ROS
Robot Operating System

Sistemi RealTime

Prof. Davide Brugali

Università degli Studi di Bergamo

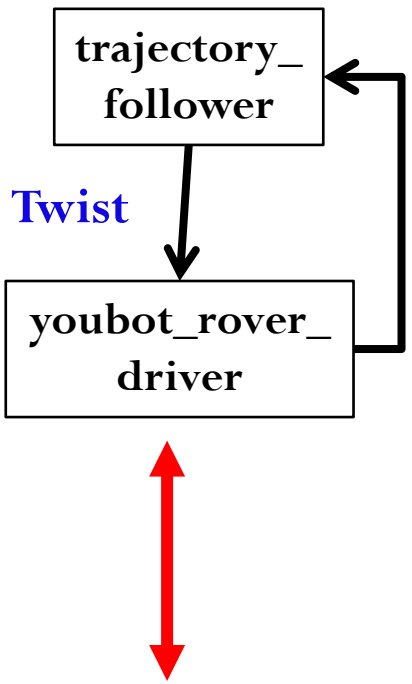
Installazione di ROS su VirtualBox

- Scaricare e installare VirtualBox 4.2.18 + Extension Pack

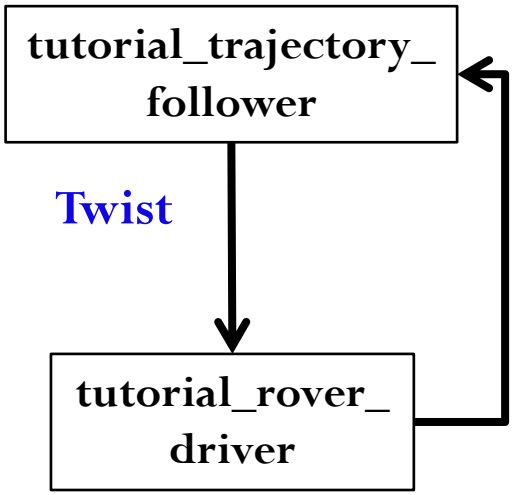
https://www.virtualbox.org/

- Creare una macchina virtuale da un file esistente
  - da menu di Virtual Box selezionare "New“
    - Nome : Ubuntu-12.04.03
    - Tipo : Linux
    - Versione: Ubuntu
  - Impostare RAM : 1024
  - Use an existing virtual hard drive file
    - Selezionare il file Ubuntu 12-04-03.vdi
- Avviare la macchina virtuale : pulsante Avvia
- Login Ubuntu
  - User : serl
  - Pwd : unibg

Sviluppo di un sistema composto da due nodi

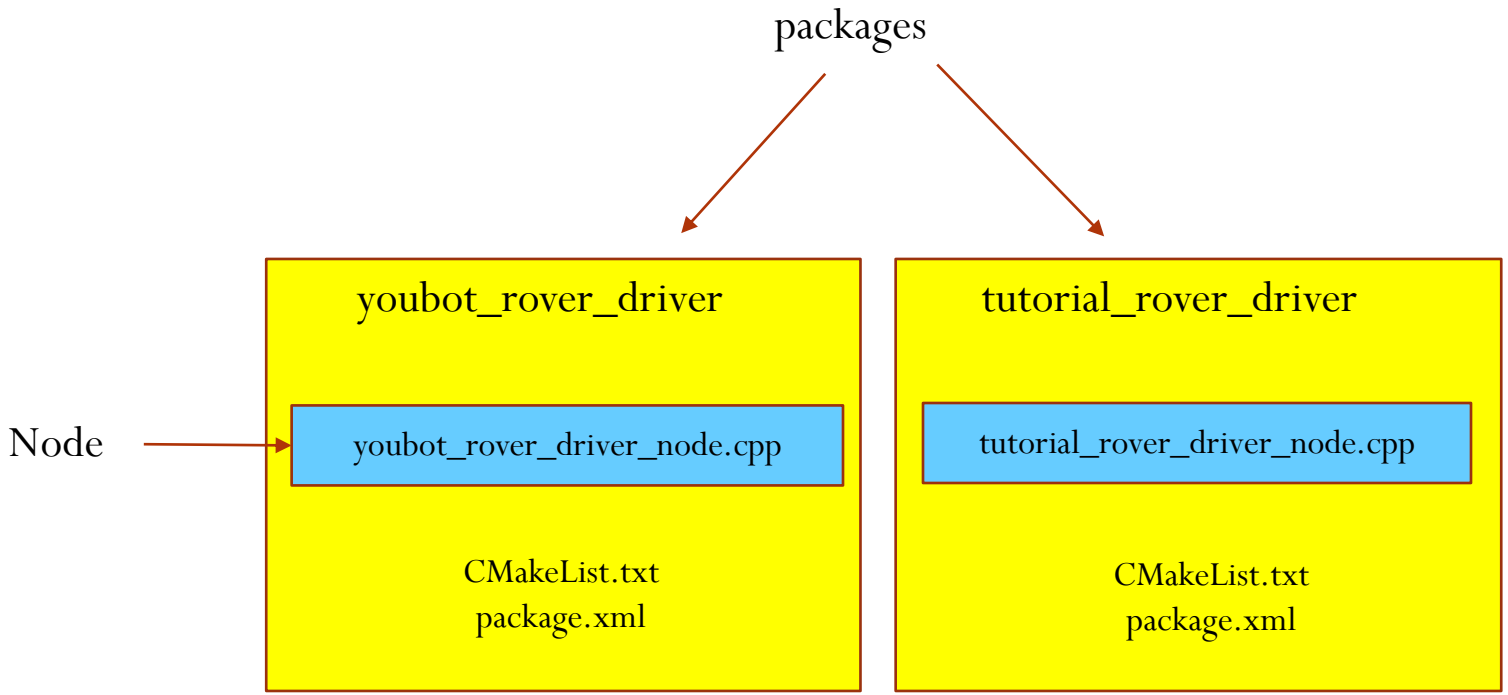


Odometry



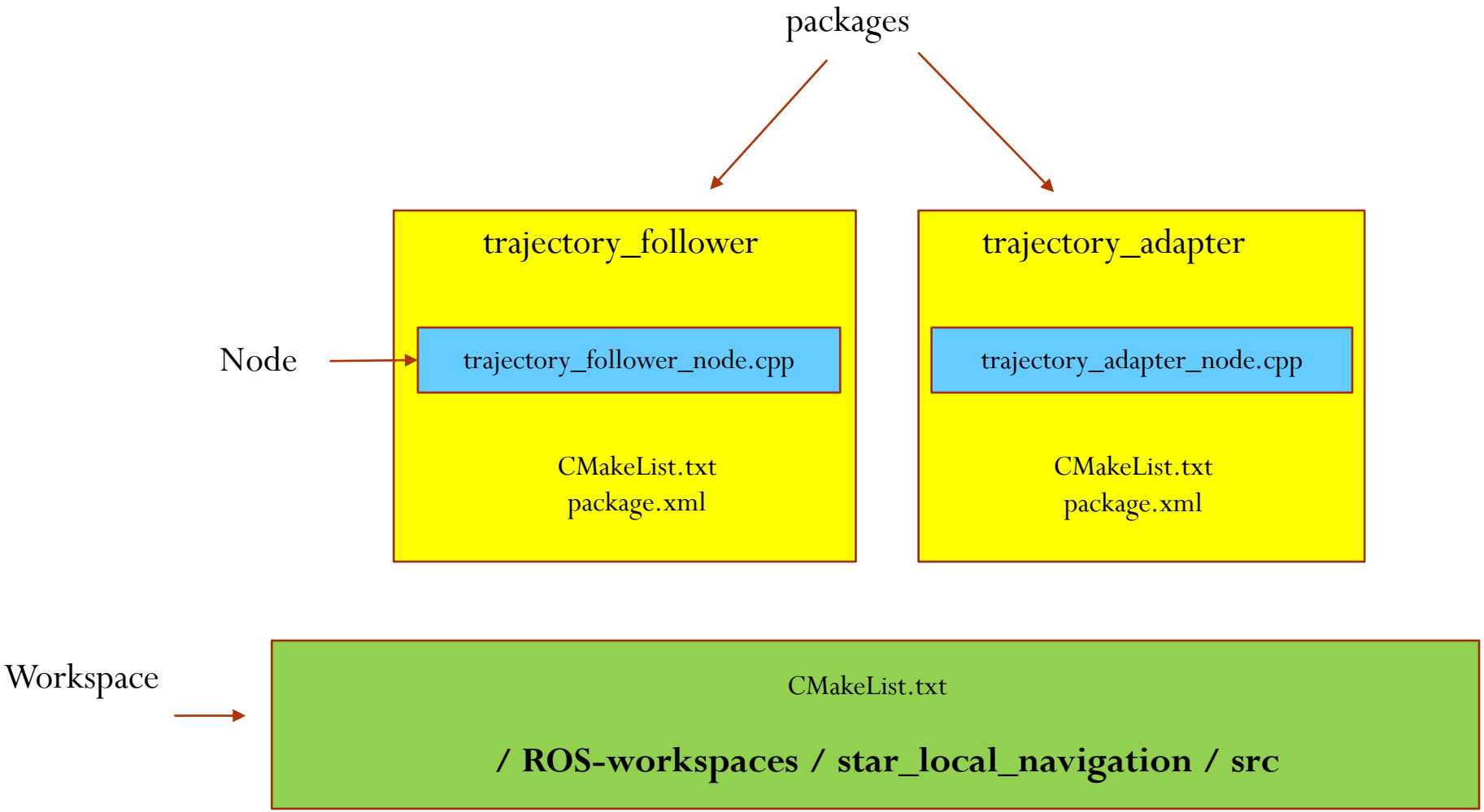
Odometry

Workspace di sviluppo per ROS



http://wiki.ros.org/catkin/Tutorials/create_a_workspace

Workspace di sviluppo per ROS



Workspace

```
$ mkdir ~ / ROS-workspaces / star_robot_drivers / src

$ cd ~ / ROS-workspaces / star_robot_drivers / src

$ catkin_init_workspace

$ cd ~ / ROS-workspaces / star_robot_drivers

$ catkin_make

$ source devel / setup.bash
```

```
star_robot_drivers /          -- WORKSPACE
    src /                     -- SOURCE SPACE
        CMakeLists.txt        -- The 'toplevel' CMake file
    build /                   -- BUILD SPACE CATKIN_IGNORE
    devel /                   -- DEVELOPMENT SPACE
```

ROS Packages

- Software in ROS is organized in *packages*. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.
- The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused.
- This means that a package is the smallest individual thing you can build in ROS and it is the way software is bundled for release (meaning, for example, there is one debian package for each ROS package), respectively.
- http://wiki.ros.org/Packages

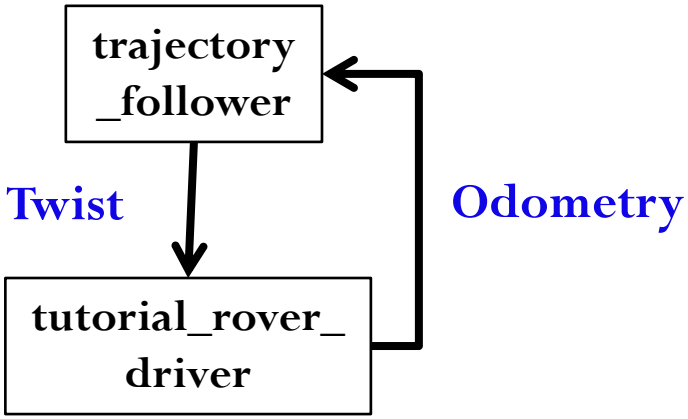
Package tutorial_rover_driver

```
$ cd ~/ROS-workspaces/star_robot_drivers/src

$ catkin_create_pkg tutorial_rover_driver std_msgs nav_msgs
geometry_msgs rospy roscpp

$ source ~/ROS-workspaces/star_robot_drivers/devel/setup.bash
```

star_robot_drivers /
src /
CMakeLists.txt

tutorial_rover_driver/ -- package
CMakeLists.txt
package.xml

ROS Messages



- **geometry_msgs / Twist.msg**
- **nav_msgs / Odometry.msg**

POSE

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of postion and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

autogenerated on Wed, 24 Sep 2014 21:16:45

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Wed, 24 Sep 2014 21:16:45

Odometry

nav_msgs/Odometry Message

File: `nav_msgs/Odometry.msg`

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Compact Message Definition

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

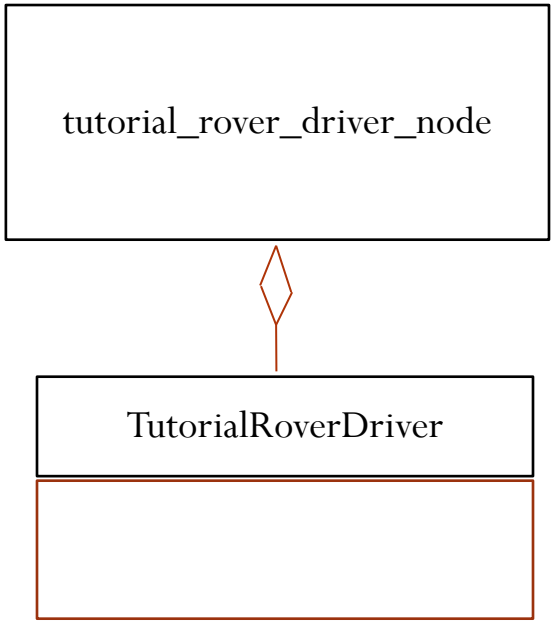
ROS Nodes

- A *node* is a process that performs computation. Nodes are combined together into a graph and communicate with one another using
  - streaming topics,
  - RPC services,
  - and the Parameter Server.
- These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes.
- For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provide a graphical view of the system, and so on.

ROS Nodes

- All running nodes have a **graph resource name** that uniquely identifies them to the rest of the system.
  - For example, /hokuyo_node could be the name of a Hokuyo driver broadcasting laser scans.
- Nodes also have a node type, that simplifies the process of referring to a node executable on the fileystem.
- These node types are **package resource name** with the name of the node's package and the name of the node executable file.
- In order to resolve a node type, ROS searches for all executables in the package with the specified name and chooses the first that it finds.
- As such, you need to be careful and not produce different executables with the same name in the same package .

Nodo tutorial_rover_driver



main program

class

N.B. comando per spostarsi direttamente nella cartella base di un package

```
$ roscd tutorial_rover_driver/

$ gedit include/TutorialRoverSimulator.hpp

$ gedit src/TutorialRoverSimulator.cpp
```

tutorial_rover_driver_node.hpp

```
#ifndef TUTORIAL_ROVER_DRIVER_NODE
#define TUTORIAL_ROVER_DRIVER_NODE

#include "ros/ros.h"
#include "TutorialRoverDriver.hpp"

// Define callbacks
void twistCallback(const geometry_msgs::Twist::ConstPtr& twist);

// Published Topics
ros::Publisher odometryPub;


// Rover Driver
TutorialRoverDriver *roverDriver;

#endif
```

tutorial_rover_driver_node.cpp

```
#include "ros/ros.h"
#include "tutorial_rover_driver_node.hpp"

void twistCallback(const geometry_msgs::Twist::ConstPtr& newTwist) {
        geometry_msgs::Twist twist = *newTwist;

        ROS_INFO("Received TWIST VX:%f   VY:%f   RZ:%f",
                twist.linear.x, twist.linear.y, twist.angular.z);

        // set the new Twist
        roverDriver->setTwist(twist);
}


int main(int argc, char **argv) {
        ros::init(argc, argv, "TutorialRoverDriver");

        int frequency = 5; // [Hz]

                . . .
```

tutorial_rover_driver_node.cpp

```
int main(int argc, char **argv) {
        ros::init(argc, argv, "TutorialRoverDriver");

        int frequency = 5; // [Hz]

        // TutorialRoverDriver initialized with the execution frequency
        roverDriver = new TutorialRoverDriver(frequency);

        // Define the node handler
        ros::NodeHandle nodeHandler;

        // Subscribed topics
        ros::Subscriber twistSub = nodeHandler.subscribe("/twist", 1000, twistCallback);

        // Published Topics
        odometryPub = nodeHandler.advertise<nav_msgs::Odometry>("/odometry", 10);

        ROS_INFO("TutorialRoverDriver : Started");

        // Periodic execution frequency
        ros::Rate loop_rate(frequency);

        // Odometry message
        nav_msgs::Odometry odometry;

        while(ros::ok()) {
                // get the new Odometry
                roverDriver->getOdometry(&odometry);

                ROS_INFO("Publish ODOMETRY   PX:%f   PY:%f   RZ:%f",
                        odometry.pose.pose.position.x,
                        odometry.pose.pose.position.y,
                        odometry.pose.pose.orientation.z);
                odometryPub.publish(odometry);

                ros::spinOnce();
                loop_rate.sleep();
        }
        free(roverDriver);
        return 0;
}
```

CMakeLists.txt

```
## Declare a cpp executable
add_executable(tutorial_rover_driver
        src/TutorialRoverDriver.cpp
        src/tutorial_rover_driver_node.cpp)
```

nome del file eseguibile

Creare il progetto Eclipse per un package

```
$ catkin_make --force-cmake -G"Eclipse CDT4 - Unix
Makefiles"

$ cd ~/ROS-workspaces/star_robot_drivers/build

$ cmake ../src -DCMAKE_BUILD_TYPE=Debug

$ cd ~/ROS-workspaces/star_robot_drivers/

$ mkdir eclipse     // Eclipse workspace folder

$ source ROS-workspaces/star_robot_drivers/devel/setup.bash

$ eclipse              // avvia Eclipse da terminal
```

- In Eclipse : File / Import / General /Existing Project
- Selezionare la cartella
  ROS-workspaces/star_robot_drivers/build
- Selezionare il progetto elencato

Compilare ed eseguire nodi ROS

```
// avvio del roscore in una finestra dedicata
$ roscore
```

```
// compilazione dei packages nel workspace
$ cd ~/ROS-workspaces/star_robot_drivers/

$ catkin_make

// avvio del nodo tutorial_robot_driver
$ cd ~/ROS-workspaces/star_robot_drivers/devel/
lib/tutorial_rover_driver

$ rosrun tutorial_rover_driver tutorial_rover_driver
```

nome del package

nome dell'eseguibile