Hochschule Pforzheim
Fakultät für Technik

ROS auf dem KUKA youBot

Jonathan Schwarz - schjon02@hs-pforzheim.de

2. September 2013

Inhalt

1 Einführung

1.1 Was ist ROS?

ROS *«Robot Operating System»* ist ein Metabetriebssystem für Roboter. Es stellt typische Dienste eines Betriebssystems wie Hardware-Abstraktion, Gerätetreiber, die Implementation von häufig genutzen Funktionalitäten, die Interprozesskommunikation und das Paketmanagment bereit. Das System beseteht aus verteilten Prozessen (Knoten), die lose gekoppelt werden. ROS ist Open Source und hat sich zum Ziel gesetzt, die Wiederverwendung von Code in der Roboterforschung und -entwicklung zu unterstützen. Es existieren bereits Implementierungen des ROS Frameworks in C++, Python und Lisp. Des Weiteren wird der Softwaretest im Software Engineering Prozess durch ein integriertes Testframework und die intergration von Simulatoren erleichtert. Existierende ROS-Pakete stellen bereits Kernfunktionen der Robotik (Navigation, Manipulation, Wahrnehmung, ...) bereit. Das Paketmanagment und die bereitgestellte Kommunikation zwischen Knoten im System ermöglicht die einfache, parallele Entwicklung von Softwaremodulen, die gemeinsam einen Roboterapplikation bilden. Dies ermöglicht auch die einfache Integration von Fremdsoftware in das eigene Projekt durch definierte Schnittstellen im System. Es existieren u.a. ROS-Treiber für den Aldeberan NAO und Lego Mindstorm NXT Roboter. Des Weiteren lassen sich durch URDF *«Unified Robot Description Format»* selbst gebaute Roboter beschreiben. Dies ermöglicht z.B. die Zusammenarbeit dieser Maschinen. Als Einarbeitung werden an dieser Stelle die ROS Tutorials [ROSte] empfohlen. Für fortgeschrittene Projekte wird [Goe13] als Einführung in bereits implementierte und sehr mächtige ROS-Softwaremodule empfholen.

2 Drahtlose Verbindung

Um eine barrierefreie Steuerung und Überwachung der mobilen Roboterplattform zu gewährleisten, kann über SSH auf den internen PC des youBot zugegriffen werden. Dazu muss der WLAN Stick mit dem Roboter verbunden sein. Im Folgenden sind folgende Schritte durch zu führen:

1. Den internen PC und des steuernden PC im selben Drahtlosnetzwerk verbinden.
2. *IP* des internen PCs herausfinden. Unix-Kommando: **$ ifconfig**
3. Verbindung über SSH herstellen: **$ ssh youbot@*IP***

Nach der Authentifizierung kann jegliche Software auf dem PC ausgeführt werden. Die Drahtlosverbindung erlaubt auch die Steuerung mehrerer Maschinen durch ein ROS-Systems [MUL].

3 ROS Wrapper der youBot API

Die Ansteuerung des youBot Arms und der omnidirektionalen Plattform im ROS Framework erfolgt über das Senden von Nachrichten an ein entprechendes Topic. Abbildung 3 zeigt die Struktur des ROS Wrappers. Die genutzten Nachrichten sind standard ROS-Nachrichten. Ihr Inhalt kann durch den Befehl

$ rosmsg show *Nachricht*.msg

angezeigt werden. Diese Art der Kommunikation wird in Kapitel 5.3 verdeutlicht.

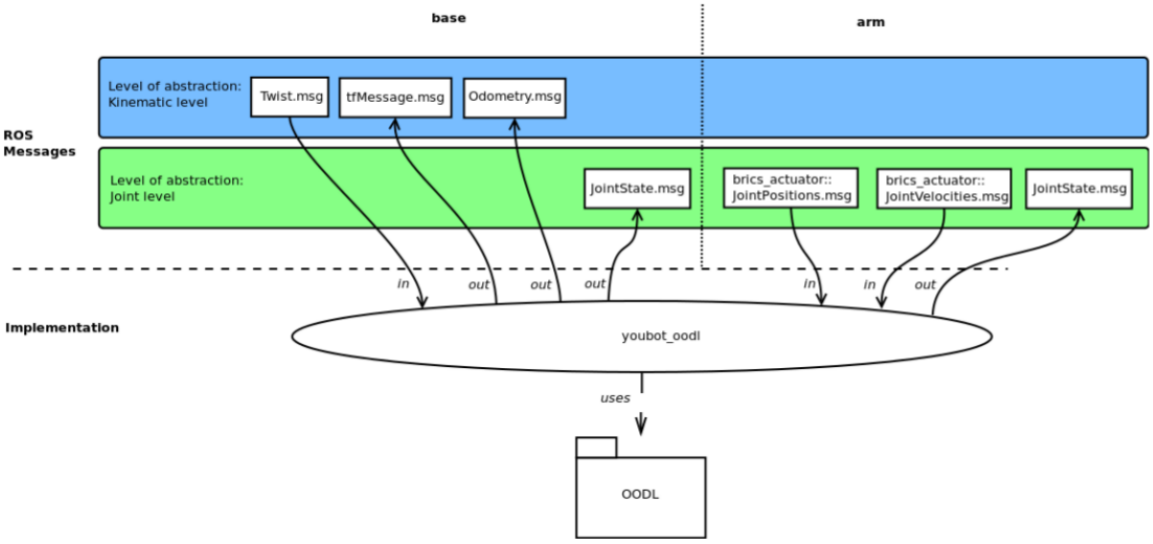

Abbildung 1: Integration der youBot API ins ROS Framework nach [Loc13]

KUKA stellt bereits einige einfache ROS-Applikationen für den Roboter bereit. In diesen quelloffenen Projekten werden alle Achsen des Roboters angespreochen. Ebenfalls ist eine komplette Beschreiung enthalten, die z.b. zur Simulation und Überwachung mit den Programmen *RViz* und *Gazebo* genutzt werden kann.

4 Microsoft Kinect

4.1 Gründe für die Verwendung

Die Verwendung dieser Tiefenkamera erlaubt das Erzeugen einer dynamischen Karte der Umgebung des Roboters. Dies wird benötigt um fortgeschrittene Konzepte wie SLAM *«Simultaneous Localization and Mapping»* zu nutzen. Dies ermöglicht einem Roboter, nicht nur, eine Karte einer unbekannten Umgebung herzustellen, sondern auch die Positionierung in dieser Karte. So kann z.b. Hindernissen ausgewichen werden. Ein einem Multi-Roboter Netzwerk könnten diese Karten zwischen Robotern ausgetauscht werden um eine bisher fremde Umgebung schnell zu 'erforschen'. Alternativ zur Microsoft Kinect kann auch ein Laserscanner verwendet werden, der meist einen deutlich größeren Bereich abdeckt (in etwa min. 240°). Die Microsoft Kinect selbst hat nur ein Sichtfeld von ca. 57°, ist dafür aber deutlich günstiger.

4.2 Treiber

Die Treiber des OpenNI-Projektes sind bereits auf dem Roboter installiert und können mit folgendem Kommando gestartet werden:

$ roslaunch openni_launch openni.launch

Dies führt bei erfolgreicher Ausführung dazu, dass Bilddaten auf entsprechenden ROS Topics veröffentlicht werden. Knoten, die von diesen Topics lesen, können diese Informationen weiter verarbeiten. Wenn die Übertragung problemlos funktioniert, sollte der Befehl

$ rsorun image_view image_view image:=/camera/rgb/image_color

den aktuellen Kamerastream anzeigen.

4.3 Installation einer Kinect Kamera auf dem Roboter

Zum Betreiben der Kamera wird eine externe Stromversorgung von 12 V/1.08 A benötigt. Um diese Spannungsversorgung auf dem Roboter bereitzustellen, kann ein Gleichstromwandler genutzt werden um die 24 V des youBot Anschlusses auf 12 V für den Sensor zu wandeln. Alternativ kann eine Batterie genutzt werden. Die Kamera lässt sich mit einem Träger auf dem Arm des Roboters montieren. Der Träger kann im youBot Store erworben werden.

5 Das ROS Paket *cirlce_detection*

5.1 Beschreibung

Um ein weiteres praktisches Beispiel für die Entwicklung von Roboterapplikationen mit Hilfe des Frameworks zu bieten, wird hier die Implementierung eines ROS Knotens vorgestellt. Angenommen, ein Team von Studenten soll eine Applikation entwickeln, die den Roboter farbige Bälle in einem Kamerabild erkennen und greifen lassen kann. Eine typische Aufteilung des Projektes könnte so vorgenomen werden:

- Wahrnehmung: Bildverarbeitungsalgorithmen, die das Zielobjekt in einem Kamerabild erkennen.
- Navigation: Die Navigation der mobilden Plattform zu einer Position in Greifnähe zum Objekt.
- Manipulation: Die Steuerung des Arms zur Aufnahme des Balls.

Die Kommunikation zwischen den Softwarekomponenten wird durch ROS Nachrichten realisiert. Dabei werden Informationen über Lage und Größe des Balls ausgetauscht. Aus diesen Informationen kann die Entfernung zum Ball berechnet werden (siehe Listing 1).

```
1 string image_id
2 uint32 radius
3 uint16 center_x
4 uint16 center_y
```

Listing 1: Nachrichtendefinition circle_msg

Im Folgenden wird anhand der Teilaufgabe zur Wahrnehmung die Kommunikation im verteilten ROS-System dargestellt. Alle Quellen können durch den git Befehl

$ git clone `https://github.com/jonathan-schwarz/circle_detection_ros`

bezogen werden.

5.2 Projektstruktur

Die Dateistruktur eines ROS Packages wird beim Erzeugen des Paketes hergestellt. Optionale Verzeichnisse werden je nach Bedarf erzeug. Im folgenden Paket (Abbildung 2) wird die Nachricht im Ordner *msg* definiert. ROS erzeugt aus der Beschreibung den C++ Quelltext der Nachricht, wenn dies in der Konfigurationsdatei *CMakeLists.txt* (für cmake) durch den Befehl **rosbuild_genmsg()** explizit angegeben wird. Die Bedeutung aller weiterer Dateien sollte durch die ROS Tutorials bekannt sein.

```
circle_detection/
├── include/
├── src/
├── msg/
├── CMake Lists.txt
├── mainpage.dox
├── Makefile
└── manifest.xml
```

Abbildung 2: Projektstruktur der circle_detection Komponente

5.3 Asychnrone Kommunikation über Topics

Die Softwarearchitekten des Projektes beschließen, dass die Kommunikation zwischen den Komponenten über ein ROS Topic realisert wird, auf dem die in Listing 1 spezifizierten Nachrichten veröffentlicht werden. Sowohl Navigation als auch Manipulation berechnen aus den Daten alle für ihre Ausführung nötigen Parameter.

Listing 2 zeigt alle ROS-Schnittstellen des veröffentlichenden Knoten (Wahrnehmung).

```
1  //...
2
3  #include <circle_detection/circle_msg.h>
4  #include "ros/ros.h"
5
6  //...
7
8  int main(int argc, char** argv)
9  {
10     ros::init(argc,argv, "circle_publisher");
11     // Initializes the node
12     ros::NodeHandle nh;
13
14     /*  Let's publish messages as defined in /msg/cirlce.msg on a topic
15      *  called 'detected_circles' with a max. buffer of 1000 messages
16      */
17     ros::Publisher circle_publisher = nh.advertise<circle_detection::circle_msg>("
           detected_circles",1000);
18
19     // We'll run at 10 hz
20     ros::Rate loop_rate(10);
21
22     // used as Image Id
23     int id_count = 0;
24
25     while(ros::ok)
```

```
26      {
27
28          /*
29           * ... (Image processing)
30           *
31           */
32
33          for(size_t i=0;i < detectedCircles.size();i++)
34          {
35              circle_detection::circle_msg msg;
36              std::stringstream ss;
37
38              ss << "IMG" << id_count;
39
40              msg.image_id = ss.str();
41
42              unsigned int radius = cvRound(detectedCircles[i].radius);
43
44              msg.radius = radius;
45
46              msg.center_x = cvRound(detectedCircles[i].center.x) ;
47              msg.center_y = cvRound(detectedCircles[i].center.y);
48
49              // actually publishes the msg on the topic
50              circle_publisher.publish(msg);
51
52              ros::spinOnce();
53              loop_rate.sleep();
54
55          }
56
57      return 0;
58  }
```

Listing 2: Kommunikation in einem ROS Knoten

Abbildungen

Tabellen

Literatur

[Goe13] Goebel, R. P.: *ROS by example*. Bd. 1 (Electric). 2013

[Loc13] Locomotec: KUKA youBot User Manual. 1.02 (2013), February

[MUL] *Running ROS across multiple machines*. `http://wiki.ros.org/ROS/Tutorials/MultipleMachines`, Abruf: 02.09.2013

[ROSte] *ROS Tutorials*. Version: Fuerte. `http://www.ros.org/wiki/ROS/Tutorials`, Abruf: 29.08.2013