Бесплатная электронная книга

учусь

ros

Free unaffiliated eBook created from
Stack Overflow contributors.

#ros

Около

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

глава 1: Начало работы с ros

замечания

Роботизированная операционная система (ROS) - это программное обеспечение Robotics Middleware для разработки программного обеспечения роботов, обеспечивающее операционную систему, такую как функциональность на гетерогенных компьютерных кластерах и платформах.

Первоначально начатая в 2007 году Лабораторией искусственного интеллекта в Стэнфорде при поддержке Стэнфордского робота-робота STAIR, с 2008 по 2013 год, была перенесена на исследовательский институт робототехники Willow Garage. В 2013 году руководство ROS перешло в Open Source Robotics Foundation.

ROS предоставляет библиотеки и инструменты, помогающие разработчикам программного обеспечения создавать приложения для роботов. Он обеспечивает аппаратную абстракцию, драйверы устройств, библиотеки, визуализаторы, передачу сообщений, управление пакетами и многое другое. ROS лицензируется по лицензии BSD с открытым исходным кодом.

Сегодня ROS объединяет более сотни роботов (см. Полный список роботов) - от автономных автомобилей до БПЛА до гуманоидных роботов и с использованием множества поддерживаемых ROS-датчиков (см. Полный список датчиков) ... ROS в значительной степени используется исследовательского сообщества для приложений сервисной робототехники, но его технология может применяться в других областях применения, включая промышленную робототехнику. Его приложения, такие как продвинутое восприятие, планирование пути / захвата, отслеживание движения, могут позволить обрабатывать роботизированные приложения, которые ранее были технически неосуществимыми или стоили непомерно высокими.

ROS в настоящее время работает только на платформах на базе Unix. Программное обеспечение для ROS в первую очередь тестируется в системах Ubuntu и Mac OS X, хотя сообщество ROS оказывает поддержку Fedora, Gentoo, Arch Linux и других Linux-платформ. В конце концов, ROS-кодирование может быть записано на любом языке программирования, если у него есть клиентская библиотека , однако в настоящее время основное внимание уделяется обеспечению сильной поддержки C ++ и Python.

ROS сегодня представляет 10-й выпуск *ROS Kinetic* .

Чтобы узнать больше об усилиях сообщества ROS и ROS, посетите http://www.ros.org/

Версии

Ros Distro	Поддерживаемые версии Ubuntu	Дата выхода
Кинетический Каме	15,10, 16,04	2016-05-23
Джейд Черепаха	14.04, 14.10, 15.04	2015-05-23
Индиго Иглу	13,10, 14,04	2014-07-22
Hydro Medusa	12.04, 12.10, 13.04	2013-09-04
Groovy Galapagos	11.10, 12.04, 12.10	2012-12-31
Фуэрте черепаха	10.04, 11.10, 12.04	2012-04-23
Электрические Emys	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
C Черепаха	9,04, 9,10, 10,04, 10,10	2010-08-02
Бокс черепаха	8,04	2010-03-02

Examples

Монтаж

В зависимости от вашей целевой машины вам нужно выбрать поддерживаемую версию ROS (или наоборот). Хотя установка ROS хорошо документирована в ROS-вики, возможно, будет сложно найти их. Итак, вот таблица версии ROS, целевых платформ и архитектуры и ссылок для соответствующих руководств по установке:

Версия ROS	Платформа	арочный	Статус	Ссылка на установочный гид
кинетический	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	поддержанный	Кинетический-гид-дружественного
	Ubuntu 15.10 (Wily)	amd64 / i386	поддержанный	Кинетическая-Вили-гид
	Debian 8 (Jessie)	amd64 / arm64	поддержанный	Кинетический-Jessie-гид
	OS X (Homebrew)	-	экспериментальный	Кинетическая-Homebrew-гид

Версия ROS	Платформа	арочный	Статус	Ссылка на установочный гид
	Gentoo	-	экспериментальный	Кинетический-Gentoo-гид
	OpenEmbedded / Yocto	-	экспериментальный	Кинетический-Yocto-гид

Работа в процессе...!

Hello World Publisher

Создание рабочего пространства

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

Создайте рабочее пространство

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

Отправьте исходный файл установки

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

Создайте новый пакет с именем hello_world с некоторыми базовыми зависимостями

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Перейдите в каталог src и создайте новый файл с именем talker.cpp

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

Отредактируйте новый файл и вставьте этот код, чтобы опубликовать сообщение «привет мир»

```
#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;
}
```

Вернитесь в корень вашего каталога пакетов

```
cd ..
```

Добавьте / раскомментируйте эти строки в CMakeLists.txt

```
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)
```

Вернитесь к корню вашей рабочей области

```
cd ..
```

Создайте свой новый издатель

```
catkin_make
```

Обновите свой файл настроек, чтобы у вас появился новый пакет и издатель

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

Запустить ROS

```
roscore
```

Оставьте roscore, и в новой вкладке / окне терминала запустите свой издатель

```
rosrun hello_world talker
```

Оставьте работу издателя и в другой вкладке / октете новых терминалов, выполните эхо-вывод

```
rostopic echo /chatter
```

Прочитайте Начало работы с ros онлайн: https://riptutorial.com/ru/ros/topic/7287/начало-работы-с-ros

глава 2: roslaunch

замечания

«node form the package» должен быть «узлом из пакета»,

изначально ваши слова «start» и «Stopping», но вы не объясняете, как сделать контролируемое завершение работы.

Examples

запускать ros-узлы и параметры загрузки из файла Yaml

roslaunch - это важный инструмент, который управляет запуском и остановкой узлов ROS. В качестве аргументов он принимает один или несколько файлов «.launch».

В этом примере я буду ссылаться на следующее (как задано в этом вопросе), так как мы можем выполнить эти команды последовательно и автоматически:

```
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
```

Прежде всего , давайте `break up these commands in pieces` . Как видно, необходимы 4 команды ros: *roscd* , *rosparam* , *rosrun* и *roslaunch* . Теперь начнем!

Это пример того, почему roslaunch является мощным. Фактически, все эти команды могут быть включены в `one and only roslaunch file in ROS` . то все, что нам нужно сделать, - запустить этот файл «solution.launch» для последовательного и автоматического вызова этих команд.

Для начала файлы запуска основаны на форматировании XML, вот основной файл запуска в ROS, мы назовем его «basic_example.launch» и включим его в пакет ROS с именем « roslaunch_example»:

```
<launch>

</launch>
```

команда для запуска этого файла запуска

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

следуя спецификации:

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

Из-за нашего файла запуска не включаются никакие команды, ничего не будет выполнено, подробнее об этом позже ...

Включение узлов в файлы запуска:
любой ROS-узел в любом установленном ROS-пакете имеет возможность вызова в файлах запуска. к этому мы должны указать пакет, содержащий узел, и его имя, как указано в пакете. например :

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

`stereo_camera` - это узел, формирующий пакет `stereo_camera` а указанные аргументы - это имя `__name:=bumblebeeLeft` `__name:=bumblebeeCenter` и `__name:=bumblebeeCenter` .

Чтобы добавить эти узлы, мы должны добавить следующие строки:

```
<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>
```

Выполняя этот файл запуска, мы будем запускать два узла.

добавление параметра в узел ROS:

Как можно видеть, мы добавили параметр «имя» для узлов как:

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

Фактически, мы можем добавить столько параметров, сколько хотим (как создано выше), а затем обратиться к ним, вызвав `"$(arg parameter_name)"` вместо фиксации его значения.

Указание вывода:

Выходной тег может быть установлен на «экран», если вам нужно увидеть журнал узла `on the terminal` или «журнал», чтобы сохранить журнал в файлах журнала в `(~/.ros)` .

Включая другие файлы запуска ROS в файл запуска ROS:

Как указано здесь , тег позволяет импортировать другой файл XML roslaunch в текущий файл. Он будет импортирован в текущую область вашего документа, включая и теги. Все содержимое в include-файле будет импортировано за исключением тега: тег выполняется только в файле верхнего уровня.

Итак, все для выполнения этих команд

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

из файла запуска все, что нам нужно сделать, это добавить следующие строки:

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

roscd в пакет в файле запуска ROS

Чтобы найти файл запуска, который мы хотим включить, нам не нужно указывать полный путь. Вместо этого roslaunch предоставляет директиву `"$(find package_name)"` , таким образом, мы можем ссылаться на наш файл запуска `relative to the package racine` .
В приведенном выше примере я предположил, что файл «kinect_center.launch» находится в папке «openni_launch_marvin» / launch / ".

Загрузка параметров из файла YAML:

Чтобы загрузить параметры из файла YAML в ROS, ROS предоставляет тег «rosparam». Как указано в wiki : «Тег позволяет использовать файлы YAML rosparam для загрузки и сброса параметров с сервера параметров ROS, а также для удаления параметров. Тег можно поместить внутри тега, и в этом случае параметр рассматривается как личное имя. "

Используя этот тег, мы можем загрузить наш файл YAML в файл запуска, добавив эту строку:

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

Как использовано выше, я предположил, что файл YAML «marvin_cameras.yaml» находится в папке «marvin_cameras / config /».

Сборка всех частей

Теперь, когда мы создали отдельно наше содержимое файла запуска, давайте соберем их в одном большом файле запуска «solution.launch».

solution.launch

```
<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>
```

Теперь у нас есть один и единственный файл roslaunch для выполнения всех команд последовательно и автоматически.

Прочитайте roslaunch онлайн: https://riptutorial.com/ru/ros/topic/7361/roslaunch

глава 3: Создание пакета

Вступление

В этом руководстве показано, как создать пакет в ROS. Пакеты размещаются внутри рабочих областей, в каталоге src. Каждый каталог пакета должен иметь `CMakeLists.txt` и файлы `package.xml` .

Examples

Создание пакета с использованием rospy

Предполагая, что рабочее пространство с именем `workspace_name` было ранее создано в домашнем каталоге, пакет с именем `package_name` может быть создан путем выполнения следующих команд.

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

Прочитайте Создание пакета онлайн: https://riptutorial.com/ru/ros/topic/8314/создание-пакета

глава 4: создание рабочего пространства

Вступление

В этом уроке показано, как создать рабочее пространство. Рабочая область - это набор каталогов, в которых живет связанный набор кода ROS. Можно иметь несколько рабочих пространств ROS, но можно работать только по одному за раз.

Examples

Создание рабочей области

Чтобы создать рабочее пространство, необходимо запустить в терминале следующее:

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

Предыдущие команды создают рабочее пространство с именем `workspace_name` . Как только рабочая область была создана, важно ее загрузить, чтобы работать с ней:

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

Прочитайте создание рабочего пространства онлайн: https://riptutorial.com/ru/ros/topic/8313/создание-рабочего-пространства

кредиты

S. No	Главы	Contributors
1	Начало работы с ros	ckirksey3, Community, Photon, Vtik
2	roslaunch	ensonic, Vtik
3	Создание пакета	lmiguelvargasf, Michael
4	создание рабочего пространства	lmiguelvargasf