EBook Gratis

APRENDIZAJE

ros

Free unaffiliated eBook created from
Stack Overflow contributors.

#ros

Tabla de contenido

Acerca de

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capítulo 1: Empezando con ros

Observaciones

El Sistema Operativo Robótico (ROS) es un software de robótica Middleware para el desarrollo de software de robots que proporciona funcionalidades similares a las de un sistema operativo en agrupaciones y plataformas informáticas heterogéneas.

Iniciado originalmente en 2007 por el Laboratorio de Inteligencia Artificial de Stanford en el soporte de Stanford AI Robot STAIR, el desarrollo, de 2008 a 2013, migró para realizarse en Willow Garage, un instituto de investigación en robótica. En 2013, ROS administró la transición a la Open Source Robotics Foundation.

ROS proporciona bibliotecas y herramientas para ayudar a los desarrolladores de software a crear aplicaciones de robot. Proporciona abstracción de hardware, controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, administración de paquetes y más. ROS está bajo una licencia de código abierto, BSD.

Hoy, ROS integra más de un centenar de robots (consulte la lista completa de robots), desde autos autónomos hasta vehículos aéreos no tripulados (UAV) y robots humanoides, y utiliza una multitud de sensores compatibles con ROS (consulte la lista completa de sensores) ... ROS es muy utilizado por Comunidad de investigación para aplicaciones de robótica de servicio, pero su tecnología se puede aplicar a otras áreas de aplicación, incluida la robótica industrial. Sus aplicaciones como la percepción avanzada, la planificación de la trayectoria / comprensión, el seguimiento del movimiento pueden permitir la fabricación de aplicaciones robóticas que antes no eran técnicamente factibles o tenían un costo prohibitivo.

ROS actualmente solo se ejecuta en plataformas basadas en Unix. El software para ROS se prueba principalmente en sistemas Ubuntu y Mac OS X, aunque la comunidad ROS ha estado contribuyendo con el soporte para Fedora, Gentoo, Arch Linux y otras plataformas Linux. Eventualmente, la codificación ROS se puede escribir en cualquier lenguaje de programación siempre que tenga su biblioteca cliente , sin embargo, el enfoque actual es proporcionar un soporte sólido de C ++ y Python.

ROS presenta hoy su décima versión de *ROS Kinetic* .

Para obtener más información sobre los esfuerzos de la comunidad ROS y ROS, visite http://www.ros.org/

Versiones

Ros Distro	Versiones de Ubuntu soportadas	Fecha de lanzamiento
Kame cinético	15.10, 16.04	2016-05-23
Tortuga de jade	14.04, 14.10, 15.04	2015-05-23

Ros Distro	Versiones de Ubuntu soportadas	Fecha de lanzamiento
Índigo Igloo	13,10, 14,04	2014-07-22
Hidro medusa	12.04, 12.10, 13.04	2013-09-04
Groovy Galapagos	11.10, 12.04, 12.10	2012-12-31
Tortuga Fuerte	10.04, 11.10, 12.04	2012-04-23
Emys electricos	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
Tortuga c	9.04, 9.10, 10.04, 10.10	2010-08-02
Tortuga de caja	8.04	2010-03-02

Examples

Instalación

Dependiendo de su máquina de destino, debe elegir una versión de ROS compatible (o viceversa). Aunque la instalación de ROS está bien documentada en la wiki de ROS, puede ser confuso encontrarlos. Entonces, aquí hay una tabla de la versión de ROS, las plataformas y la arquitectura de destino y los enlaces para las guías de instalación apropiadas:

Versión ROS	Plataforma	Arco	Estado	Instalar guía de enlace
Cinético	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	Soportado	Guía cinética-xenial
	Ubuntu 15.10 (astuto)	amd64 / i386	Soportado	Guía cinética-astuta
	Debian 8 (Jessie)	amd64 / arm64	Soportado	Cinética-Jessie-guía
	OS X (Homebrew)	-	Experimental	Cinética-Homebrew-guía
	Gentoo	-	Experimental	Cinética-gentoo-guía
	OpenEmbedded / Yocto	-	Experimental	Cinética-yocto-guía

Trabajo en progreso...!

Hola World Publisher

Crear un espacio de trabajo

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

Construye tu espacio de trabajo

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

Fuente de su archivo de configuración

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

Cree un nuevo paquete llamado hello_world con algunas dependencias básicas

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Navegue a su directorio src y cree un nuevo archivo llamado talker.cpp

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

Edite su nuevo archivo y pegue este código para publicar un mensaje de "hola mundo"

```
#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;
}
```

Volver a la raíz de su directorio de paquetes

```
cd ..
```

Agregue / descomente estas líneas a su 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)
```

Regresa a la raíz de tu espacio de trabajo.

```
cd ..
```

Construye tu nuevo editor

```
catkin_make
```

Fuente su archivo de configuración de nuevo para que tenga el nuevo paquete y el editor

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

Iniciar ros

```
roscore
```

Deje Roscore en ejecución y en una nueva pestaña / ventana de terminal, inicie su editor

```
rosrun hello_world talker
```

Deje el editor en ejecución y en OTRA nueva pestaña / ventana de terminal, haga eco de la salida

```
rostopic echo /chatter
```

Lea Empezando con ros en línea: https://riptutorial.com/es/ros/topic/7287/empezando-con-ros

Capítulo 2: creando un espacio de trabajo

Introducción

Este tutorial muestra cómo crear un espacio de trabajo. Un espacio de trabajo es un conjunto de directorios en el que vive un conjunto relacionado de código ROS. Uno puede tener múltiples espacios de trabajo de ROS, pero es posible trabajar solo de uno en uno.

Examples

Creando un espacio de trabajo

Para crear un espacio de trabajo, se debe ejecutar lo siguiente en el terminal:

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

Los comandos anteriores crean un espacio de trabajo llamado `workspace_name` . Una vez que se ha creado un espacio de trabajo, es importante buscarlo para poder trabajar con él:

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

Lea creando un espacio de trabajo en línea: https://riptutorial.com/es/ros/topic/8313/creando-un-espacio-de-trabajo

Capítulo 3: Creando un paquete

Introducción

Este tutorial muestra cómo crear un paquete en ROS. Los paquetes se encuentran dentro de los espacios de trabajo, en el directorio src. Cada directorio de paquetes debe tener un `CMakeLists.txt` y un `package.xml` .

Examples

Creando un paquete usando rospy

Suponiendo que un espacio de trabajo llamado `workspace_name` se haya creado previamente en el directorio de inicio, se puede crear un paquete llamado `package_name` ejecutando las siguientes líneas de comando.

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

Lea Creando un paquete en línea: https://riptutorial.com/es/ros/topic/8314/creando-un-paquete

Capítulo 4: roslaunch

Observaciones

'nodo desde el paquete' debe ser 'nodo desde el paquete'

Inicialmente, usted dice "comenzando" y "deteniéndose", pero no explica cómo hacer un apagado controlado.

Examples

lanza los nodos ros y carga los parámetros desde el archivo Yaml

roslaunch es una herramienta importante que gestiona el inicio y la detención de los nodos ROS. Toma uno o más archivos "* .launch" como argumentos.

Para este ejemplo, me referiré a lo siguiente (como se pregunta en esta pregunta), entonces, ¿cómo podemos ejecutar esos comandos de forma consecutiva y automática?

```
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
```

En primer lugar , vamos a `break up these commands in pieces` . Como puede verse, se necesitan 4 comandos ros: *roscd* , *rosparam* , *rosrun* y *roslaunch* . Ahora vamos a empezar!

Este es un ejemplo de por qué roslaunch es poderoso. De hecho, todos esos comandos podrían incluirse en `one and only roslaunch file in ROS` . Luego, todo lo que tenemos que hacer es lanzar este archivo "solution.launch" para una llamada consecutiva y automática de esos comandos.

Para comenzar, los archivos de inicio se basan en el formato XML, aquí hay un archivo de inicio básico en ROS, lo llamaremos "basic_example.launch" y se incluirá en un paquete de ROS llamado "roslaunch_example":

```
<launch>

</launch>
```

El comando para ejecutar este archivo de inicio es

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

siguiendo la especificación:

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

Debido a que nuestro archivo de inicio no incluye ningún comando, no se ejecutará nada, más sobre esto más adelante ...

Incluyendo nodos en los archivos de inicio:
cualquier nodo ROS en cualquier paquete ROS instalado puede activarse en los archivos de inicio. para eso tenemos que especificar el paquete que contiene el nodo y su nombre como se especifica en el paquete. por ejemplo :

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

`stereo_camera` es un nodo del paquete `stereo_camera` y los argumentos especificados son su nombre `__name:=bumblebeeLeft` `__name:=bumblebeeCenter` y `__name:=bumblebeeCenter` .

Para agregar esos nodos, tenemos que agregar las siguientes líneas:

```
<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>
```

Al ejecutar este archivo de inicio, tendremos los dos nodos en ejecución.

añadiendo un parámetro a un nodo ROS:

Como se puede ver, agregamos un parámetro "nombre" para los nodos como:

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

De hecho, podemos agregar tantos parámetros como queramos (como se creó anteriormente) y luego referirnos a ellos llamando a `"$(arg parameter_name)"` lugar de arreglar su valor.

Especificando la salida:

La etiqueta de salida se puede establecer en "pantalla", si necesita ver el registro de nodo `on the terminal` o "registro" para guardar el registro en los archivos de registro en `(~/.ros)` .

Incluyendo otros archivos de lanzamiento de ROS en un archivo de lanzamiento de ROS:

Como se indica aquí , la etiqueta le permite importar otro archivo XML de lanzamiento en el archivo actual. Se importará dentro del alcance actual de su documento, incluidas las etiquetas. Todo el contenido del archivo de inclusión se importará, excepto la etiqueta: la etiqueta solo se obedece en el archivo de nivel superior.

Así que, todo para ejecutar estos comandos.

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

Desde un archivo de inicio, todo lo que tenemos que hacer es agregar las siguientes líneas:

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

> roscd a un paquete en el archivo de lanzamiento de ROS

Para encontrar el archivo de inicio que queremos incluir, no necesitamos especificar la ruta completa. En su lugar, roslaunch proporciona la directiva `"$(find package_name)"` , de esta manera, podemos referirnos a nuestro archivo de inicio en `relative to the package racine .`
En el ejemplo anterior, asumí que el archivo "kinect_center.launch" está en la carpeta "openni_launch_marvin) / launch /".

Cargando parámetros desde el archivo YAML:

Para cargar los parámetros de un archivo YAML en ROS, ROS proporciona la etiqueta "rosparam". Como se indica en la wiki : "La etiqueta permite el uso de archivos YAML de rosparam para cargar y descargar parámetros desde el servidor de parámetros de ROS. También se puede usar para eliminar parámetros. La etiqueta se puede colocar dentro de una etiqueta, en cuyo caso parámetro es tratado como un nombre privado ".

Usando esta etiqueta, podemos cargar nuestro archivo YAML en el archivo de inicio agregando esta línea:

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

Como se usó anteriormente, asumí que el archivo YAML "marvin_cameras.yaml" está en la carpeta "marvin_cameras / config /".

Ensamblando todas las piezas

Ahora que hemos creado por separado los contenidos de nuestro archivo de inicio, unámoslos en un gran archivo de inicio "solution.launch".

solucion

```
<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>
```

Ahora, tenemos nuestro único y único archivo para ejecutar todos los comandos de forma consecutiva y automática.

Lea roslaunch en línea: https://riptutorial.com/es/ros/topic/7361/roslaunch

Creditos

S. No	Capítulos	Contributors
1	Empezando con ros	ckirksey3, Community, Photon, Vtik
2	creando un espacio de trabajo	lmiguelvargasf
3	Creando un paquete	lmiguelvargasf, Michael
4	roslaunch	ensonic, Vtik