### 2.4.1 URDF-Unified Robot Description Format

Das *URDF*-Format wird standardmässig in *ROS* verwendet, für die Beschreibung von Robotern. Es kann nur kinematische Strukturen abbilden die einen Baum-Form haben. Somit können keine geschlossenen kinematischen Ketten beschrieben werden. Im Kapitel xx 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 könne auch geschlossenen kinematische Ketten beschrieben werden.

### 2.4.3 Konvertierung

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 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 xx 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.

#### Link

Das Link-Element beschreibt ein einzelnes Glied vom Roboter. Folgende Informationen sind als Sub-Elementen 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 xx dargestellt.

### 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 Beziehung dieser Angaben zueinander ist in Abbildung xx dargestellt.

## 2.5 Transformationen

Eine Transformation beschreibt wie eMit dem in 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
- 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:

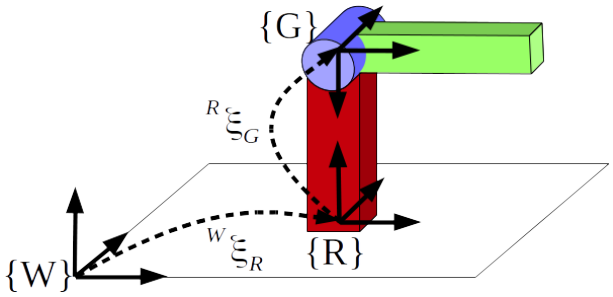


Abbildung 2.2: Konzept

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* der *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 dann 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 um die Leistung des Reglers besser beurteilen zu können werden Stell- und Regel-Grössen in Plots dargestellt.

Durch das einfache Beispiel können alle Grundlagen vorgestellt werden, die es braucht für das erstellen von einer Simulation und Visualisierungen.

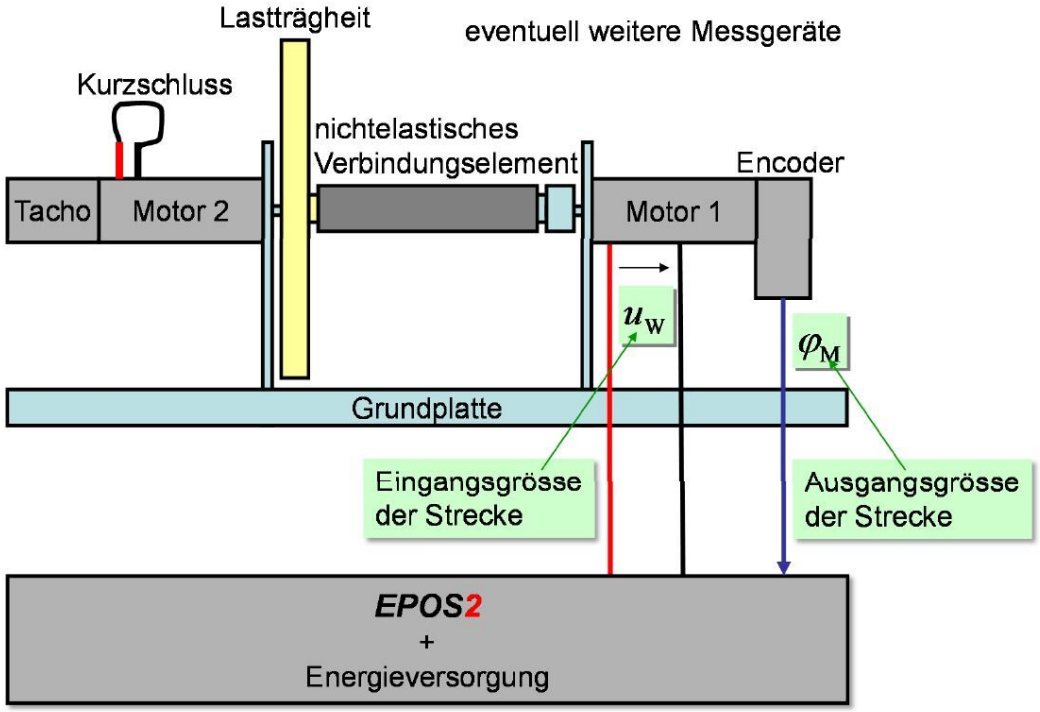


Abbildung 3.1: schematische Darstellung von Motor-Baugruppe

## 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 Simulations-Model konnten grössten teils aus den Datenblättern der jeweiligen Komponenten entnommen werden. Berechnung dämpfung 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.

Auflistung von Parametern

## 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. Zu sehen ist das der Dämpfer noch nicht mit dem Schwungrad verbunden ist, dieser Umstand wird im Kapitel xx erläutert.

Die komplette Datei ist im Repositiory:motor_sim abgelegt.

### 3.2.1 Kinematische Kette schliessen

Da das *URDF*="Format kann keine geschlossenen kinematischen Strukturen abbilden. Dieser Mangel kann aber behoben werden mit: einem *SDF* 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 einem 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. Und 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 *robot_state_publisher*

### 3.2.2 Gazebo Plugins

beschreiben für was die Plugins sind, wie sie eingesetzt werden joint publisher

#### 3.2.2.1 Joint Force Plugin

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 rviz

system wird geladen mit robot description braucht auch tf

### 3.3.1 Joint State Publisher

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

## 3.4 rqt

keine speziellen anpassungen an urdf

## 3.5 Starten

# 4 Delta

### 4.0.1 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.

#### 4.0.1.1 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.

# 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