eBook Gratuit

APPRENEZ ros

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#ros

Table des matières

À propos

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

Chapitre 1: Commencer avec ros

Remarques

Robotic Operating System (ROS) est un logiciel robotique pour le développement de logiciels de robots fournissant des fonctionnalités de type système d'exploitation sur des plates-formes informatiques hétérogènes.

Initialement lancé en 2007 par le Stanford Artificial Intelligence Laboratory dans le cadre du soutien de Stanford AI Robot STAIR, le développement, de 2008 à 2013, a été effectué au Willow Garage, un institut de recherche en robotique. En 2013, la gestion de ROS est passée à la Open Source Robotics Foundation.

ROS fournit des bibliothèques et des outils pour aider les développeurs de logiciels à créer des applications de robot. Il fournit l'abstraction matérielle, les pilotes de périphérique, les bibliothèques, les visualiseurs, la transmission des messages, la gestion des packages, etc. ROS est sous licence open source, licence BSD.

Aujourd'hui, ROS intègre plus d'une centaine de robots (voir la liste complète des robots), allant des voitures autonomes aux drones, en passant par des robots humanoïdes et une multitude de capteurs supportés par ROS (voir la liste complète des capteurs). communauté de recherche pour les applications de robotique de service, mais sa technologie peut être appliquée à d'autres domaines d'application, y compris la robotique industrielle. Ses applications telles que la perception avancée, la planification de trajectoire / compréhension, le suivi de mouvement peuvent permettre de fabriquer des applications robotiques qui étaient auparavant techniquement irréalisables ou coûteuses.

ROS ne fonctionne actuellement que sur les plates-formes Unix. Les logiciels pour ROS sont principalement testés sur les systèmes Ubuntu et Mac OS X, bien que la communauté ROS ait fourni un support pour Fedora, Gentoo, Arch Linux et d'autres plates-formes Linux. Finalement, le codage ROS peut être écrit dans n'importe quel langage de programmation, à condition qu'il s'agisse de sa bibliothèque cliente , cependant, l'accent est actuellement mis sur la fourniture d'un support C ++ et Python puissant.

ROS présente aujourd'hui sa 10e édition *ROS Kinetic* .

Pour en savoir plus sur les efforts de ROS et de la communauté ROS, visitez le site http://www.ros.org/

Versions

Ros Distro	Versions Ubuntu prises en charge	Date de sortie
Kinetic Kame	15.10, 16.04	2016-05-23
Tortue de jade	14.04, 14.10, 15.04	2015-05-23

Ros Distro	Versions Ubuntu prises en charge	Date de sortie
Igloo Indigo	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
Tortue Fuerte	10.04, 11.10, 12.04	2012-04-23
Emys électriques	10.04, 10.10, 11.04, 11.10	2011-08-30
Diamondback	10.04, 10.10, 11.04	2011-03-02
C Tortue	9.04, 9.10, 10.04, 10.10	2010-08-02
Tortue-boîte	8.04	2010-03-02

Examples

Installation

Selon votre machine cible, vous devez choisir une version ROS prise en charge (ou inversement). Bien que l'installation de ROS soit bien documentée dans le wiki ROS, il peut être difficile de les trouver. Voici donc un tableau de la version ROS, des plates-formes et de l'architecture cibles et des liens vers les guides d'installation appropriés:

Version ROS	Plate-forme	Cambre	Statut	Guide d'installation Lien
Cinétique	Ubuntu 16.04 (Xenial)	amd64 / i386 / armhf	Prise en charge	Kinetic-Xenial-guide
	Ubuntu 15.10 (Wily)	amd64 / i386	Prise en charge	Kinetic-Wily-guide
	Debian 8 (Jessie)	amd64 / arm64	Prise en charge	Kinetic-Jessie-guide
	OS X (Homebrew)	-	Expérimental	Kinetic-Homebrew-guide
	Gentoo	-	Expérimental	Kinetic-Gentoo-guide
	OpenEmbedded / Yocto	-	Expérimental	Kinetic-Yocto-guide

Travail en cours ...!

Bonjour éditeur mondial

Créer un espace de travail

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

Construisez votre espace de travail

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

Source votre fichier d'installation

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

Créer un nouveau paquet nommé hello_world avec quelques dépendances de base

```
catkin_create_pkg hello_world std_msgs rospy roscpp
```

Accédez à votre répertoire src et créez un nouveau fichier appelé talker.cpp

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

Editez votre nouveau fichier et collez ce code pour publier un message "hello world"

```
#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;
}
```

Revenir à la racine du répertoire de votre paquet

```
cd ..
```

Ajouter / décommenter ces lignes à votre 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)
```

Revenir à la racine de votre espace de travail

```
cd ..
```

Construisez votre nouvel éditeur

```
catkin_make
```

Source votre fichier d'installation à nouveau pour que vous ayez le nouveau package et éditeur

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

Démarrer ROS

```
roscore
```

Laissez roscore en cours d'exécution et dans un nouvel onglet / fenêtre de terminal, démarrez votre éditeur

```
rosrun hello_world talker
```

Laisser l'éditeur en cours d'exécution et dans un autre nouvel onglet / fenêtre de terminal, faire écho à la sortie

```
rostopic echo /chatter
```

Lire Commencer avec ros en ligne: https://riptutorial.com/fr/ros/topic/7287/commencer-avec-ros

Chapitre 2: créer un espace de travail

Introduction

Ce tutoriel montre comment créer un espace de travail. Un espace de travail est un ensemble de répertoires dans lesquels se trouve un ensemble de codes ROS associés. On peut avoir plusieurs espaces de travail ROS, mais il est possible de ne travailler que dans un seul à la fois.

Examples

Création d'un espace de travail

Pour créer un espace de travail, il faut exécuter les opérations suivantes dans le terminal:

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

Les commandes précédentes créent un espace de travail nommé `workspace_name` . Une fois qu'un espace de travail a été créé, il est important de le rechercher pour pouvoir l'utiliser:

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

Lire créer un espace de travail en ligne: https://riptutorial.com/fr/ros/topic/8313/creer-un-espace-de-travail

Chapitre 3: Créer un package

Introduction

Ce tutoriel montre comment créer un package dans ROS. Les paquets sont placés dans les espaces de travail, dans le répertoire src. Chaque répertoire de package doit avoir un `CMakeLists.txt` et un fichier `package.xml` .

Examples

Créer un paquet en utilisant rospy

En supposant qu'un espace de travail nommé `workspace_name` a déjà été créé dans le répertoire de base, un package nommé `package_name` peut être créé en exécutant les lignes de commande suivantes.

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

Lire Créer un package en ligne: https://riptutorial.com/fr/ros/topic/8314/creer-un-package

Chapitre 4: roslaunch

Remarques

'noeud forme le paquet' doit être 'noeud du paquet'

initialement, vous dites "démarrer" et "arrêter", mais vous n'expliquez pas comment faire un arrêt contrôlé.

Examples

lancer les nœuds ros et charger les paramètres depuis le fichier Yaml

roslaunch est un outil important qui gère le démarrage et l'arrêt des nœuds ROS. Il faut un ou plusieurs fichiers "* .launch" comme arguments.

Pour cet exemple, je ferai référence à ce qui suit (comme demandé dans cette question), alors comment pouvons-nous exécuter ces commandes consécutivement et automatiquement:

```
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
```

Tout d'abord , divisons `break up these commands in pieces` . Comme on peut le voir, 4 commandes ros sont nécessaires: *roscd* , *rosparam* , *rosrun* et *roslaunch* . Maintenant commençons!

Ceci est un exemple de la raison pour laquelle Roslaunch est puissant. En fait, toutes ces commandes pourraient être incluses dans `one and only roslaunch file in ROS` . il suffit ensuite de lancer ce fichier "solution.launch" pour un appel consécutif et automatique de ces commandes.

Pour commencer, les fichiers de lancement sont basés sur le formatage XML, voici un fichier de lancement de base dans ROS, nous l'appellerons "basic_example.launch" et il est inclus dans un package ROS nommé "roslaunch_example":

```
<launch>

</launch>
```

la commande pour exécuter ce fichier de lancement est

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

suivant les spécifications:

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

En raison de notre fichier de lancement n'inclut aucune commande, rien ne sera exécuté, plus sur cela plus tard ...

Y compris les nœuds dans les fichiers de lancement:
tout nœud ROS dans tout package ROS installé est appelable dans les fichiers de lancement. à cela nous devons spécifier le paquet contenant le noeud et son nom comme spécifié dans le paquet. par exemple :

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

`stereo_camera` est un nœud du paquet `stereo_camera` et les arguments spécifiés sont `__name:=bumblebeeLeft` `__name:=bumblebeeCenter` et `__name:=bumblebeeCenter` .

Pour ajouter ces nœuds, il faut ajouter les lignes suivantes:

```
<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>
```

En exécutant ce fichier de lancement, nous aurons les deux nœuds en cours d'exécution.

ajouter un paramètre à un noeud ROS:

Comme on peut le voir, nous avons ajouté un paramètre "name" pour les nœuds comme:

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

En fait, nous pouvons ajouter autant de paramètres que nous voulons (comme créé ci-dessus), puis nous y référer en appelant `"$(arg parameter_name)"` au lieu de fixer sa valeur.

Spécification de la sortie:

La balise de sortie peut être définie sur "screen" si vous devez voir le journal du noeud `on the terminal` ou "log" pour enregistrer le journal dans les fichiers journaux `(~/.ros)` .

Y compris les autres fichiers de lancement ROS dans un fichier de lancement ROS:

Comme indiqué ici , la balise vous permet d'importer un autre fichier XML roslaunch dans le fichier en cours. Il sera importé dans la portée actuelle de votre document, y compris les balises. Tout le contenu du fichier d'inclusion sera importé, à l'exception de la balise: la balise est uniquement respectée dans le fichier de niveau supérieur.

Donc, tout pour exécuter ces commandes

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

à partir d'un fichier de lancement, il suffit d'ajouter les lignes suivantes:

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

> roscd à un paquet dans le fichier de lancement ROS

Pour trouver le fichier de lancement que nous voulons inclure, nous n'avons pas besoin de spécifier le chemin complet. Au lieu de cela, roslaunch fournit la directive `"$(find package_name)"`, de cette façon, nous pouvons nous référer à notre fichier de lancement `relative to the package racine`.
Dans l'exemple ci-dessus, j'ai supposé que le fichier "kinect_center.launch" se trouvait dans le dossier "openni_launch_marvin) / launch /".

Chargement des paramètres depuis le fichier YAML:

Afin de charger les paramètres d'un fichier YAML dans ROS, ROS fournit la balise "rosparam". Comme indiqué dans le wiki : "La balise permet d'utiliser les fichiers rosparam YAML pour le chargement et le déchargement des paramètres du serveur de paramètres ROS. Elle peut également être utilisée pour supprimer des paramètres. La balise peut être placée à l'intérieur Le paramètre est traité comme un nom privé. "

En utilisant cette balise, nous pouvons charger notre fichier YAML dans le fichier de lancement en ajoutant cette ligne:

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

Comme utilisé ci-dessus, je suppose que le fichier YAML "marvin_cameras.yaml" se trouve dans le dossier "marvin_cameras / config /".

Assembler toutes les pièces

Maintenant que nous avons créé séparément le contenu de notre fichier de lancement, assemblons-les dans un gros fichier de lancement "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>
```

Maintenant, nous avons notre seul et unique fichier roslaunch pour exécuter toutes les commandes consécutivement et automatiquement.

Lire roslaunch en ligne: https://riptutorial.com/fr/ros/topic/7361/roslaunch

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec ros	ckirksey3, Community, Photon, Vtik
2	créer un espace de travail	lmiguelvargasf
3	Créer un package	lmiguelvargasf, Michael
4	roslaunch	ensonic, Vtik