Kostenloses eBook

LERNEN

ros

Free unaffiliated eBook created from
Stack Overflow contributors.

#ros

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ros

It is an unofficial and free ros ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ros.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit ros

Bemerkungen

Robotic Operating System (ROS) ist eine Robotik-Middleware für die Softwareentwicklung von Robotern, die betriebssystemähnliche Funktionen auf heterogenen Computerclustern und Plattformen bietet.

Ursprünglich im Jahr 2007 vom Stanford Artificial Intelligence Laboratory mit Unterstützung des Stanford AI Robot STAIR gegründet, entwickelte sich die Entwicklung von 2008 bis 2013 für Willow Garage, ein Roboterforschungsinstitut. 2013 wechselte ROS Stewardship zur Open Source Robotics Foundation.

ROS stellt Bibliotheken und Tools zur Verfügung, mit denen Softwareentwickler Roboteranwendungen erstellen können. Es bietet Hardware-Abstraktion, Gerätetreiber, Bibliotheken, Visualizer, Nachrichtenübermittlung, Paketverwaltung und mehr. ROS ist unter einer Open Source-BSD-Lizenz lizenziert.

Heute integriert ROS mehr als hundert Roboter (siehe vollständige Roboterliste), von autonomen Fahrzeugen über UAVs bis hin zu humanoiden Robotern und die Verwendung einer Vielzahl von ROS-unterstützten Sensoren (siehe vollständige Sensorliste) ... ROS wird von der ROS stark beansprucht Forschungsgemeinschaft für Servicerobotik-Anwendungen, ihre Technologie kann jedoch auch auf andere Anwendungsbereiche angewendet werden, einschließlich Industrierobotik. Seine Anwendungen wie erweiterte Wahrnehmung, Pfad- / Griff-Planung und Bewegungsverfolgung ermöglichen die Herstellung von Roboteranwendungen, die zuvor technisch nicht durchführbar oder zu teuer waren.

ROS läuft derzeit nur auf Unix-basierten Plattformen. Software für ROS wird hauptsächlich auf Ubuntu- und Mac OS X-Systemen getestet, obwohl die ROS-Community Unterstützung für Fedora, Gentoo, Arch Linux und andere Linux-Plattformen zur Verfügung gestellt hat. Schließlich kann ROS-Codierung in jeder Programmiersprache geschrieben werden, vorausgesetzt, es verfügt über eine Clientbibliothek . Derzeit liegt der Fokus jedoch auf der Bereitstellung einer starken C ++ - und Python-Unterstützung.

ROS präsentiert heute die 10. Version von *ROS Kinetic* .

Weitere Informationen zu den Bemühungen von ROS und ROS Community finden Sie unter http://www.ros.org/.

Versionen

Ros Distro	Unterstützte Ubuntu-Versionen	Veröffentlichungsdatum
Kinetic Kame	15.10, 16.04	2016-05-23
Jade Schildkröte	14.04, 14.10, 15.04	2015-05-23

Ros Distro	Unterstützte Ubuntu-Versionen	Veröffentlichungsdatum
Indigo-Iglu	13.10, 14.04	2014-07-22
Hydro Medusa	12.04, 12.10, 13.04	2013-09-04
Groovige Galapagos	11.10, 12.04, 12.10	2012-12-31
Fuerte Schildkröte	10.04, 11.10, 12.04	2012-04-23
Elektrische Emys	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
C Schildkröte	9.04, 9.10, 10.04, 10.10	2010-08-02
Schildkröte	8.04	2010-03-02

Examples

Installation

Abhängig von Ihrem Zielcomputer müssen Sie eine unterstützte ROS-Version auswählen (oder umgekehrt). Obwohl die Installation von ROS im ROS-Wiki gut dokumentiert ist, kann es verwirrend sein, sie zu finden. Hier ist eine Tabelle der ROS-Version, der Zielplattformen und der Architektur sowie der Links zu den entsprechenden Installationsanleitungen:

ROS-Version	Plattform	Bogen	Status	Installieren Sie den Guide Link
Kinetisch	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	Unterstützt	Kinetic-Xenial-Guide
	Ubuntu 15.10 (Wily)	amd64 / i386	Unterstützt	Kinetic-Wily-Guide
	Debian 8 (Jessie)	amd64 / arm64	Unterstützt	Kinetic-Jessie-Führer
	OS X (Homebrew)	-	Experimental	Kinetic-Homebrew-Führer
	Gentoo	-	Experimental	Kinetic-Gentoo-Führer
	OpenEmbedded / Yocto	-	Experimental	Kinetic-Yocto-Führer

In Arbeit...!

Hallo Weltverleger

Erstellen Sie einen Arbeitsbereich

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

Bauen Sie Ihren Arbeitsbereich auf

```
cd ~/catkin_ws/
catkin_make
```

Quelldatei Ihre Setup-Datei

```
source devel/setup.bash
```

Erstellen Sie ein neues Paket namens hello_world mit einigen grundlegenden Abhängigkeiten

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Navigieren Sie zu Ihrem src-Verzeichnis und erstellen Sie eine neue Datei mit dem Namen talker.cpp

```
cd hello_world/src
touch talker.cpp
```

Bearbeiten Sie Ihre neue Datei und fügen Sie diesen Code ein, um eine "Hallo Welt" -Meldung zu veröffentlichen

```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");

  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);

  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
```

```
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }

  return 0;
}
```

Kehren Sie zum Stammverzeichnis Ihres Paketverzeichnisses zurück

```
cd ..
```

Fügen Sie diese Zeilen zu Ihrer CMakeLists.txt hinzu bzw. entfernen Sie deren Kommentar

```
catkin_package(
 INCLUDE_DIRS include
 LIBRARIES hello_world
#  CATKIN_DEPENDS roscpp rospy std_msgs
#  DEPENDS system_lib
)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker hello_world_generate_messages_cpp)
```

Kehren Sie zum Stamm Ihres Arbeitsbereichs zurück

```
cd ..
```

Bauen Sie Ihren neuen Publisher

```
catkin_make
```

Stellen Sie Ihre Setup-Datei erneut bereit, damit Sie über das neue Paket und den neuen Publisher verfügen

```
source devel/setup.bash
```

Starten Sie ROS

```
roscore
```

Lassen Sie roscore laufen und starten Sie Ihren Publisher in einem neuen Terminal-Fenster

```
rosrun hello_world talker
```

Lassen Sie den Herausgeber laufen und wiederholen Sie die Ausgabe in EINEM ANDEREN Terminal / Fenster

```
rostopic echo /chatter
```

Erste Schritte mit ros online lesen: https://riptutorial.com/de/ros/topic/7287/erste-schritte-mit-ros

Kapitel 2: einen Arbeitsbereich erstellen

Einführung

Dieses Tutorial zeigt, wie Sie einen Arbeitsbereich erstellen. Ein Arbeitsbereich ist eine Reihe von Verzeichnissen, in denen eine verwandte Menge von ROS-Code lebt. Es können mehrere ROS-Arbeitsbereiche vorhanden sein, es ist jedoch möglich, jeweils nur in einem zu arbeiten.

Examples

Arbeitsbereich erstellen

Um einen Arbeitsbereich zu erstellen, sollte man im Terminal Folgendes ausführen:

```
$ mkdir -p ~/workspace_name/src
$ cd ~/workspace_name/src
$ catkin_init_workspace
$ cd ~/workspace_name/
$ catkin_make
```

Mit den vorherigen Befehlen wird ein Arbeitsbereich mit dem Namen `workspace_name` . Nachdem ein Arbeitsbereich erstellt wurde, ist es wichtig, ihn zu beschaffen, um damit arbeiten zu können:

```
$ source ~/workspace_name/devel/setup.bash
```

einen Arbeitsbereich erstellen online lesen: https://riptutorial.com/de/ros/topic/8313/einen-arbeitsbereich-erstellen

Kapitel 3: Paket erstellen

Einführung

Dieses Tutorial zeigt, wie Sie ein Paket in ROS erstellen. Pakete befinden sich in Arbeitsbereichen im Verzeichnis src. Jedes Paketverzeichnis muss eine `CMakeLists.txt` und eine `package.xml` Datei enthalten.

Examples

Ein Paket mit Rospy erstellen

Unter der Annahme, dass zuvor ein Arbeitsbereich namens `workspace_name` im `package_name` erstellt wurde, kann ein Paket namens `package_name` erstellt werden, indem die folgenden Befehlszeilen ausgeführt werden.

```
$ cd ~/workspace_name/src/
$ catkin_create_pkg package_name rospy
```

Paket erstellen online lesen: https://riptutorial.com/de/ros/topic/8314/paket-erstellen

Kapitel 4: Roslaunch

Bemerkungen

"Knoten vom Paket" sollte "Knoten vom Paket" sein

Anfangs sagen Sie "Starten" und "Stoppen", erklären jedoch nicht, wie Sie ein kontrolliertes Herunterfahren durchführen.

Examples

Rosenknoten starten und Parameter aus der Yaml-Datei laden

roslaunch ist ein wichtiges Werkzeug, das das Starten und Stoppen von ROS-Knoten verwaltet. Es benötigt eine oder mehrere "* .launch" -Dateien als Argumente.

In diesem Beispiel werde ich auf Folgendes verweisen (wie in dieser Frage gestellt), also wie können wir diese Befehle nacheinander und automatisch ausführen:

```
roscd stereo_camera
rosparam load marvin_cameras.yaml
rosrun stereo_camera stereo_camera __name:=bumblebeeLeft
rosrun stereo_camera stereo_camera __name:=bumblebeeCenter

roslaunch openni_launch_marvin kinect_left.launch
roslaunch openni_launch_marvin kinect_center.launch
```

Zunächst einmal wollen wir `break up these commands in pieces` . Wie zu sehen ist, werden 4 ros-Befehle benötigt: *roscd* , *rosparam* , *rosrun* und *roslaunch* . Jetzt fangen wir an!

Dies ist ein Beispiel dafür, warum Roslaunch so mächtig ist. Tatsächlich könnten all diese Befehle in `one and only roslaunch file in ROS` . Alles was wir tun müssen, ist, diese "solution.launch" -Datei für einen fortlaufenden und automatischen Aufruf dieser Befehle zu starten.

Um zu starten, basieren Startdateien auf XML-Formatierung. Hier eine grundlegende Startdatei in ROS. Wir nennen sie "basic_example.launch" und sind in einem ROS-Paket mit dem Namen "roslaunch_example" enthalten:

```
<launch>

</launch>
```

Der Befehl zum Ausführen dieser Startdatei lautet

```
$ roslaunch roslaunch_example basic_example.launch
```

nach der Spezifikation:

```
$ roslaunch package_name launch_file_name.launch
```

Da unsere Startdatei keine Befehle enthält, wird nichts ausgeführt, dazu später mehr ...

Knoten in Startdateien einschließen:
Jeder ROS-Knoten in einem installierten ROS-Paket ist in Startdateien aufrufbar. dazu müssen wir das Paket angeben, das den Knoten und seinen Namen enthält, wie im Paket angegeben. zum Beispiel :

```
rosrun stereo_camera stereo_camera __name:=bumblebeeLeft
rosrun stereo_camera stereo_camera __name:=bumblebeeCenter
```

`stereo_camera` ist ein Knoten aus dem Paket `stereo_camera` und die angegebenen Argumente lauten Name `__name:=bumblebeeLeft` und `__name:=bumblebeeCenter` .

Um diese Knoten hinzuzufügen, müssen wir die folgenden Zeilen hinzufügen:

```
<launch>
    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeLeft" />
    </node>

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeCenter" />
    </node>
</launch>
```

Wenn Sie diese Startdatei ausführen, werden die beiden Knoten ausgeführt.

Hinzufügen eines Parameters zu einem ROS-Knoten:

Wie zu sehen ist, haben wir einen Parameter "name" für die Knoten hinzugefügt:

```
<param name="name" value="bumblebeeCenter" />
```

Tatsächlich können wir beliebig viele Parameter hinzufügen (wie oben erstellt) und dann auf sie verweisen, indem wir `"$(arg parameter_name)"` aufrufen, anstatt ihren Wert zu fixieren.

Angabe der Ausgabe:

Das Ausgabe-Tag kann auf "screen" gesetzt werden, wenn Sie das Knotenprotokoll `on the terminal (~/.ros)` oder "log", um das Protokoll in den Protokolldateien in `(~/.ros)` .

Einschließen anderer ROS-Startdateien in eine ROS-Startdatei:

Wie hier angegeben, können Sie mit dem Tag eine weitere Roslaunch-XML-Datei in die aktuelle Datei importieren. Es wird im aktuellen Bereich Ihres Dokuments einschließlich der Tags importiert. Der gesamte Inhalt der Include-Datei wird mit Ausnahme des Tags importiert: Das Tag wird nur in der Datei der obersten Ebene berücksichtigt.

Also alle diese Befehle ausführen

```
roslaunch openni_launch_marvin kinect_left.launch
roslaunch openni_launch_marvin kinect_center.launch
```

Von einer Startdatei müssen wir nur noch folgende Zeilen hinzufügen:

```
<include file="$(find openni_launch_marvin)/launch/kinect_left.launch" />
<include file="$(find openni_launch_marvin)/launch/kinect_center.launch" />
```

> roscd zu einem Paket in der ROS-Startdatei

Um die Startdatei zu finden, die wir einschließen möchten, müssen wir nicht den vollständigen Pfad angeben. Stattdessen stellt roslaunch die Direktive `"$(find package_name)"` . Auf diese Weise können wir auf unsere `"$(find package_name)" relative to the package racine` verweisen.
Im obigen Beispiel ging ich davon aus, dass sich die Datei "kinect_center.launch" im Ordner "openni_launch_marvin) / launch /" befindet.

Parameter aus YAML-Datei laden:

Um Parameter aus einer YAML-Datei in ROS zu laden, stellt ROS das Tag "rosparam" zur Verfügung. Wie im Wiki festgestellt: "Das Tag ermöglicht die Verwendung von Rosparam-YAML-Dateien zum Laden und Ablegen von Parametern vom ROS-Parameterserver. Es kann auch zum Entfernen von Parametern verwendet werden. Das Tag kann in ein Tag eingefügt werden, in diesem Fall das Parameter wird wie ein privater Name behandelt. "

Mit diesem Tag können wir unsere YAML-Datei in die Startdatei laden, indem Sie folgende Zeile hinzufügen:

```
<rosparam command="load" file="$(find marvin_cameras)/config/marvin_cameras.yaml" />
```

Wie oben verwendet, ging ich davon aus, dass sich die YAML-Datei "marvin_cameras.yaml" im Ordner "marvin_cameras / config /" befindet.

Alle Teile zusammenbauen

Nachdem wir nun den Inhalt der Startdatei separat erstellt haben, legen wir sie in einer großen Startdatei "solution.launch" zusammen.

Lösung.Lauf

```
<launch>

    <rosparam command="load" file="$(find marvin_cameras)/config/marvin_cameras.yaml" />

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeLeft" />
    </node>

    <node name="$(arg name)" pkg="stereo_camera" type="stereo_camera" output="screen">
          <param name="name" value="bumblebeeCenter" />
    </node>
```

```
    <include file="$(find openni_launch_marvin)/launch/kinect_left.launch" />
    <include file="$(find openni_launch_marvin)/launch/kinect_center.launch" />

</launch>
```

Jetzt haben wir unsere einzige Roslaunch-Datei, um alle Befehle nacheinander und automatisch auszuführen.

Roslaunch online lesen: https://riptutorial.com/de/ros/topic/7361/roslaunch

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit ros	ckirksey3, Community, Photon, Vtik
2	einen Arbeitsbereich erstellen	lmiguelvargasf
3	Paket erstellen	lmiguelvargasf, Michael
4	Roslaunch	ensonic, Vtik