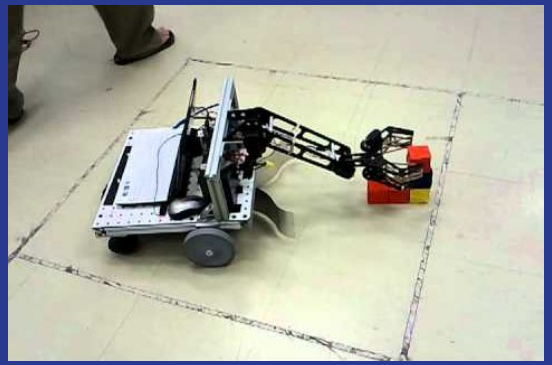

ROBOGO
A bot in the bag

Boston dynamics' robots celebrate the end of 2020 with a perfectly choreographed dance party

We will need an architecture which can support more complex hardware and software components.



Goal: Develop “big” software for robots

Challenges encountered in robotics

-The world is asynchronous

-Robots must manage significant complexity

-Robot hardware requires abstraction

Problems

Sequential Programming

What happens if an obstacle appears while you are going forward?

What happens to the encoder data while you are turning?

What if some other module wants the same data?

Solution: Callbacks

Complexity

Solution: Organising code

Separate processes: Cameras, Odometry, Laser Scanner, Map Building can all be separated out: they'll interact through an interface.

Interfaces: Software processes ("nodes" in ROS) communicate about shared "topics" in ROS

Publish/Subscribe: Have each module receive only the data (messages) it requests

Hardware dependent code

Solution:

Abstracting hardware

Summary

We want:

-Callbacks
-Separate processes that communicate through a messaging interface
-A messaging interface that helps avoid hardware dependencies

What is ROS?

A meta operating system

- Open source
- Runs in Linux (esp. Ubuntu)
- Ongoing Windows implementation

Message passing

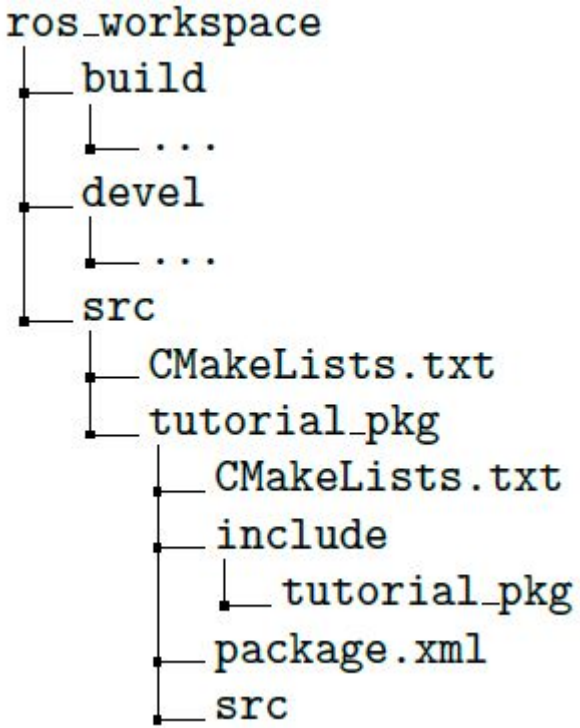
- Publish
- Subscribe
- Services via remote invocation

Support

- Supports numerous programming languages (C++, Python, Lisp, Java)
- Agent based (nodes)
- Exponential adoption
- Countless commercial, hobby, and academic robots use ROS (http://wiki.ros.org/Robots)

NAVIGATING THE ROS FILESYSTEM

NAVIGATING THE ROS FILESYSTEM



Packages: The ROS software is divided into packages that can contain various types of programs, images, data, and even tutorials. The specific contents depend on the application for the package. The site http://wiki.ros.org/Packages discusses ROS packages.

A package can contain programs written in Python or C++ to control a robot or another device. For the turtlesim simulator package for example, the package contains the executable code used to change the background color or move a turtle around on the screen. This package also contains images of a turtle for display and files used to create the simulator.

Creating & building a ros package

Creating - Using the *catkin_create_pkg* command

Syntax:

catkin_create_pkg <package_name> [depend1] [depend2]

Building - Using the *catkin_make* command

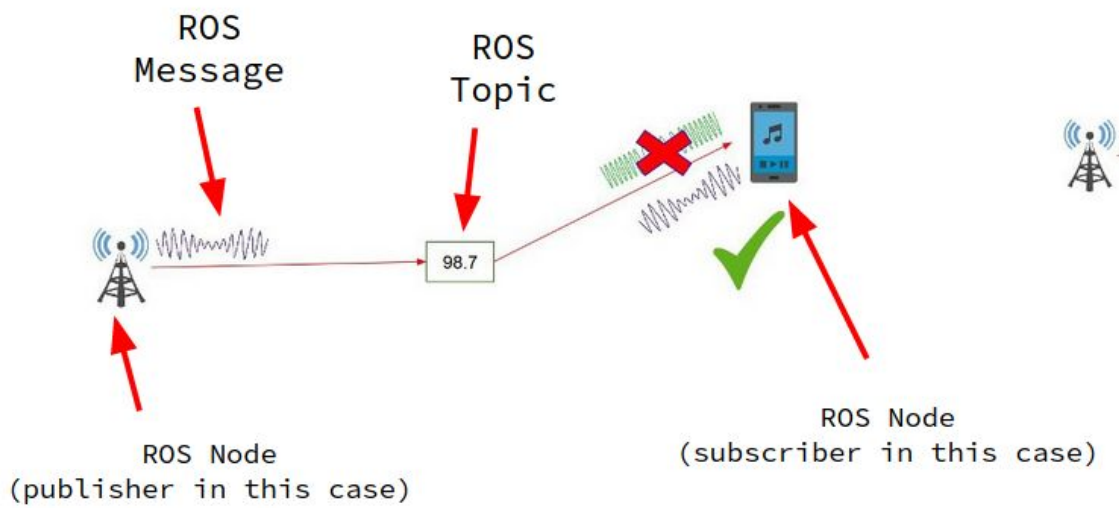
cd ~/catkin_ws && catkin_make - builds all packages in src directory

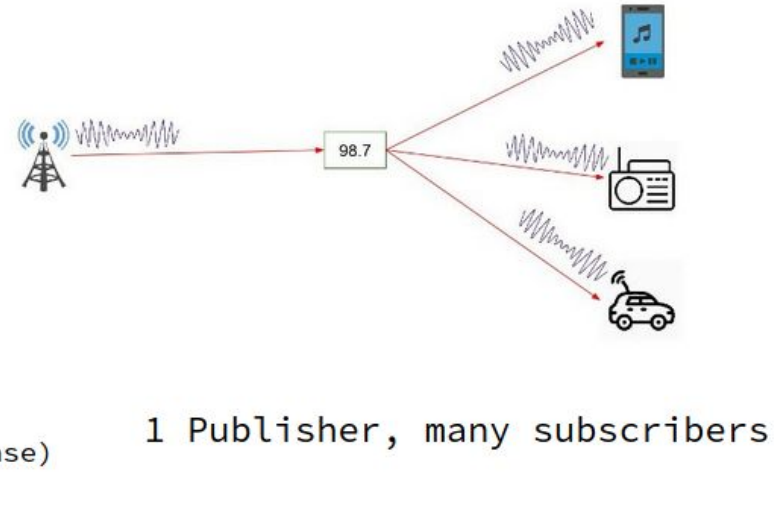
To build only a specific package

cd ~/catkin_ws

catkin_make -DCATKIN_WHITELIST_PACKAGES="<package-name>"

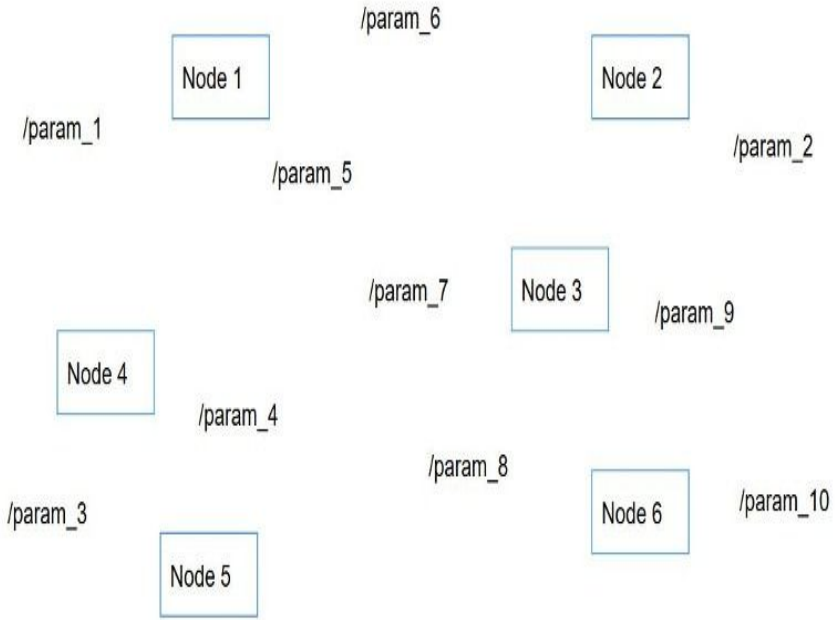
ROS NODES, TOPICS & MESSAGES



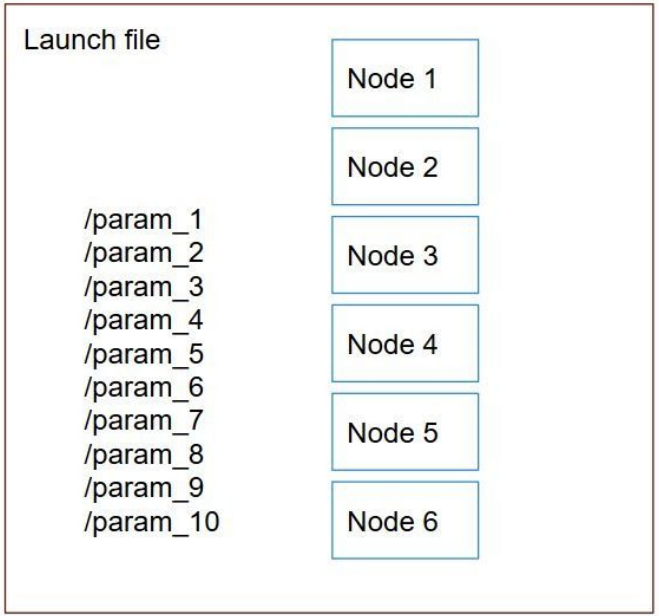


Roslaunch

Problem: Lots of messy terminals



The solution: use ROS launch files



Turtlesim – the first ROS robot simulation

A simple way to learn the basics of ROS is to use the turtlesim simulator that is part of the ROS installation. The simulation consists of a graphical window that shows a turtle-shaped robot. The background color for the turtle's world can be changed using the Parameter Server. The turtle can be moved around on the screen by ROS commands or using the keyboard.

https://subscription.packtpub.com/book/hardware_and_creative/9781788479592/1/ch01lvl1sec14/turtlesim-the-first-ros-robot-simulation

The `turtlesim_node` subscribes to the `/turtle1/cmd_vel` topic, so the turtle can be commanded to move by sending messages on this topic. First, determine the type of messages for the topic by typing:

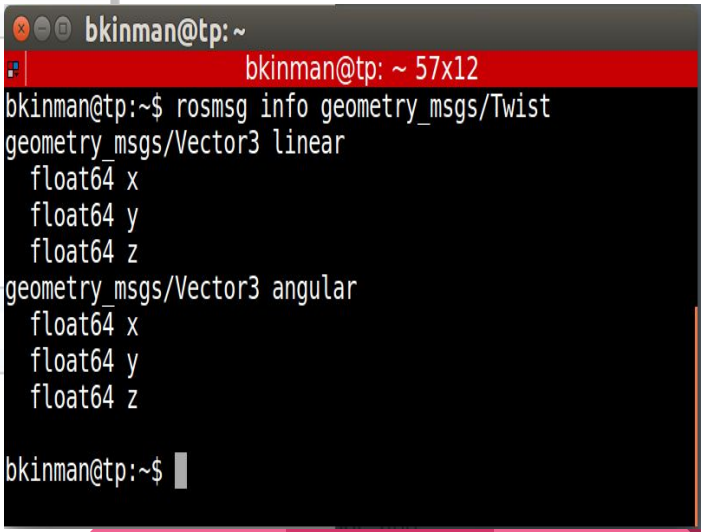
Copy

```
$ rostopic type /turtle1/cmd_vel
```

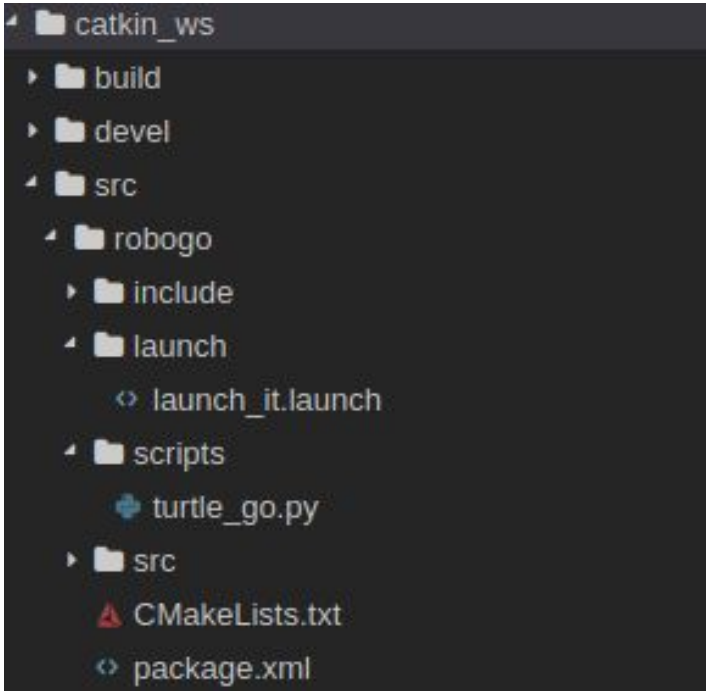
which displays the message type as the `Twist` message from the `geometry_msgs` package:

```
geometry_msgs/Twist
```

Next, the format of the message can be determined with the command:



SUMMARY



robogo: Package name

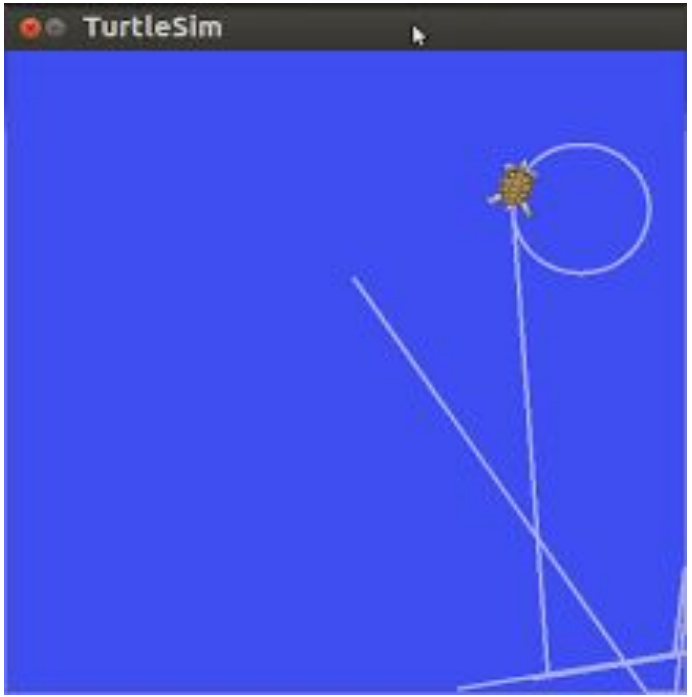
launch: Has a launch file “launch_it.launch”
Launches the two nodes

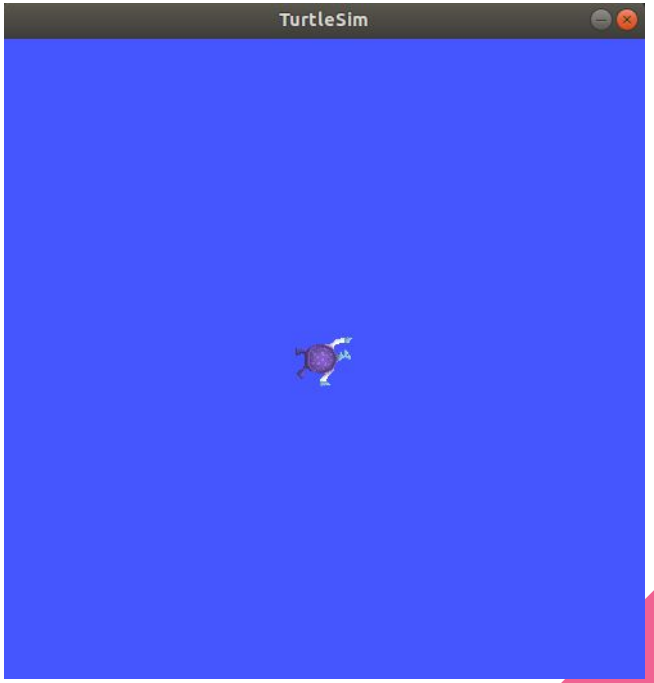
turtle_go.py : Publisher node where you’ll publish velocities according to which turtle in turtlesim will move

turtlesim_node : subscriber node which subscribes to the topic “/turtle1/cmd_vel”
It also publishes it’s pose (x,y,theta coordinates) on the topic “/turtle1/pose”

Implementation

Turtlesim window





http://wiki.ros.org/ROS/Tutorials/Understar es
http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

Using Turtlesim to trace paths

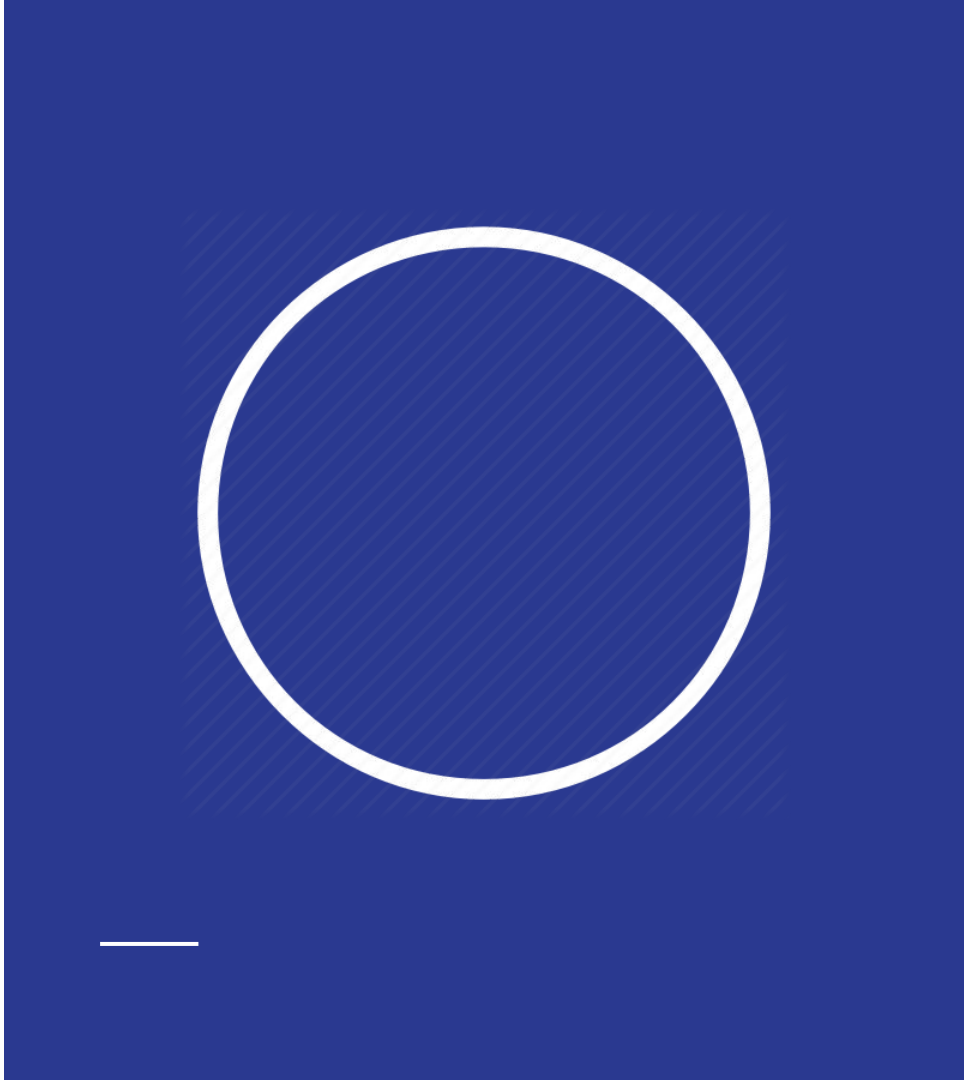
You can get the following information:

- x, y, theta coordinates of the turtle
- Time at any instant

You can change the following parameters:

- Linear velocities in x, y direction
- Angular velocity in z direction

1. Trace a Circle

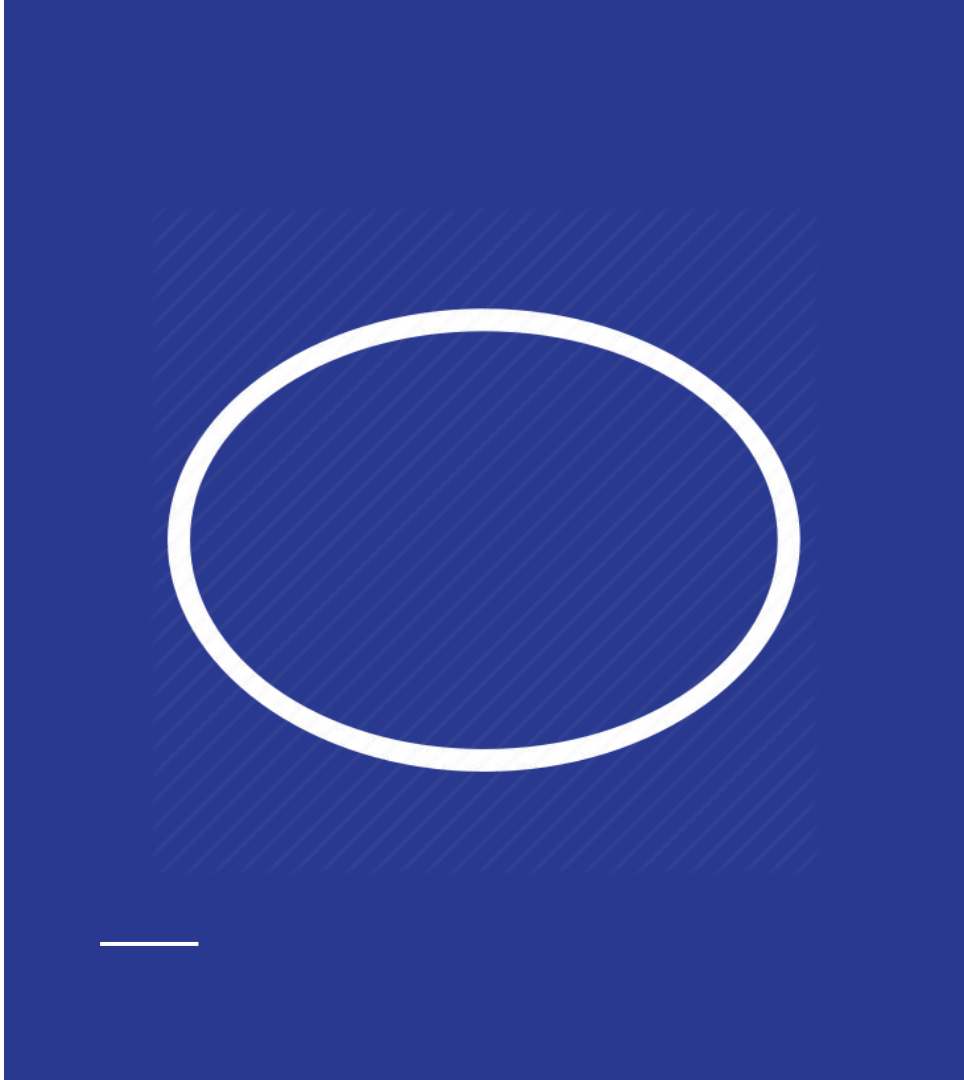


2.Trace a square

3. Spiral

4. Solomon’s star

Try this: Ellipse



Thank you!