EBook Gratuito

APPRENDIMENTO

ros

Free unaffiliated eBook created from
Stack Overflow contributors.

#ros

Sommario

Di

Capitolo 1: Iniziare con Ros

Osservazioni

Robotic Operating System (ROS) è un middleware di robotica per lo sviluppo di software per robot che fornisce funzionalità come sistemi operativi su cluster e piattaforme di computer eterogenei.

Originariamente avviato nel 2007 dal Laboratorio di Intelligenza Artificiale di Stanford nel supporto dello Stanford AI Robot STAIR, lo sviluppo, dal 2008 al 2013, è migrato per essere eseguito presso la Willow Garage, un istituto di ricerca di robotica. Nel 2013, la gestione delle ROS è passata alla Open Source Robotics Foundation.

ROS fornisce librerie e strumenti per aiutare gli sviluppatori di software a creare applicazioni robot. Fornisce l'astrazione hardware, driver di dispositivo, librerie, visualizzatori, trasmissione di messaggi, gestione dei pacchetti e altro ancora. ROS è concesso in licenza con licenza open source BSD.

Oggi, ROS integra più di un centinaio di robot (vedi la lista completa dei robot), che vanno dalle auto autonome, agli UAV ai robot umanoidi e utilizzando una moltitudine di sensori supportati da ROS (vedi elenco completo dei sensori) ... ROS è fortemente utilizzato dal comunità di ricerca per applicazioni di robotica di servizio, ma la sua tecnologia può essere applicata ad altre aree di applicazione, compresa la robotica industriale. Le sue applicazioni come la percezione avanzata, la pianificazione del percorso / della presa, il rilevamento del movimento possono consentire applicazioni robotiche di produzione che prima erano tecnicamente irrealizzabili o proibitive.

Attualmente ROS funziona solo su piattaforme basate su Unix. Il software per ROS è principalmente testato su sistemi Ubuntu e Mac OS X, sebbene la comunità ROS abbia contribuito al supporto di Fedora, Gentoo, Arch Linux e altre piattaforme Linux. Alla fine, la codifica ROS può essere scritta in qualsiasi linguaggio di programmazione, purché abbia la libreria client , tuttavia, l'obiettivo attuale è fornire un forte supporto C ++ e Python.

ROS presenta oggi la sua decima uscita *ROS Kinetic* .

Per ulteriori informazioni sugli sforzi della comunità ROS e ROS, visitare http://www.ros.org/

Versioni

Ros Distro	Versioni di Ubuntu supportate	Data di rilascio
Kinetic Kame	15.10, 16.04	2016/05/23
Jade Turtle	14.04, 14.10, 15.04	2015/05/23
Igloo indaco	13.10, 14.04	2014/07/22

Ros Distro	Versioni di Ubuntu supportate	Data di rilascio
Hydro Medusa	12.04, 12.10, 13.04	2013/09/04
Groovy Galapagos	11.10, 12.04, 12.10	2012-12-31
Fuerte Turtle	10.04, 11.10, 12.04	2012-04-23
Emys elettrico	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
C Tartaruga	9.04, 9.10, 10.04, 10.10	2010-08-02
Scatola tartaruga	8.04	2010-03-02

Examples

Installazione

A seconda del computer di destinazione, è necessario scegliere una versione ROS supportata (o viceversa). Sebbene l'installazione di ROS sia ben documentata nel wiki di ROS, potrebbe essere difficile trovarli. Quindi, ecco una tabella della versione ROS, piattaforme di destinazione e architettura e i collegamenti per le guide di installazione appropriate:

Versione ROS	piattaforma	Arco	Stato	Installa il link della guida
cinetico	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	supportato	Kinetic-Xenial-guida
	Ubuntu 15.10 (Wily)	amd64 / i386	supportato	Kinetic-Wily-guida
	Debian 8 (Jessie)	amd64 / arm64	supportato	Kinetic-Jessie-guida
	OS X (Homebrew)	-	Sperimentale	Kinetic-Homebrew-guida
	Gentoo	-	Sperimentale	Kinetic-Gentoo-guida
	OpenEmbedded / Yocto	-	Sperimentale	Kinetic-Yocto-guida

Lavori in corso...!

Ciao World Publisher

Crea uno spazio di lavoro

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

Costruisci il tuo spazio di lavoro

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

Dai il tuo file di installazione

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

Crea un nuovo pacchetto chiamato hello_world con alcune dipendenze di base

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Passare alla directory src e creare un nuovo file denominato talker.cpp

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

Modifica il tuo nuovo file e incolla questo codice per pubblicare un messaggio "ciao mondo"

```
#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;
}
```

Torna alla radice della directory del pacchetto

```
cd ..
```

Aggiungi / Rimuovi queste righe al tuo 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)
```

Torna alla radice del tuo spazio di lavoro

```
cd ..
```

Costruisci il tuo nuovo editore

```
catkin_make
```

Inserisci di nuovo il file di installazione in modo da disporre del nuovo pacchetto e del nuovo editore

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

Inizia ROS

```
roscore
```

Lascia in esecuzione roscore e in una nuova scheda / finestra del terminale, avvia il tuo editore

```
rosrun hello_world talker
```

Lascia in esecuzione il publisher e in ALTRO una nuova scheda / finestra del terminale, fai eco all'output

```
rostopic echo /chatter
```

Leggi Iniziare con Ros online: https://riptutorial.com/it/ros/topic/7287/iniziare-con-ros

Capitolo 2: creando uno spazio di lavoro

introduzione

Questo tutorial mostra come creare uno spazio di lavoro. Un workspace è un insieme di directory in cui vive un set correlato di codice ROS. Uno può avere più spazi di lavoro ROS, ma è possibile lavorare solo uno alla volta.

Examples

Creare un workspace

Per creare uno spazio di lavoro, è necessario eseguire quanto segue nel terminale:

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

I comandi precedenti creano uno spazio di lavoro denominato `workspace_name` . Una volta creato uno spazio di lavoro, è importante procurarselo per poter lavorare con esso:

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

Leggi creando uno spazio di lavoro online: https://riptutorial.com/it/ros/topic/8313/creando-uno-spazio-di-lavoro

Capitolo 3: Creare un pacchetto

introduzione

Questo tutorial mostra come creare un pacchetto in ROS. I pacchetti si trovano all'interno degli spazi di lavoro, nella directory src. Ogni directory del pacchetto deve avere un file `CMakeLists.txt` e un file `package.xml` .

Examples

Creare un pacchetto usando Rospy

Supponendo che uno spazio di lavoro denominato `workspace_name` sia stato precedentemente creato nella directory home, è possibile creare un pacchetto denominato `package_name` eseguendo le seguenti righe di comando.

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

Leggi Creare un pacchetto online: https://riptutorial.com/it/ros/topic/8314/creare-un-pacchetto

Capitolo 4: roslaunch

Osservazioni

'node form the package' dovrebbe essere 'nodo dal pacchetto'

inizialmente dici "avvio" e "arresto", ma non spieghi come eseguire uno spegnimento controllato.

Examples

avviare i nodi Ros e caricare i parametri dal file Yaml

roslaunch è uno strumento importante che gestisce l'avvio e l'arresto dei nodi ROS. Prende uno o più file "* .launch" come argomenti.

Per questo esempio, farò riferimento a quanto segue (come richiesto in questa domanda), quindi come possiamo eseguire questi comandi in modo consecutivo e automatico:

```
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
```

Prima di tutto , `break up these commands in pieces` . Come si può vedere, sono necessari 4 comandi ros: *roscd* , *rosparam* , *rosrun* e *roslaunch* . Ora iniziamo!

Questo è un esempio del perché roslaunch è potente, infatti tutti questi comandi potrebbero essere inclusi in `one and only roslaunch file in ROS` . allora tutto ciò che dobbiamo fare è lanciare questo file "solution.launch" per una chiamata consecutiva e automatica per quei comandi.

Per iniziare, i file di avvio sono basati sulla formattazione XML, ecco un file di avvio di base in ROS, lo chiameremo "basic_example.launch" ed è incluso in un pacchetto ROS chiamato "roslaunch_example":

```
<launch>

</launch>
```

il comando per eseguire questo file di avvio

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

seguendo le specifiche:

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

A causa del nostro file di avvio non include alcun comando, non verrà eseguito nulla, ne riparleremo più avanti ...

Compresi i nodi nei file di avvio:
qualsiasi nodo ROS in qualsiasi pacchetto ROS installato è richiamabile nei file di avvio. a questo dobbiamo specificare il pacchetto contenente il nodo e il suo nome come specificato nel pacchetto. per esempio :

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

`stereo_camera` è un nodo del pacchetto `stereo_camera` e gli argomenti specificati sono il nome `__name:=bumblebeeLeft` e `__name:=bumblebeeCenter` .

Per aggiungere quei nodi, dobbiamo aggiungere le seguenti linee:

```
<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>
```

Eseguendo questo file di avvio, avremo i due nodi in esecuzione.

aggiunta di un parametro a un nodo ROS:

Come si può vedere, abbiamo aggiunto un parametro "nome" per i nodi come:

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

In effetti, possiamo aggiungere tutti i parametri che vogliamo (come creati sopra) e quindi fare riferimento ad essi chiamando `"$(arg parameter_name)"` invece di correggerne il valore.

Specifica dell'output:

Il tag di output può essere impostato su "screen", se è necessario visualizzare il log del nodo `on the terminal` o "log" per salvare il log nei file di log in `(~/.ros)` .

Compreso altri file di avvio ROS in un file di avvio di ROS:

Come indicato qui , il tag consente di importare un altro file XML Roslaunch nel file corrente. Verrà importato nell'ambito corrente del tuo documento, inclusi e tag. Tutto il contenuto nel file di inclusione verrà importato, ad eccezione del tag: il tag è solo rispettato nel file di livello superiore.

Quindi, tutto per eseguire questi comandi

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

da un file di avvio, tutto ciò che dobbiamo fare è aggiungere le seguenti righe:

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

roscd in un pacchetto nel file di avvio ROS

Per trovare il file di avvio di quello che vogliamo includere, non è necessario specificare il percorso completo. Invece, roslaunch fornisce la direttiva `"$(find package_name)"` , in questo modo possiamo fare riferimento al nostro file di lancio `relative to the package racine` .
Nell'esempio sopra, ho assunto che il file "kinect_center.launch" si trovi nella cartella "openni_launch_marvin) / launch /".

Caricamento dei parametri dal file YAML:

Per caricare i parametri da un file YAML in ROS, ROS fornisce il tag "rosparam". Come affermato nella wiki : "Il tag abilita l'uso di file YAML di rosparam per caricare e scaricare i parametri dal ROS Parameter Server e può anche essere usato per rimuovere i parametri.Il tag può essere inserito all'interno di un tag, nel qual caso il tag parametro è trattato come un nome privato. "

Usando questo tag, possiamo caricare il nostro file YAML nel file di avvio aggiungendo questa riga:

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

Come sopra, ho presupposto che il file YAML "marvin_cameras.yaml" si trovi nella cartella "marvin_cameras / config /".

Assemblando tutti i pezzi

Ora che abbiamo creato separatamente i contenuti del nostro file di avvio, assembliamoli in un unico file di avvio "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>
```

Ora, abbiamo il nostro unico file Roslaunch per eseguire tutti i comandi in modo consecutivo e automatico.

Leggi roslaunch online: https://riptutorial.com/it/ros/topic/7361/roslaunch

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Ros	ckirksey3, Community, Photon, Vtik
2	creando uno spazio di lavoro	lmiguelvargasf
3	Creare un pacchetto	lmiguelvargasf, Michael
4	roslaunch	ensonic, Vtik