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 System oder vom simulierten System dargestellt werden.

Das erste Fallbeispiel ist eine einfachen Motor-Baugruppe. Mit diesem Beispiel konnte erfolgreich aufgezeigt werden, wie die Entwicklung einer *EEROS*-Applikation 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.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Anforderungen/Aufgabenstellung . . . 1
1.2 ROS . . . 1
1.2.1 ROS-Komponenten . . . 1
1.2.1.1 Nodes . . . 1
1.2.1.2 Messages . . . 2
1.2.1.3 Topics . . . 2
1.3 EEROS . . . 2

2 Konzept **3**
2.1 Gazebo . . . 3
2.2 rviz . . . 4
2.3 rqt . . . 4
2.4 System-Beschreibung . . . 4
2.4.1 URDF-Unified Robot Description Format . . . 4
2.4.2 SDF-Simulation Description Format . . . 4
2.4.2.1 Konvertierung . . . 5
2.4.3 Modell . . . 5
2.4.4 Vereinfachung . . . 5
2.4.4.1 XACRO . . . 6
2.5 TF . . . 6

3 Motor **7**
3.1 Motor-Baugruppe . . . 7
3.2 URDF . . . 8
3.2.1 Kinematische Kette schliessen . . . 8
3.2.2 Gazebo Plugins . . . 8
3.2.2.1 Joint Force Plugin . . . 8
3.3 rviz . . . 8
3.3.1 Joint State Publisher . . . 8
3.4 rqt . . . 8

4 Delta **9**

5 Ergebnisse, Fazit und Ausblick **10**
5.1 Ergebnisse . . . 10
5.2 Fazit . . . 10
5.3 Ausblick . . . 10

Quellenverzeichnis **11**

A Anhang **1**

1 Einleitung

1.1 Anforderungen/Aufgabenstellung

- ros konzepte kurz vorstellen wie:
  - msg
  - topic
  - node
- ros-ökosystem: gazebo, rviz, rqt

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

1.2 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 für *ROS*-Neulinge empfohlen sich einen Überblick über *ROS* zu verschaffen. Gute Quellen für dies sind:

- Core Componets
- ROS Wiki
- Arbeit Andreas Kalberer

1.2.1 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.

1.2.1.1 Nodes

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

1.2.1.2 Messages

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

1.2.1.3 Topics

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

1.3 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

- Übersicht von Konzept mit Graphik
- 3 Bereiche aufzeigen: EEROS, Gazebo, rqt & rviz
- Schnittstellen zu einander
- Verweis zu Mäsis Arbeit
- urdf, sdf vorstellen und erklären, hier??? oder in Motor

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.

Damit *EEROS* mit *ROS*-Komponenten kommunizieren kann, wurde einen *ROS*-Node ins *EEROS* integriert. Somit hat *EEROS* die Fähigkeit über das *ROS*-Kommunikationsprinzip *Publish & Subscribe* Daten auszutauschen. Das Konzept sieht vor das *EEROS* wahlweise ein echtes oder simuliertes System regelt. Mit dem Begriff System werden in dieser Arbeit allgemein mechatronische Systeme wie zum Beispiel ein Roboter bezeichnet. Zu und vom System fliessen Daten wie Stell- und Regel-Grössen. Diese Daten werden einfachheitshalber Grössen genannt. Aus den Grössen die vom System kommen kann der Zustand vom System interpretiert werden. Der Zustand von einem System kann zum Beispiel die Winkel-Position eines Motors beschreiben.

Die Zustände und Grössen eines Systems werden visualisiert egal ob an einem realen oder ein simulierten System gearbeitet wird.

Für die Simulation von Systemen wird *Gazebo* eingesetzt. Die Darstellung von den System Zuständen wird mit *rviz* realisiert. Für die Visualisierung der Grössen werden Plots eingesetzt. Für das erzeugen der Plots wird das *rqt* Plugin multiplot eingesetzt.

2.1 Gazebo

- festkörper simulation
- kann mit selbst geschriebenen Plugins erweitert werden.
- benötig sdf für beschreibung des zu simulierenden systems

Gazebo ist eine Simulationsumgebung für Starrkörper. Das zu simulierenden Systems wird mit einem *sdf*-File beschrieben. Das File ist im XML-Format aufgebaut.

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

2.2 rviz

- darstellen von System und deren Zuständen
- benötigt urdf-file für Darstellung

Das Visualisierungs-Tool *rviz* ist für die Darstellung von Systemen geeignet. Dabei können z.B. Zustände von einem Roboter visualisiert werden oder Sensor-Daten wie von einer 3D-Kamera.

Ein System besteht aus mehren Körpern. Um sie darzustellen braucht es unter anderem Informationen über das Aussehen und Form dieser. Diese Informationen müssen dem *rviz* über eine *urdf*-File zur Verfügung gestellt werden. Für die Darstellung der Körper im Raum benötigt *rviz* für jeden Körper noch die Position und Orientierung von diesem im Raum. Deshalb müssen dem *rviz* stetig Koordinaten-Daten für jeden Körper übermittelt werden.

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 Darstellung von den zeit veränderlichen Grössen des Systems wird das Plugin *multiplot* eingesetzt.

2.4 System-Beschreibung

Die Programme *Gazebo* und *rviz* verwenden für die Beschreibung der Systeme unterschiedliche Datei-Formate. In diesem Abschnitt werden diese beiden XML-Formate kurz vorgestellt. Sie können beide kinematische Strukturen repräsentieren. Diese Strukturen werden als Modell bezeichnet.

2.4.1 URDF-Unified Robot Description Format

Das *URDF*-Format wird standardmässig in *ROS* verwendet, für die Beschreibung von Systemen. 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.

2.4.2.1 Konvertierung

Damit für ein System 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 behandelt.

2.4.3 Modell

Ein Modell besteht aus den zwei Elementen 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.

Link

Das Link-Element beschreibt einen einzelne starr Körper in einem System. Die Beschreibung besteht aus:

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

In der Auflistung xx ist zusehen wie ein Link-Element aufgebaut ist.

Joint

Mit dem Joint-Element können Gelenke unterschiedlichster Art beschrieben werden. Die Beschreibung besteht aus:

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

Wie ein Joint-Element aufgebaut sieht man in der Auflistung xx.

2.4.4 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.

2.4.4.1 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.

2.5 TF

joint state publisher erklären für was robot state publisher

verweis rviz braucht tf daten

abklären ob absolute frames oder relative/tf

3 Motor

In diesem Kapitel wird anhand eines einfachen Fallbeispiels gezeigt, wie mit *ROS* die Entwicklung einer *EEROS* Applikation unterstützt werden kann. Das System ins diesem 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, soll eine Simulation verwendet werden. Und um die Leistung des Reglers besser beurteilen zu können sollen Plots von den zeit-variablen Grössen erstellt werden.

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

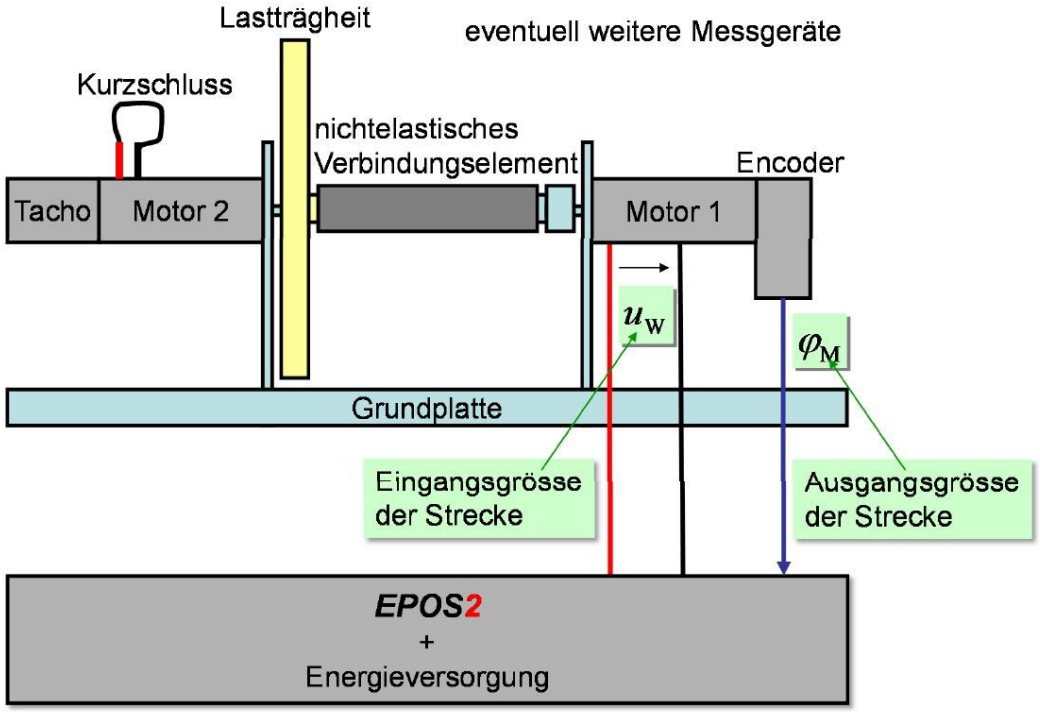

Abbildung 3.1: Motor-Baugruppe

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 dieser Simulation keinen nennenswerten Einfluss.

Auflistung von Parametern

3.2 URDF

nur Beispiel hier vom Link Dämpfer und vom joint Dämpfer Achse

erwähnen es könnten Vereinfachungen vom Model gemacht werden jedoch hier bewusst nicht gemacht um Spezialfälle zu zeigen

bild tree structure

bild kin close

3.2.1 Kinematische Kette schliessen

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

4 Delta

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