ROS -Robot Operating System

10/2021

ROS Workspace, Package e compilazione

WpWeb S.r.l. via Livorno 60 - 10144 Torino (Environment Park) | Tel. +39. 011 225.7570 | Fax: 011.2257.7573 | info@wpweb.com
P.IVA e Cod. Fisc. 07292240012 | Reg. Impr. Torino, n° 07292240012 | REA 883122 | Cap. Sociale € 10.920,00 i.v.

Panoramica

- ROS
- ROS nodes e package
- ROS catkin_tools
- ROS C++ (roscpp)
- ROS Python (rospy)
- ROS subscribers e publishers

ROS Master 1/2

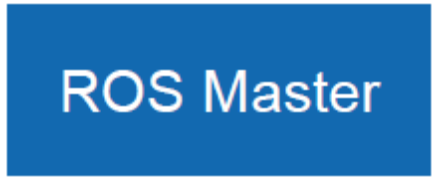
- Gestisce la comunicazione tra i nodi (processi)
- Ogni nodo all'avvio viene registrato sul master

Il nodo master viene lanciato con il comando

```
> roscore
```

- ROS_MASTER_URI: indirizzo della macchina su cui gira il master
- ROS_HOSTNAME: indirizzo della macchina su cui gira un nodo non master nella stessa rete

ROS Master
2/2



ROS Nodes 1/2

- I nodi ROS sono dei programmi eseguibile a scopo singolo
- Vengono compilati singolarmente, eseguiti e gestititi
- I singoli nodi sono raccolti all'interno di pacchetti

Un nodo diverso dal master viene lanciato con il comando

```
> rosrun nome_pacchetto nome_nodo
```

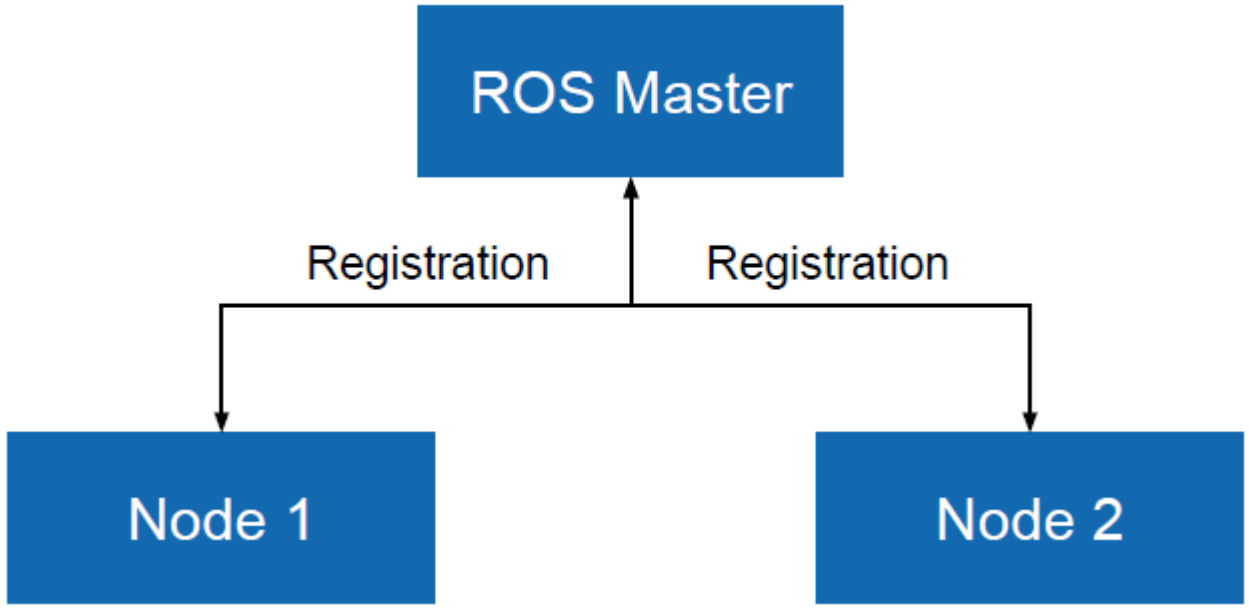
Per vedere i nodi attivi al momento

```
> rosnode list
```

Per ottenere informazioni su un nodo

```
> rosnode info nome_nodo
```

ROS Nodes 2/2



ROS Topics 1/2

- I nodi ROS comunicano attraverso singoli stream di messaggi detti topic
  - Un nodo può pubblicare (publish) o sottoscrivere (subscribe) un topic
  - Tipicamente si ha 1 publisher e n subscriber

Per visualizzare la lista dei topic attualmente attivi

```
> rostopic list
```

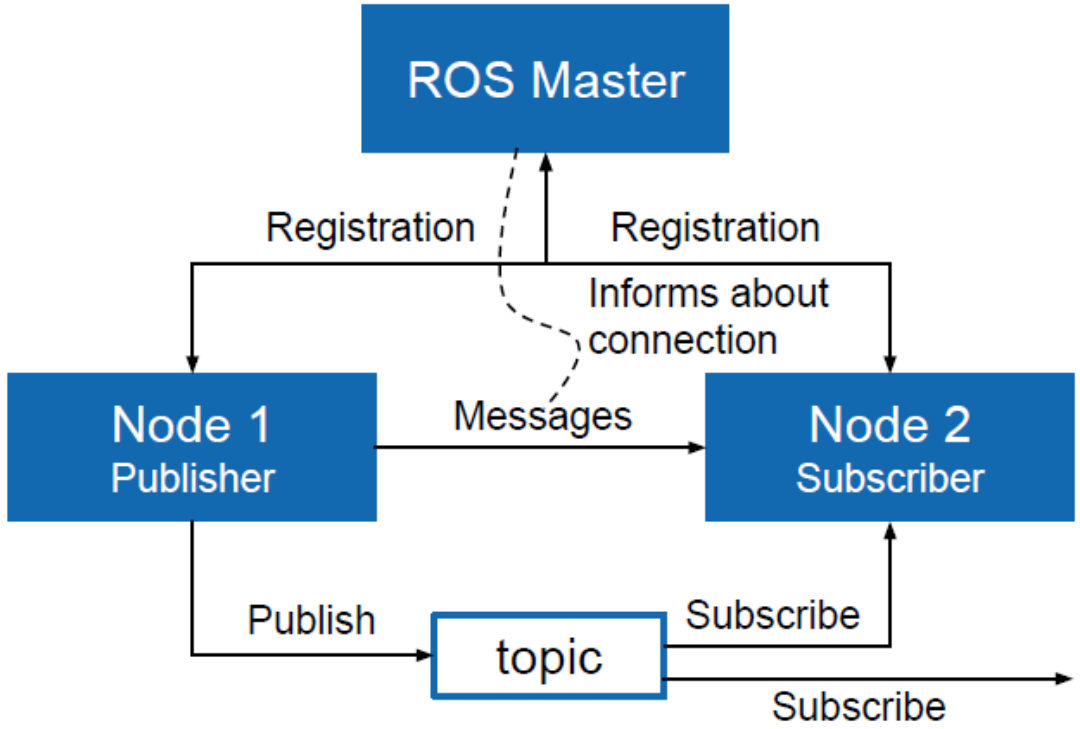
Per vedere i nodi attivi al momento

```
> rostopic echo /topic
```

Per ottenere informazioni su un nodo

```
> rostopic info /topic
```

ROS Topics 2/2



ROS Messages 1/3

- I messaggi sono strutture di dati che definiscono il tipo di topic
- Sono formati da una struttura nidificata di diversi tipi di variabili:
  - Interi
  - Float
  - Booleani
  - Stringhe
  - Array di oggetti
  - Ecc…
- Sono definiti all'interno di file *.msg

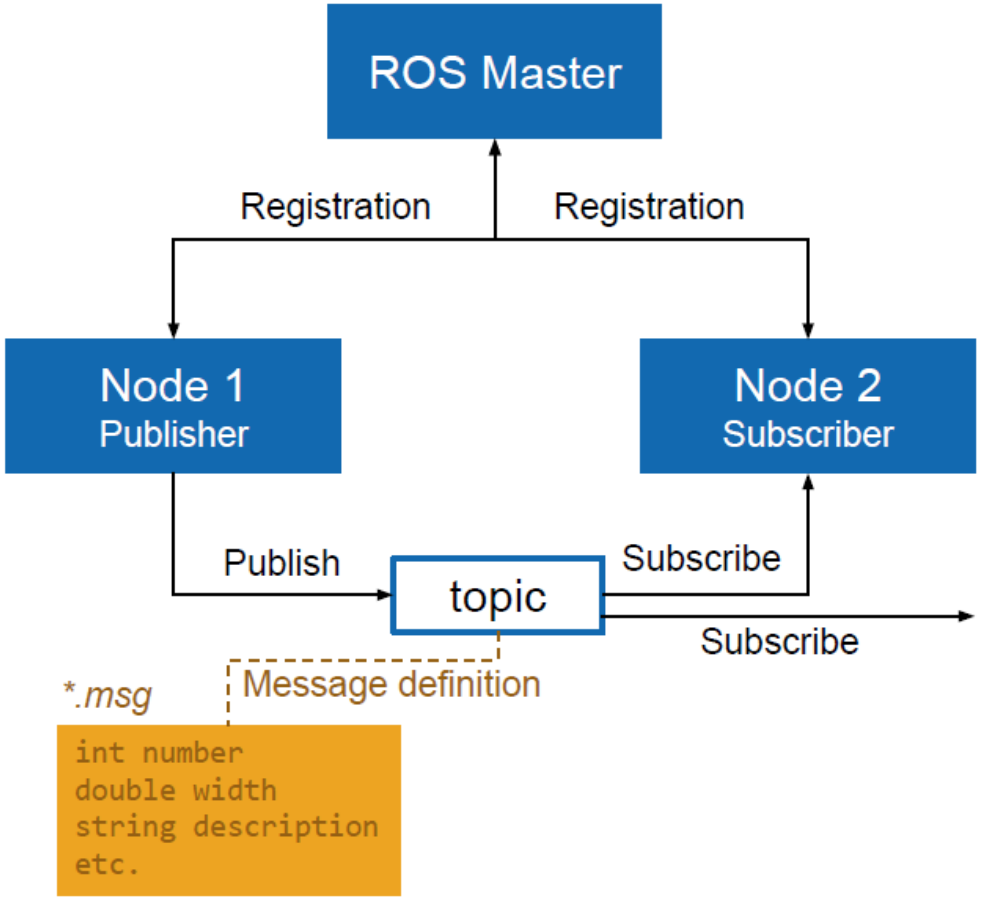
Per vedere il tipo di messaggio definito nel topic

```
> rostopic type /topic
```

Per pubblicare un messaggio a un topic

```
> rostopic pub /topic type dato_da_inviare
```

ROS Messages 2/3



ROS Messages 3/3

POSE STAMPED EXAMPLE

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
geometry_msgs/Pose pose
    geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
    geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
```

geometry_msgs/PoseStamped.msg

ROS Service

- I Service sono strutture di dati che permettono di instaurare una comunicazione di tipo client/server tra nodi, quindi richiesta/risposta
- Un Service è definito da una coppia di messaggi:
  - Request →inviata da un nodo client
  - Responce →inviata da un nodo server
- Sono definiti all'interno di file *.srv

catkin build System

Un catkin workspace contiene i seguenti spazi

Lavorare qui

Il source space contiene il nostro codice. Qui è dove vengono creati, clonati e modificati i pacchetti del progetto che stiamo realizzando.

Non toccare

Il build space è dove il CMake viene chiamato per compilare i pacchetti del source space.

Non toccare

Il development space è dove gli elementi compilati vengono generati, da qui è possibile installarli.

ROS Workspace

- Come prima cosa bisogna definire lo spazio di lavoro in cui andremo a costruire il nostro sistema.
  Di default il workspace è settato in

```
> source /opt/ros/melodic/setup.bash
```

- Per comodità normalmente la cartella di workspace si crea all'interno della home di linux

```
> mkdir –p ~/catkin_ws/src
> cd ~/catkin_ws/src
> catkin_init_workspace
> cd ..
> catkin build
```

- Per sovrascrivere e verificare il workspace appena creato

```
> source devel/setup.bash
> echo $ROS_PACKAGE_PATH
```

ROS C++

roscpp é un implementazione C++ di ROS. Si tratta di una libreria che permette l'interfacciamento con i topic, i service e i parameters.

Gli elementi essenziali che compongono questa libreria sono:

- L' inizializzatine e lo spinning
- La gestione dei nodi (node hanler)
- Logging
- Subscriber e Publisher
- Parameters
- Services
- Actions
- Times

ROS Python

rospy é una pura libreria Python per ROS e molti tool sono basati su di essa.

A differenza di roscpp è ottimizzata a favore del programmatore. Gli algoritmi possono essere prototipati e testati velocemente.

Gli elementi essenziali che compongono questa libreria sono i medesimi di roscpp:

- L' inizializzatine e lo spinning
- La gestione dei nodi (node hanler)
- Logging
- Subscriber e Publisher
- Parameters
- Services
- Actions
- Times

ROS Packages 1/5

Un programma ROS è organizzato in packages contenenti:

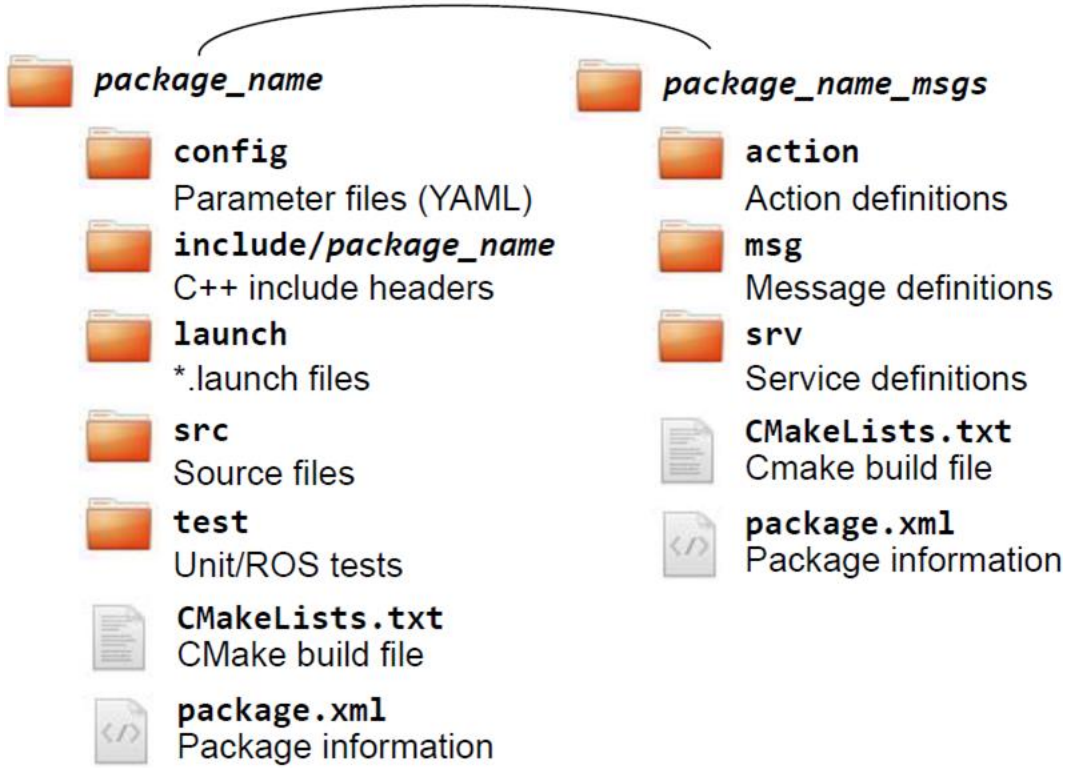
- Source code
- Launch files
- Configuration files
- Librerie
- Dati
- Documentazione varia

Un package che necessita di altri pacchetti durante la compilazione (ad esempio messaggi) deve dichiararle al momento della creazione

```
> catkin_create_pkg package_name dipendenza1 dipendenza2 ... dipendenzaN
```

ROS Packages 2/5

È buona norma separare le definizioni dei messaggi custom dagli altri package!



ROS Packages 3/5 package.xml

Il manifesto di un pacchetto è un file .xml, chiamato package.xml ed è univoco per ogni pacchetto.

Questo file definisce le proprietà del package:

- Nome
- Versione
- Autore
- Dipendenze da altri pacchetti :
  - <buildtool_depend>
  - <build_depend>
  - <run_depend>
  - <test_depend>

ROS Packages 4/5 package.xml

- **<buildtool_depend>**
  Specifica il sistema di compilazione necessario al pacchetto. In genere serve solo catkin.
- **<build_depend>**
  Specifica i pacchetti necessari alla compilazione del pacchetto.
- **<run_depend>**
  Specifica i pacchetti necessary ad eseguire il codice nel pacchetto.
- **<test_depend>**
  Specifica solo dipendenze aggiuntive per test di verifica. Normalmente é omesso perché bastano quelle inserite per la compilazione e l'esecuzione.

ROS Packages 5/5 CMakeLists.txt

Questo file è l'input al sistema di compilazione Cmake:

1. Versione CMake richiesta (*cmake_minimum_required*)
2. Nome del package (*project()*)
3. Configurazione C++
4. Trovare altri pacchetti CMake/Catkin necessari alla compilazione (*find_package()*)
5. Message/Service/Action Generators (*add_message_files()*, *add_service_files()*, *add_action_files()*)
6. Richiede la generazione di message/service/action (*generate_messages()*)
7. Specifica le informazioni di compilazione per il package (*catkin_package()*)
8. Libraries/Executables da compilare(*add_library()/add_executable()/target_link_libraries()*)
9. Tests da compilare (*catkin_add_gtest()*)
10. Regole di installazione (*install()*)

Esercizio

- Creare il proprio workspace
- Creare il proprio package

```
> catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

- Creazione messaggi e servizi http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
- Publisher e Subscriber C++ http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29
  - Compilare ed analizzare
- Publisher e Subscriber Python http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29
  - Compilare ed analizzare
- Service e Client C++ http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29
  - Compilare ed analizzare
- Service e Client Python http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29
  - Compilare ed analizzare

Service esercizio

- Creazione messaggi e servizi http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

.../srv/AddTwoInts.srv

```
int64 a
int64 b
---
int64 sum
```

ROS Launch

- launch è un tool di ROS che permette di lanciare diversi nodi contemporaneamente e di definire all'occorrenza dei parametri
- launch funziona tramite file XML con estensione *.launch

```
<node pkg="nome package" name="nome nodo" type="tipo di nodo"
output="screen" />
```

- Anche se è sconsigliabile a livello pratico, launch avvia in automatico il comando roscore (se non è già stato lanciato in precedenza)

Esistono due modi per utilizzare un launch file:

1) Spostarsi nella cartella in cui è presente e lanciarlo

```
> roslaunch file.launch
```

2) Lanciare il file tramite il package all'interno del workspace

```
> roslaunch nome_package file.launch
```

Compilazione 1/8

La compilazione di un workspace ROS avviene tramite il pacchetto Python **catkin_tools**. Questo pacchetto non è altro che una collezione di macro e relativi codici utilizzato per compilare i pacchetti ROS.

È stato inizialmente introdotto nel 2015 come parte di ROS Fuerte, l' ultima versione stabile è stata rilasciata a luglio 2021 (0.7.1).

Attualmente è considerato ancora un pacchetto beta.

Compilazione 2/8

- La command line interface di catkin è il punto di partenza per la maggior parte delle funzionalità del pacchetto.

```
> catkin [opzioni] <verb> [argomenti e opzioni verb]
```

- Con verb si intendono i seguenti sottocomandi:
  - **build** – Compila i pacchetti in un catkin workspace
  - **config** – Configura il catkin workspace
  - **clean** – Ripulisce dalla compilazione il workspace
  - **create** – Crea un pacchetto catkin
  - **env** – Esegue un comando in un workspace modificato
  - **init** – Inizializza un catkin workspace
  - **list** – Trova e mostra le informazioni sui pacchetti nel workspace
  - **locate** – Restituisce il percorso dei pacchetti catkin
  - **profile** – Gestisce i profili catking

Compilazione 3/8 catkin build

Il verb build viene utilizzato per la compilazione di uno o più pacchetti in un workspace catkin.

Come la maggior parte dei verb, build è context-aware. Quindi può essere eseguito da qualsiasi directory contenuta da un workspace inizializzato.

L'utilizzo di base è finalizzato alla creazione del build e devel space.

```
> catkin build
```

- Il **build space** è il punto in cui viene invocato il CMake per la copilazione dei pacchetti presenti nel source space.
- Il **devel space** è dove vengono posizionati i target del build prima di essere installati.

Compilazione 4/8 catkin config

Il verb config viene utilizzato sia per visualizzare che per configurare le opzioni di un workspace come ad esempio i profili.

Normalmente durante lo sviluppo si utilizza il profilo di default che corrisponde a quello di debug.

Le opzioni più comunemente usate sono:

- -h
- --profile
- --init
- --whitelist e --no-whitelist
- --blacklist e --no-blacklist
- --install
- -j
- --cmake-args

Compilazione 5/8 catkin clean

Il verb clean permette di pulire in sicurezza il workspace dai rusultati di una compilazione.

```
> catkin clean
```

Oltre a rimuovere completamente il build, devel e instal space, permette anche di rimuovere anche solamente parte di questi space.

```
> catkin clean nome_package
```

```
$ catkin clean
[clean] Warning: This will completely remove the following directories. (Use `--yes` to skip this check)
[clean] Log Space:     /tmp/quickstart_ws/logs
[clean] Build Space:   /tmp/quickstart_ws/build
[clean] Devel Space:   /tmp/quickstart_ws/devel

[clean] Are you sure you want to completely remove the directories listed above? [yN]:
```

Compilazione 6/8 catkin init

Il verb init è la via più semplice per inizializzare un catkin workspace, in questo modo viene rilevato automaticamente da altri verb che necessitato il path del workspace.

Questo verb non salva nessuna informazione sulla configurazione del workspace, ma crea la directory nascosta .catkin_tools all'interno del workspace.

```
> catkin init
```

Se risulta che una cartella contiene già un workspace con il comando

```
> catkin config
```

è possibile determinare la root del workspace

```
$ catkin init
--------------------------------------------------------------
Profile:                     default
Extending:                   None
Workspace:                   /tmp/path/to/my_catkin_ws
--------------------------------------------------------------
Source Space:       [exists] /tmp/path/to/my_catkin_ws/src
Log Space:          [exists] /tmp/path/to/my_catkin_ws/logs
Build Space:        [exists] /tmp/path/to/my_catkin_ws/build
Devel Space:        [exists] /tmp/path/to/my_catkin_ws/devel
Install Space:      [unused] /tmp/path/to/my_catkin_ws/install
DESTDIR:            [unused] None
--------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------------------
...
--------------------------------------------------------------
Initialized new catkin workspace in `/tmp/path/to/my_catkin_ws`
--------------------------------------------------------------


--------------------------------------------------------------
WARNING: Your workspace is not extending any other result
space, but it is set to use a `linked` devel space layout.
This requires the `catkin` CMake package in your source space
in order to be built.
--------------------------------------------------------------
```

Compilazione 7/8 catkin list

Il verb list viene utilizzato per visualizzare le dipendenze di un pacchetto catkin presente nel workspace.

```
> catkin list --deps
```

Compilazione 8/8 catkin locate

Il verb locate permette di ottenere il path degli space presenti nel catkin workspace:

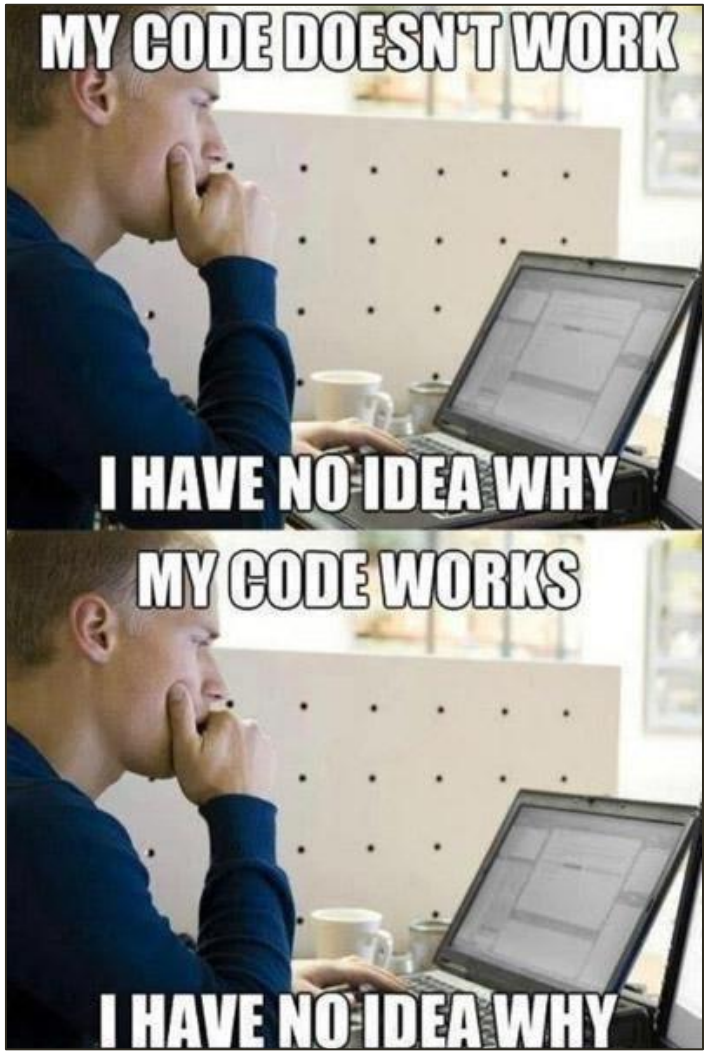
```
> catkin locate --nome_space
```

- --src/-s restituisce il path del source space
- --build/-b restituisce il path del build space
- --devel/-d restituisce il path del devel space
- --install/-i restituisce il path del install space

Esercizio

- Lanciare i programmi degli esercizi precedenti con due differenti launchfile

Domande?


MY CODE DOESN'T WORK
I HAVE NO IDEA WHY
MY CODE WORKS
I HAVE NO IDEA WHY

Link utili

ROS Wiki

- http://wiki.ros.org/

Catkin

- https://catkin-tools.readthedocs.io/en/latest/index.html#
- http://github.com/ros/catkin

Tutorials

- http://wiki.ros.org/ROS/Tutorials

Pacchetti disponibili

- http://www.ros.org/browse/

ROS Cheat Sheet

- https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
- https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index

Fine seconda parte