

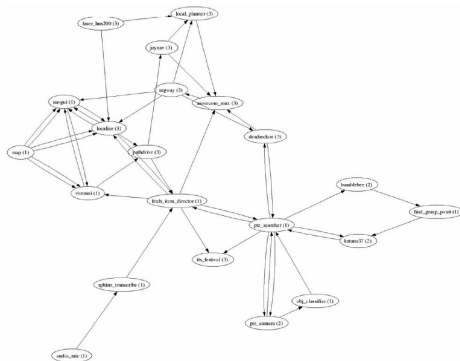
# Conceptos básicos de Ros

Clase 4  
Ing. Alexander López

# Problemas en robótica

## ¡Buscar un objeto!

## Graph ROS



## Objetivos del diseño de ROS

- La aplicación de la tarea se puede descomponer en muchos subsistemas independientes, como la navegación, la visión por ordenador, el agarre, etc.
- Estos subsistemas se pueden utilizar para otras tareas, como patrullas de seguridad, limpieza, entrega de correo, etc.
- Con hardware apropiado y geometría de capas de abstracción, la gran mayoría de aplicaciones de software puede ejecutarse en cualquier robot.

## Gráficos en ROS



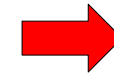
Comunicación entre nodos a través de mensajes (messages).

Con estos elementos se pueden implementar programas de Inteligencia artificial

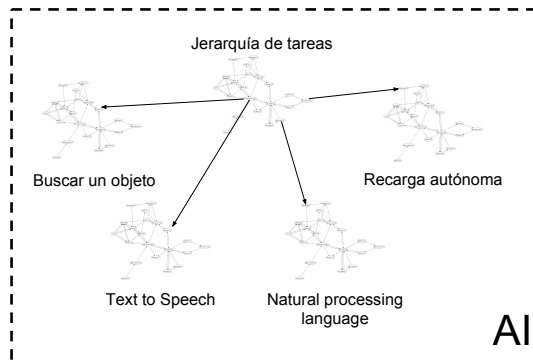
## Inteligencia Artificial

Estudio de “agentes inteligentes”: cualquier dispositivo que perciba su entorno y tome acciones que maximicen sus posibilidades de éxito en algún objetivo.

~~NO~~



## Subgraph



## Roscore

- Roscore es un servicio que proporciona información de conexión a los nodos para que puedan transmitir mensajes entre sí.
- Cada nodo se conecta a roscore al inicio para registrar los detalles de los flujos de mensajes que publica y los flujos (stream) a los que desea suscribirse.
- Cuando aparece un nuevo nodo, roscore le proporciona la información que necesita para una conexión directa peer-to-peer con otros nodos publicando y suscribiéndose a los mismos temas de mensaje.
- Cada sistema ROS necesita un roscore en funcionamiento, ya que sin él, los nodos no pueden encontrar otros nodos.

## Roscore

¿Por qué 11311?

Puerto 11311 fue elegido sin ninguna razón en especial alrededor del 2007.

Puede ser cualquiera de los puertos desde (1025-65535) en su lugar.

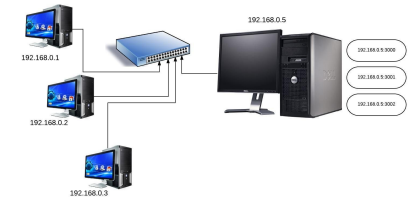
```
$ echo $ROS_MASTER_URI
```

## Roscore

Cuando un nodo ROS se inicia, se espera que su proceso tenga una variable de entorno (environment variable) denominada:

```
$ export ROS_MASTER_URI = http://hostname:11311/
```

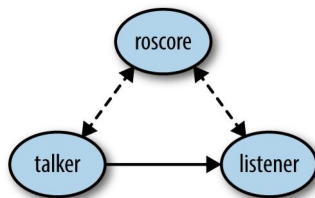
```
$ export ROS_IP=IP_COMPUTADORA_EJECUTA_ROSCORE
```



## Roscore

Ejemplo

- Los nodos del “talker” y del “listener” pueden hacer llamadas periódicamente a roscore mientras intercambian mensajes de peer-to-peer (de igual a igual) directamente ellos mismos.



## Roscore

- Roscore también proporciona un servidor de parámetros, que es utilizado extensivamente por nodos ROS para la configuración.
- El servidor de parámetros permite a los nodos almacenar y recuperar estructuras de datos arbitrarias, como descripciones de robots, parámetros para algoritmos, etc.
- Como con todo en ROS, hay una simple herramienta de línea de comandos para interactuar con el parámetro “rosparam”.

## Roscore

```
roscore http://alex-VirtualBox:11311/
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://alex-VirtualBox:35969/
ros_comm version 1.12.6

SUMMARY
=====

PARAMETERS
 * /roslstrobe: kinetic
 * /rosversion: 1.12.6

NODES

auto-starting new master
process[master]: started with pid [27862]
ROS_MASTER_URI=http://alex-VirtualBox:11311/

setting /run_id to 69e6ef8e-f6cf-11e6-9160-080027dc6b2b
process[roscout-1]: started with pid [27875]
started core service [/roscout]
```

## Catkin

¿Que es Catkin?

CMake es un sistema de compilación de código abierto comúnmente utilizado. Sin embargo, para el usuario más casual de catkin, todo lo que realmente necesita saber es que hay dos archivos, [CMakeLists.txt](#) y [package.xml](#), que necesita agregar alguna información específica para que todo funcione correctamente. Ya que vamos a compilar en python, no necesitamos entrar en tantos detalles.

## Workspace

¿Que es workspace?

- Antes de iniciar nuestro código, necesitamos instanciar nuestro Workspace.
- Este es solo un simple conjunto de directorios donde reside ciertos elementos y código de ROS.
- Se puede tener muchos Workspace, pero solo uno se puede ejecutar a la vez.

## Workspace

¿Que contiene un workspace?

Hay que asegurarnos que se ha añadido el script el cual ejecuta el contenido del archivo añadido, la configuración de todo ROS.

```
$ source /opt/ros/kinetic/setup.bash
```

```
alex@alex-VirtualBox:~$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

```
.bashrc
Save
alias alert='notify-send --urgency=low -i "${FILE}" "Alert: ${FILE}"'
alias error='cat /dev/null >& echo error'
alias info='cat /dev/null >& echo info'
alias success='cat /dev/null >& echo success'
alias warning='cat /dev/null >& echo warning'
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
if [ -f ~/.bash_aliases ] then
    . ~/.bash_aliases
fi
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -o posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -o posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
source /opt/ros/kinetic/setup.bash
```

# Workspace

¿Que contiene un workspace?

Creamos un Catkin (compilador que se usa en ROS) Workspace y lo inicializamos:

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

Ps.: Debemos ubicarnos en la raíz del sistema.

```
alex@alex-VirtualBox: ~/catkin_ws/src
alex@alex-VirtualBox:~$ mkdir -p ~/catkin_ws/src
alex@alex-VirtualBox:~$ ls
catkin_ws  Desktop  Downloads  Music  Pictures  Templates
alex@alex-VirtualBox:~$ cd catkin_ws/src/
alex@alex-VirtualBox:~/catkin_ws/src$ catkin_init_workspace
Creating symlink "/home/alex/catkin_ws/src/CMakeLists.txt" pointing to "/opt/ros/kinetic/share/catkin/cmake/toplevel.cmake"
alex@alex-VirtualBox:~/catkin_ws/src$
```



# Tecla TAB



Nos ayuda a autocompletar una función o hasta darnos la lista de opciones de posibles opciones que disponemos

```
alex@alex-VirtualBox:~/catkin_ws/src
alex@alex-VirtualBox:~$ mkdir -p ~/catkin_ws/src
alex@alex-VirtualBox:~$ ls
catkin_ws  Desktop  Downloads  Music  Pictures  Templates
alex@alex-VirtualBox:~$ cd catkin_ws/src/
alex@alex-VirtualBox:~/catkin_ws/src$ catkin_in
```

```
alex@alex-VirtualBox:~/catkin_ws/src
alex@alex-VirtualBox:~$ mkdir -p ~/catkin_ws/src
alex@alex-VirtualBox:~$ ls
catkin_ws  Desktop  Downloads  Music  Pictures  Templates
alex@alex-VirtualBox:~$ cd catkin_ws/src/
alex@alex-VirtualBox:~/catkin_ws/src$ catkin_init_workspace
```

# Workspace

¿Que contiene un workspace?

El comando `catkin_init_workspace` crea un archivo `CMakeLists.txt` en el archivo `src`, este crea dichos archivos:

```
CMakeLists.txt [read-only] (-:catkin_ws/src) - gedit
Open  Save
# top level CMakeLists.txt for a catkin workspace
# catkin/cmake/toplevel.cmake

cmake_minimum_required(VERSION 2.8.3)
set(CATKIN_TOPLEVEL TRUE)

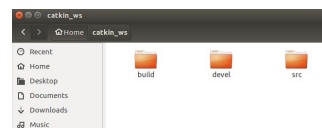
# search for catkin within the workspace
set(catkin_find_path "${CMAKE_SOURCE_DIR}")
execute_process(COMMAND ${CMAKE_COMMAND} -E
  RESULT_VARIABLE _res
  ERROR_VARIABLE _err
  OUTPUT_STRIP_TRAILING_WHITESPACE
  ERROR_STRIP_TRAILING_WHITESPACE
)
if((NOT _res EQUAL 0 AND NOT _res EQUAL 2)
  # searching for catkin resulted in an error
  string(REPLACE "-" "" _catkin_src "${_err}")
  message(FATAL_ERROR "search for 'catkin' in workspace failed (${_catkin_src})")
endif()
```

# Workspace

¿Que contiene un workspace?

Ahora se necesita crear los directorios `build` (Aquí guardan los resultados como las librerías y ejecutables que se crean cuando se trabaja en C++.) y `devel` (Contiene los archivos de instalación de cada workspace.):

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



```
alex@alex-VirtualBox:~/catkin_ws
- Using empy: /usr/bin/empy
- Using CATKIN_ENABLE_TESTING: ON
- Call enable_testing()
- Using CATKIN_TEST_RESULTS_DIR: /home/alex/catkin_ws/build/test_results
- Looking for pthread.h - found
- Looking for pthread_create - not found
- Looking for pthread_create in pthreads - not found
- Looking for pthread_create in pthread - found
- Found Threads: TRUE
- Found gtest sources under '/usr/src/gtest': gtests will be built
- Using Python nosetests: /usr/bin/nosetests-2.7
- CATKIN 0.7.4
- BUILD_SHARED_LIBS is on
- Configuring done
- Generating done
- Build files have been written to: /home/alex/catkin_ws/build
-- Running command: "make -j1 -l1" in "/home/alex/catkin_ws/build"
alex@alex-VirtualBox:~/catkin_ws$
```

## Workspace

¿Que contiene un workspace?

Se debe inicializar las configuraciones de nuestro workspace en el bash.

```
$ cd ~/catkin_ws  
$ source devel/setup.bash
```

Se debe ubicarse en la raíz del workspace

Ps.: Añadir al bashrc para no escribir esto siempre que se abre un terminal.

```
elif [ -f /etc/bash_completion ]; then  
    . /etc/bash_completion  
fi  
  
source /opt/ros/kinetic/setup.bash  
source /home/alex/catkin_ws/devel/setup.bash
```

## Workspace

¿Qué fuentes tenemos?

El archivo setup.bash incluye la dirección de los paquetes dentro del “environment variable”.

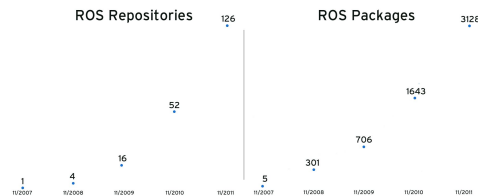
Este comando nos indica las direcciones de los “workspace”, que dentro de ellos existen varios paquetes los cuales ya han sido incluidos.

```
$ echo $ROS_PACKAGE_PATH
```

```
alex@alex-VirtualBox:~$ echo $ROS_PACKAGE_PATH  
/home/alex/catkin_ws/src:/opt/ros/kinetic/share  
alex@alex-VirtualBox:~$
```

## ROS packages

- El software ROS está organizado en paquetes, cada uno de los cuales contiene una combinación de código, datos y documentación.
- El ecosistema de ROS incluye miles de paquetes disponibles públicamente en repositorios abiertos.
- El objetivo de estos paquetes es proporcionar esta útil funcionalidad de una manera fácil de usar para que el software pueda ser reutilizado.



## ROS packages

- Crear un nuevo paquete es fácil:

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg new_package rospy
```

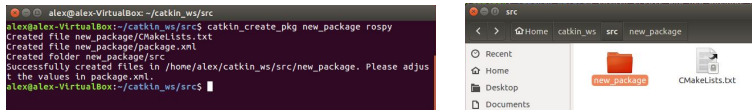
Los paquetes se ubican dentro del directorio src

Nombre del nuevo paquete

Paquete del cual depende

## ROS packages

Cada directorio de cada paquete (package) debe incluir un archivo CMakeLists.txt y un package.xml.



## ROS packages

¿Qué contiene package.xml?

```
<?xml version="1.0"?>
<package>
  <name>new_package</name>
  <version>0.0.0</version>
  <description> This package is for just teaching. </description>
  <maintainer email="user@todo.todo">user</maintainer>
  <license>TODO</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <run_depend>rospy</run_depend>
</package>
```

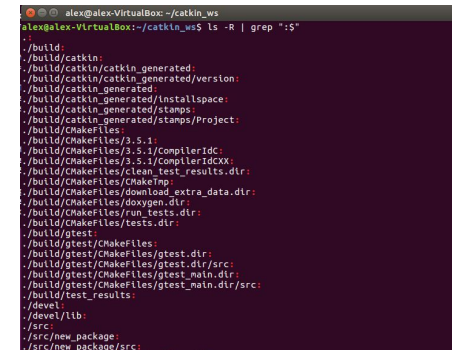
Incluir una licencia como BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3

Paquetes de los cuales depende nuestro "new\_package"

## ROS packages

Una vez creado el paquete, ahora se puede colocar nodos hechos en Python dentro del directorio "src" de nuestro paquete.

## ROS packages



## Rosbash

El paquete rosbash contiene algunas funciones bash útiles y agrega la terminación de tabulación a un gran número de las utilidades básicas de ros.

- Roscd: Cambia el directorio por la ubicación exacta de donde se encuentra el paquete.
- Rospd: Muestra la dirección donde se ubica el paquete.
- Rosls: Lista los archivos que se encuentran actualmente en el paquete.
- Rosed: Permite editar un archivo del paquete.
- Roscp: Copia un archivo de un paquete a la dirección actual.
- Rosrun: Ejecute nodos de un paquete de ros.

## Roscd

Cambia el directorio por la ubicación exacta de donde se encuentra el paquete.

```
alex@alex-VirtualBox: /opt/ros/kinetic/share/rospy_tutorials
alex@alex-VirtualBox:~$ roscd new_package
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ cd
alex@alex-VirtualBox:~$ roscd rospy
rospy/
alex@alex-VirtualBox:~$ roscd rospy_tutorials/
alex@alex-VirtualBox:/opt/ros/kinetic/share/rospy_tutorials$
```

## Rospd

Muestra la dirección donde se ubica el paquete.

```
alex@alex-VirtualBox: /opt/ros/kinetic/share/rospy_tutorials
alex@alex-VirtualBox:~$ rospd rospy_tutorials/
0 /opt/ros/kinetic/share/rospy_tutorials
1 -
2 -
alex@alex-VirtualBox:/opt/ros/kinetic/share/rospy_tutorials$
```

## Rosls

Lista los archivos que se encuentran actualmente en el paquete.

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox:~$ rosls new_package/
CMakeLists.txt package.xml src
alex@alex-VirtualBox:~$
```



# Rosed

Permite editar un archivo del paquete.

```
alex@alex-VirtualBox: ~  
alex@alex-VirtualBox:~$ rosd rosps_tutorials talker
```

```

* @alexg@VirtualBox:~$
jusr@fritz:~/python$
Software License Agreement (SBL License)
Copyright (c) 2008, Willow Garage, Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the
  distribution.
* The names of Willow Garage, Inc. nor the names of its
  contributors may be used to endorse or promote products derived
  from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

```

# Roscp

Copia un archivo de un paquete a la dirección actual.

```
alex@alex-VirtualBox:~$ rscp rospy_tutorials talker.py .
alex@alex-VirtualBox:~$ ls
catkin_ws  Documents  examples.desktop  Pictures  talker.py  Videos
Desktop    Downloads  Music            Public   Templates
```

# Rosrun

- Los paquetes solo son locaciones de nuestros archivos de sistema, los nodos son los programas ejecutables.
- Ejemplo:

```
$ roscore
$ rosrun rospy_tutorials talker
```

Diagram illustrating the command execution:

- \$ roscore**: Iniciamos el Master
- \$ rosrun rospy\_tutorials talker**: Nombre del nodo
- rospy\_tutorials**: Nombre del paquete que contiene el nodo
- \$ rosrun**: un nuevo terminal se ejecuta lo siguiente

# Rosrun

- El programa “talker” reside en el paquete llamado “rospy\_tutorials” el cual se encuentra en /opt/ros/kinetic/share/rospy\_tutorials.
- Utilizando “roslaunch” se puede ejecutar el programa desde cualquier carpeta ubicada en la cual esta ubicada terminal.



## Rosrun

Envía un mensaje:  
"hello world"

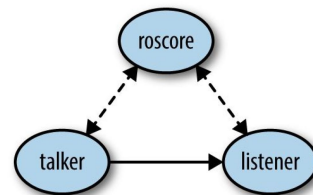
Este se repite cada  
10 segundos

El mensaje puede  
ser escuchado por  
cualquier nodo

```
alex@alex-VirtualBox:~$ rosrun rospy_tutorials talker
[INFO] [1487541205.386226]: hello world 1487541205.31
[INFO] [1487541205.406610]: hello world 1487541205.41
[INFO] [1487541205.506791]: hello world 1487541205.51
[INFO] [1487541205.607240]: hello world 1487541205.61
^Calex@alex-VirtualBox:~$ python
Python 2.7.12 (default, Nov 19 2016, 08:48:10)
[CC 5.4.0 20160909] on linux2
Type "help", "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.time()
1487541221.001217
>>>
```

## Rosrun

- El sistema más pequeño necesita de al menos 2 nodos, donde uno envía mensajes al otro.
- Se inicializará el nodo "listener" para recibir los mensajes provenientes del nodo "talker"



## Rosrun

\$ rosrun rospy\_tutorials listener

El mensaje  
escuchado por este  
nodo en específico

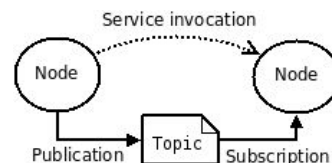
```
alex@alex-VirtualBox:~$ rosrun rospy_tutorials listener
[INFO] [1487544267.924204]: /Listener_31043_1487544267682I heard hello world 148
7544267.92
[INFO] [1487544268.023857]: /Listener_31043_1487544267682I heard hello world 148
7544268.02
[INFO] [1487544268.123487]: /Listener_31043_1487544267682I heard hello world 148
7544268.12
^Calex@alex-VirtualBox:~$
```

Este se repite cada  
10 segundos

Recibe el mensaje:  
"hello world"

## Rosrun

- Los nodos pueden publicar o suscribirse a un tópico.
- Podemos usar el comando rostopic para ver la lista de tópicos publicados, en este caso un tópico por nodo.



\$ rostopic list

```
alex@alex-VirtualBox:~$ rostopic list
/chatter
/rosout
/rosout_agg
alex@alex-VirtualBox:~$ rosnode list
/listener_31085_1487544301826
/rosout
/talker_30934_1487544243060
alex@alex-VirtualBox:~$
```

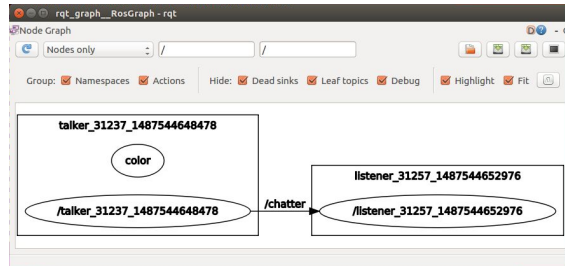
\$ rosnode list

```
alex@alex-VirtualBox:~$ rostopic list
alex@alex-VirtualBox:~$ rostopic list
/rosout
/rosout_agg
alex@alex-VirtualBox:~$ rosnode list
/rosout
/talker_30934_1487544243060
alex@alex-VirtualBox:~$
```

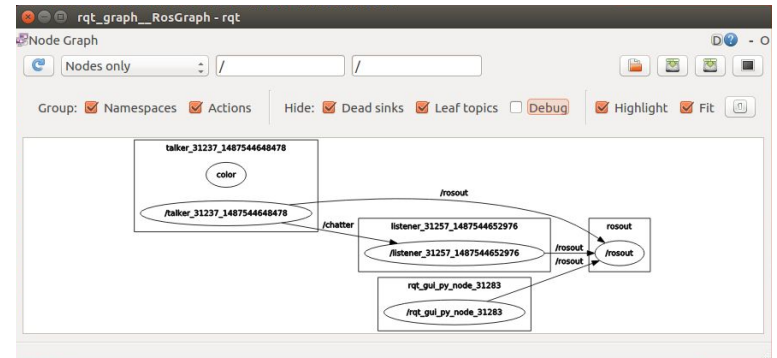
# Rosrun

Ahora podemos observar nuestro propio “Graph” a partir de los nodos que hemos conectado con el comando:

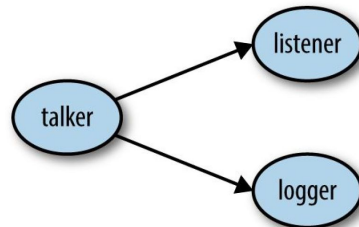
`$ rqt_graph`



# Rosrun



# Rosrun

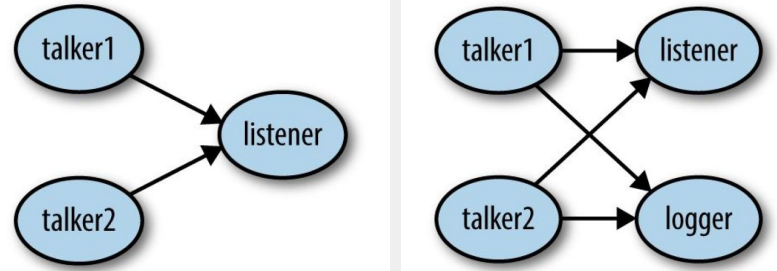


Ahora se puede demostrar algunos de los beneficios de la arquitectura de pasar mensajes (message-passing).

Se puede depurar guardando la data proveniente de “talker” dentro de la computadora.

# Rosrun

Otras opciones con las que se puede trabajar:



## Network Rosrun

Con este comando agregamos la nueva dirección del Robot Master por la dirección ip en nuestra computadora en nuestra red.

```
$ export ROS_MASTER_URI=http://ip:11311
```

```
alex@alex-VirtualBox: ~  
alex@alex-VirtualBox:~$ echo $ROS_MASTER_URI  
http://localhost:11311  
alex@alex-VirtualBox:~$
```

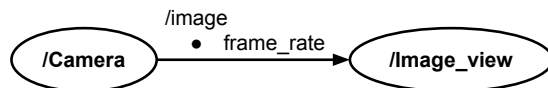
## Network Rosrun

```
alex@alex-VirtualBox:~$ ifconfig  
alex@alex-VirtualBox:~$ ifconfig  
enp0s3  
Link encap:Ethernet HWaddr 08:00:27:dc:6b:2b  
inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0  
inet6 addr: fe80::b2b3:d70a:b753:2c32/64 Scope:Link  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:525365 errors:0 dropped:0 overruns:0 frame:0  
TX packets:130425 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:605940229 (605.9 MB) TX bytes:8862271 (8.8 MB)  
  
lo  
Link encap:Local Loopback  
inet addr:127.0.0.1 Mask:255.0.0.0  
inet6 addr: ::1/128 Scope:Host  
UP LOOPBACK RUNNING MTU:65536 Metric:1  
RX packets:19201 errors:0 dropped:0 overruns:0 frame:0  
TX packets:19201 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1  
RX bytes:2389626 (2.3 MB) TX bytes:2389626 (2.3 MB)  
  
alex@alex-VirtualBox:~$  
alex@alex-VirtualBox:~$ export ROS_MASTER_URI=http://10.0.2.15:11311  
alex@alex-VirtualBox:~$ echo $ROS_MASTER_URI  
http://10.0.2.15:11311  
alex@alex-VirtualBox:~$
```

## Names

Los **nombres** son un concepto fundamental en ROS. Los nodos, los flujos de mensajes (a menudo denominados "topics") y los parámetros deben tener nombres únicos.

Ejemplo:



## Namespaces

¿que pasa cuando  
tenemos dos cámaras?

- Las colisiones entre estos son extremadamente comunes en sistemas robóticos.
- Estos usan la convención de Unix y direcciones de Internet (URLs).
- Roslaunch usa esta herramienta para evitar colisiones entre paquetes.

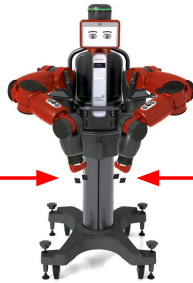
Ejemplo:

```
/home/user1/readme.txt  
/home/user2/readme.txt
```

Archivos con nombres  
iguales pero direcciones  
(paths) diferentes.

## Namespaces

¿que pasa cuando  
tenemos dos cámaras?



¿Esto soluciona el  
problema?

No con los tópicos

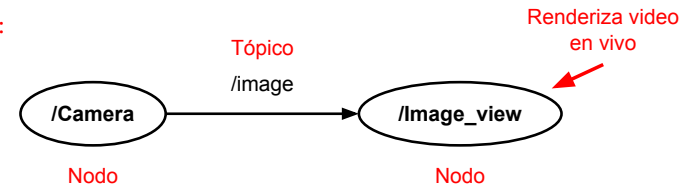
right/image → ← left/image

## Remapping

¿que pasa cuando  
tenemos dos cámaras?

- En ROS, cualquier “String” en un programa que define a un nombre puede ser remapeado en tiempo de ejecución.
- Podemos remapear el nombre de cualquiera de las cámaras de Baxter, sin tener que cambiar el nodo “Image\_view”.

Ejemplo:



## Remapping

- Permite la posibilidad de reusar el código para diversas funciones, no cambiar los programas existentes, solo las distintas variables de entrada.
- ROS provee una sintaxis estándar para remapear nombres cuando van a usarse nodos en la línea de comandos.



Ejemplos:

`$ ./image_view image:=right/image`

## Remapping

Ejemplo real:

Cámaras de Baxter:

`/cameras/head_camera/image`

`/cameras/right_hand_camera/image`

`/cameras/left_hand_camera/image`

Namespaces

`$ rosrn image_view image_view image:=/cameras/right_hand_camera/image`

↑  
Package

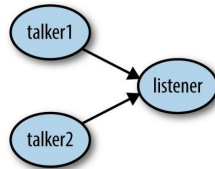
↑  
Node

## Remapping

- Los nombres en ROS deben ser únicos, ya que muchos de ellos son los nombres de elementos específicos.
- Si un nodo es iniciado 2 veces, “roscore” sale del nodo más antiguo para inicializar el nodo más reciente.
- También se puede remapear los nombres de los nodos, por ejemplo:

```
$ ./talker __name:=talker1  
$ ./talker __name:=talker2
```

↑  
Se usa doble  
guión bajo



## Roslaunch

Comando diseñado para iniciar un grupo de nodos, es parecido a roscore. Ejemplo:

```
$ roslaunch rospy_tutorials talker_listener
```

↑  
Paquete

↑  
Archivo tipo \*.launch

## Roslaunch

```
<launch>  
  <node name="talker" pkg="rospy_tutorials"  
    type="talker.py" output="screen" />  
  <node name="listener" pkg="rospy_tutorials"  
    type="listener.py" output="screen" />  
</launch>
```

↑  
Archivo del cual  
proviene el nodo

↑  
Indica que cada nodo  
debe mostrarse en  
respectivo terminal

## Roslaunch

- También puede iniciar programas en otras computadoras dentro de una misma red usando “ssh”.
- Si se cierra el proceso iniciado por “roslaunch”, se cierran todos los nodos que este inicializo.

Inicializa automaticamente el roscore?

**¡Gracias!**

¡La única pregunta tonta es la que no se  
hace!