Introduction to ROS

ROS: Robot Operating System

- What is it?
- Brief History
- Key ROS Concepts: Nodes & Publishers
- Getting started with ROS: workspaces & packages

Movebase:

- A basic starting point for motion control under ROS

Why ROS?

Robots are *computer-controlled electromechanical* devices

- First dedicated robot programming languages in the 1970's
  - Robot-centric data types and some robot function libraries
  - Didn't allow for much hardware abstraction, multi-robot interaction, helpful human interface, or integrated simulation.
  - Not much code reuse, or standardization
- Efforts to build robot programming systems continued through 80's, 90's
- Several efforts beginning in the 2000's to *standardize* robot components, their interfaces, and basic functions. Sensing, computation, communication become *cheap*, and *distributed*

As robot components and computers became standardized:

- Need fast *prototyping* (fast debugging, pre-existing drivers, ….)
- Want plug-and-play characteristic for basic hardware modules
- Linux model of community development and contributions

High Level View of ROS

A mix of **“Meta” operating system, “Middleware”,** and **programming model**

- A set of libraries, tools, and “packages”
  - Allows for hardware abstraction across different robot platforms
  - Low level device control
  - Encourages *Code Reuse* so that you can build on others’ work
  - Tool-based development
- Provides computation models and communication protocols
- Supports Multiple Development Languages (C++, Python, Java, MATLAB, LISP, ….)
- Scalable (in theory) to large systems and system-level development
- Not quite “real-time”, but can work with real-time modules

Works under Ubuntu computer operating system

- In *theory* it works in Windows: http://wiki.ros.org/Installation/Windows
- In *practice*, dual boot or virtual machine (https://itsfoss.com/install-linux-in-virtualbox/) is better

High Level View of ROS

Peer-to-Peer philosophy

- Main functions are in “nodes”, whose computation can be distributed anywhere
  - A node is a “process.”
    - There can be multiple processes on one CPU (time sharing).
    - A node can be dedicated to one core in a CPU
    - Nodes need not even be on the same physical computer, or even robot.
- Communication via messages
  - one-to-many communication model (publish, subscribe)
  - Many-to-many communication model is possible, but not desirable
- “Services” are the third main organizational unit in ROS
- ROS is meant to be “thin”: Users create self-contained functions/libraries that communicate via ROS messages

Main Aspects of ROS

Software Development & Implementation Infrastructure

- Message passing & communication protocols
  - Memory & buffer management
- Low level device & hardware control
  - Common sensors and input devices
- Key robot data structures, such as frames, and their management
- Start-up and system configuration
- Data logging

- Tools to managing package development
- Debugging tools
- Simulation & Visualization Tools

User Contributed & Specialized "Packages"

- Implement Key Robot Functions
  - SLAM
  - Navigation & Motion Planning
  - Perception
    - Vision
    - Lidar Processing
- Hardware-specific packages
  - E.g., Velodyne VLP-15 "driver"
- Visualization add-ons
- ……

A brief ROS History

Originated by a grad student at Stanford AI Lab ~2007.

Taken up and developed by *Willow Garage*

- a now defunct, but influential, robotics start-up
- Probably the driving influence behind ROS adoption

Since 2013, supported by the ***Open Source Robotics Foundation*** (OSRF)

- Openrobotics.org
- Some Caltech Alums work for/with the foundation

A series of “releases” define different generations of ROS

- There are several good tutorials, and even books, on ROS (see later in the slides)
- But some of the “details” can become obsolete in newer releases

Some ROS Resources

ROS Wiki: http://wiki.ros.org/

- Tutorials: http://wiki.ros.org/ROS/Tutorials
- Instructions for downloading & installing ROS
  - http://wiki.ros.org/ROS/Installation
- Information on packages available for specific robots:
  - http://wiki.ros.org/Robots
- FAQ and User Questions: https://answers.ros.org/questions/

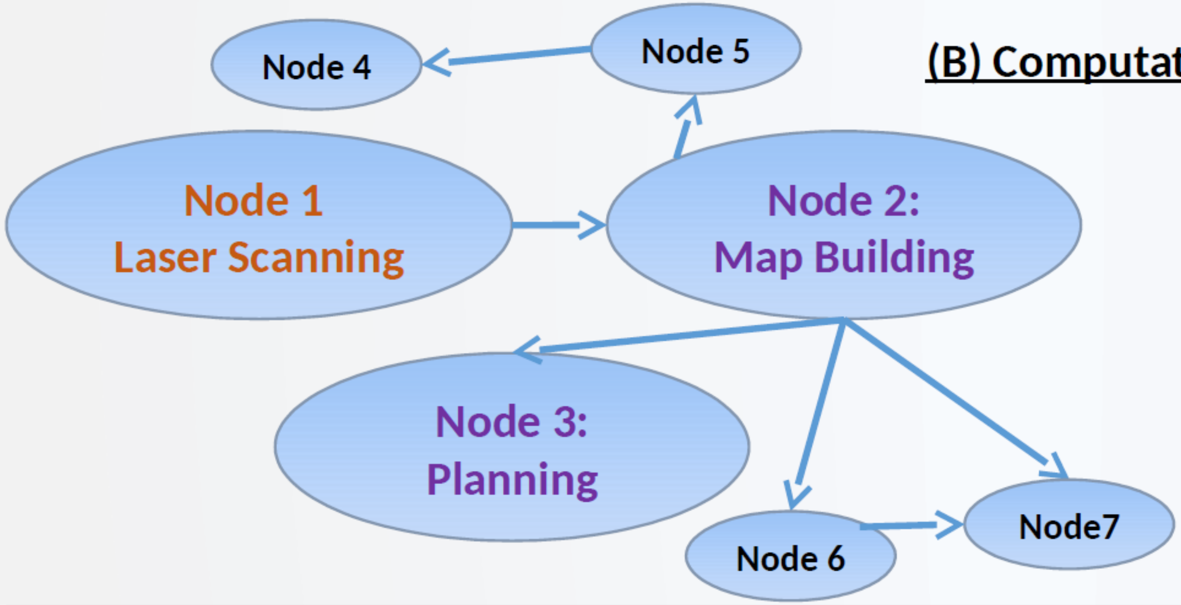
On-line ROS books & tutorials

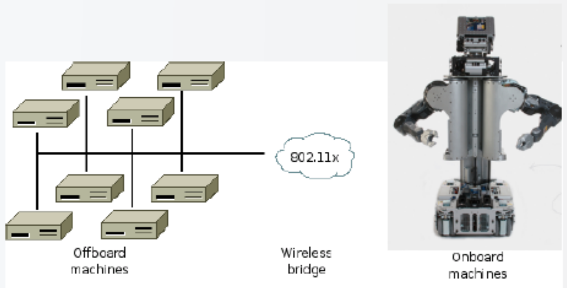
- "*A Gentle Introduction to ROS*", Jason O'Kane (2016)
  - https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf
- "A Guided Journey to the Use of ROS," G.A. Di Caro
  - https://web2.qatar.cmu.edu/~gdicaro/16311/slides/start-with-ros.pdf

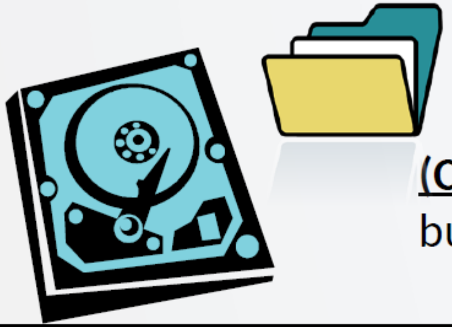
Conceptual levels of design

(A) ROS Community: ROS Distributions, Repositories

(B) Computation Graph: Peer-to-Peer Network of ROS nodes (processes).







(C) File-system level: ROS Tools for managing source code, build instructions, and message definitions.

ROS Nodes

Node:

- Single purpose, executable program
  - Can contain many functions, can call other nodes
- Nodes are assembled into a graph (via communication links)
  - Communication via topics, or with a service, or with a parameter server

Examples:

- sensor or actuator driver, control loop (steering control in RC car)
- Motion planning module

Programming: Nodes are developed with the use of a ROS *client library*

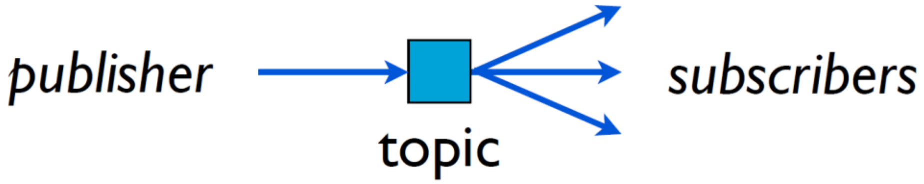
  - *Roscpp* for C++ programs, *rospy* for python programs.
- Nodes receive data by *subscribing* to a *topic*
- Nodes can make data available to other nodes by *publishing* to a *topic*
- Nodes can provide or use a *service*.

ROS Topic

Topic:

- A topic is a name for a *data stream* (TCP or UDP)
- A message *bus* over which nodes exchange *messages*
    - E.g., *lidar* can be the topic that a robot's on-board LiDAR uses to communicate its sensor data. The data could be *raw*, or it could be *preprocessed* by the lidar sensor node. It can send data once, or repeatedly.
- Topics are best for *unidirectional, streaming* communication. A request/response model is handled by a *service.* Fixed data is handled by a *parameter server.*
- Topic statistics available: age of data, traffic volume, # dropped messages

Publish: 1-to-N communication model



Subscribe:

- If a node *subscribes* to a topic, then it receives and understands data published under that topic.

ROS Messages

Messages are published to topics

Message Format:

- Strictly typed *data structure:*
  - Typed fields (some are predefined in *std_msgs*) ,
  - but user definable as well

E.g.

float64 x			vector3 linear_velocity
float64 y	vector3 position		vector3 angular_velocity
float64 z			

- .msg *text files* specify the data structure of a message, and are stored in message subdirectory of a *package*

Message Guarantees:

- Will not block until receipt, messages get queued
- Can set buffer length: e.g., *N* messages before oldest is thrown away

- Example: *built-in laser scan data message*


```
--- sensor_msgs/msg/LaserScan.msg ---

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

- Another example: *remote interface service in Cobot*

request

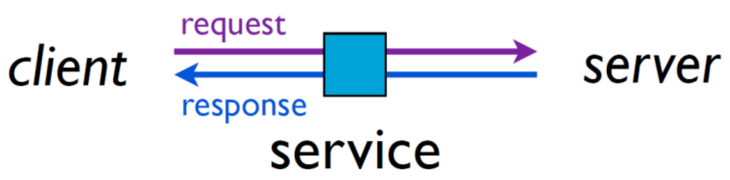
response

```
--- cobot_msgs/srv/CobotRemoteInterfaceSrv.srv ---

# "Joystick" velocity commands:
float32 drive_x #Distance to move along x in meters
float32 drive_y #Distance to move along x in meters
float32 drive_r #Distance to turn in radians

# command_num must increment every time the service is called - used to reject out of sync commands
int32 command_num

# valid command flags:
#   CmdMove = 0x0001
#   CmdSetLocation = 0x0002
#   CmdGetLocation = 0x0004
#   CmdGetParticlesSampling = 0x0010
#   CmdSetTarget = 0x0020
int32 command_type

# The following parameters are used for commands CmdSetLocation and CmdSetTarget
float32 loc_x
float32 loc_y
float32 orientation
string map

---

float32 loc_x
float32 loc_y
float32 orientation

float32[] particles_x
float32[] particles_y
float32[] particles_weight
float32[] locations_weight

int8 err_code
```

ROS Service

Service:

- A mechanism for a node to send a request to another node, and receive a response:
  - Synchronous node interaction
  - two way communication
  - Trigger functions and "behaviors"



- Uses a *request-response* paradigm:
  - A *request structure* contains the message to request the service
  - A *response structure* is returned by the service
  - Analogous to a **Remote Procedure Call** (RPC)

Examples:

- Request an updated map, or portion of a map from a "map server"
- Request and receive status information from another vehicle

Parameter Server

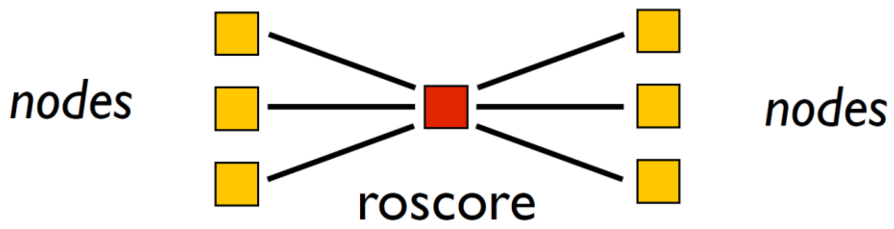
A shared "Dictionary"

- Best used for *static* data, such as parameters that are needed at start-up.
- Runs in the ROS master
- E.g.:
  - lidar scan rate
  - Number of Real-Sense sensors in a networked sensing situation

ROS Master

Master: Matchmaker between nodes

- Nodes make be on different cores, different computers, different robots, even different networks. This should be transparent to each node's code
- The "master" service runs on *one* machine.
    - It provides name registration & lookup of nodes and services
- *roscore* starts the master server, parameter server, and logging processes (if any)
- *Roscore* acts like a name server so that nodes get to know each other

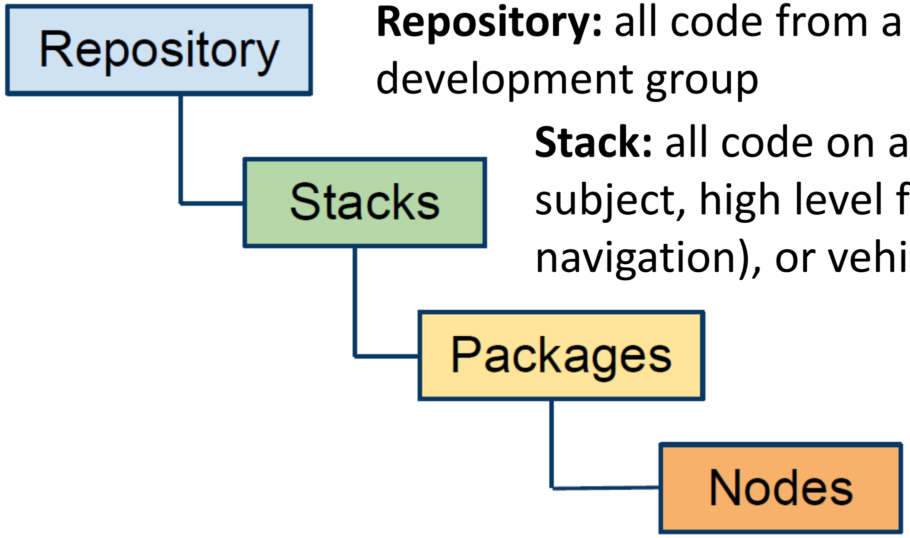


- Every node connects to the master at start-up to register details of the message streams that it publishes. Also determine its connectivity with the rest of the computation graph via its subscriptions.

ROS Packages

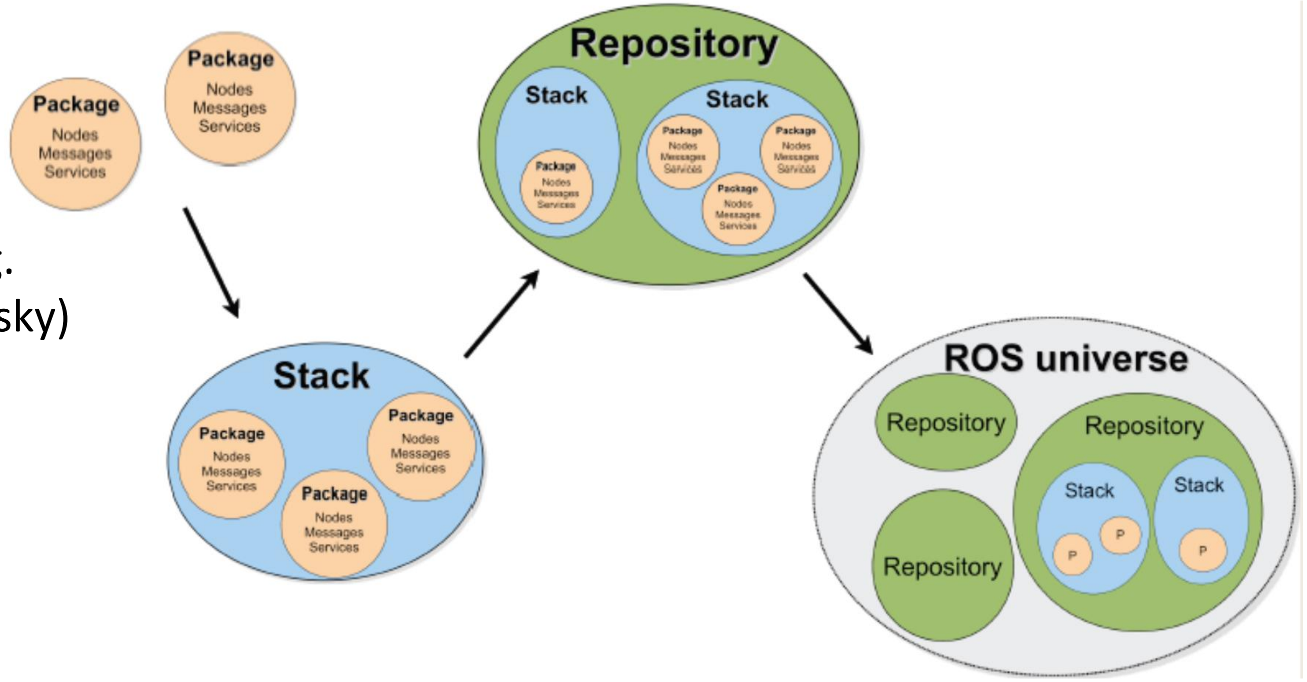
Package: Basic organizational and *code reuse* unit of ROS software

- Contains one or more nodes & provides a ROS interface (via messages, services)
- Typically implements a well defined function, like making a map from sensory data
- Organized into a self-contained directory (with a specific structure) containing source code for nodes, message definitions, services, etc.



Repository: all code from a development group

Stack: all code on a particular subject, high level function (e.g. navigation), or vehicle (e.g., husky)



ROS *Distribution*

A versioned set of ROS Packages

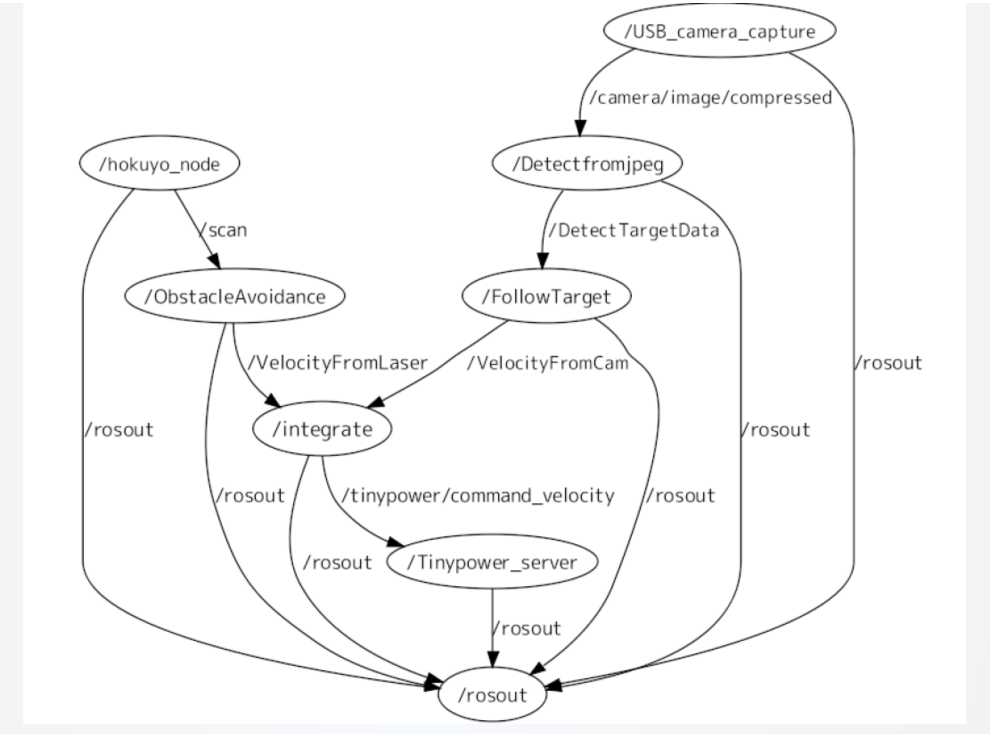
- Like a Linux distribution
- Provide a *relatively* stable codebase for development.
- Primarily for core ROS components
  - User contributed packages must make their own updates

ROS Noetic Ninjemys	May, 2020 (planned, see Upcoming Releases (/Distributions#Upcoming_releases))	TBA	TBA	May, 2025 (planned)
ROS Melodic Morenia (/melodic) (**Recommended**)	May 23rd, 2018	 (/melodic)		May, 2023 (Bionic EOL)
ROS Lunar Loggerhead (/lunar)	May 23rd, 2017	 (/lunar)		May, 2019
ROS Kinetic Kame (/kinetic)	May 23rd, 2016	 (/kinetic)		April, 2021 (Xenial EOL)
ROS Jade Turtle (/jade)	May 23rd, 2015	 (/jade)		May, 2017
ROS Indigo Igloo (/indigo)	July 22nd, 2014	 (/indigo)		April, 2019 (Trusty EOL)

Many ROS Tools

Developer Tools:

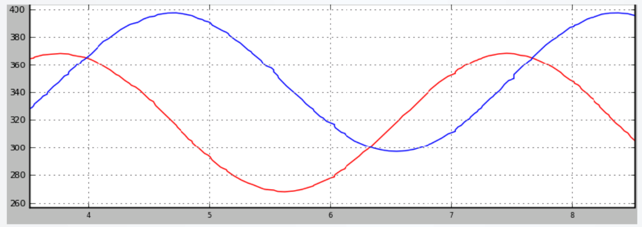
- Building ROS nodes: catkin_make
- Running ROS nodes: rosrun, roslaunch
- Viewing network topology: rqt_graph



Debugging Tools:

- **Rostopic:** display info about active topics (publishers, subscribers, data rates and content)
- rostopic echo [topic name]   (*prints topic data)*
- rostopic list *(prints list of active topics)*
- **Rqt_plot:** plots topic data

`rqt_plot  /turtle1/pose/x,/turtle1/pose/y` ✉ graph data from 2 topics in 1 plot

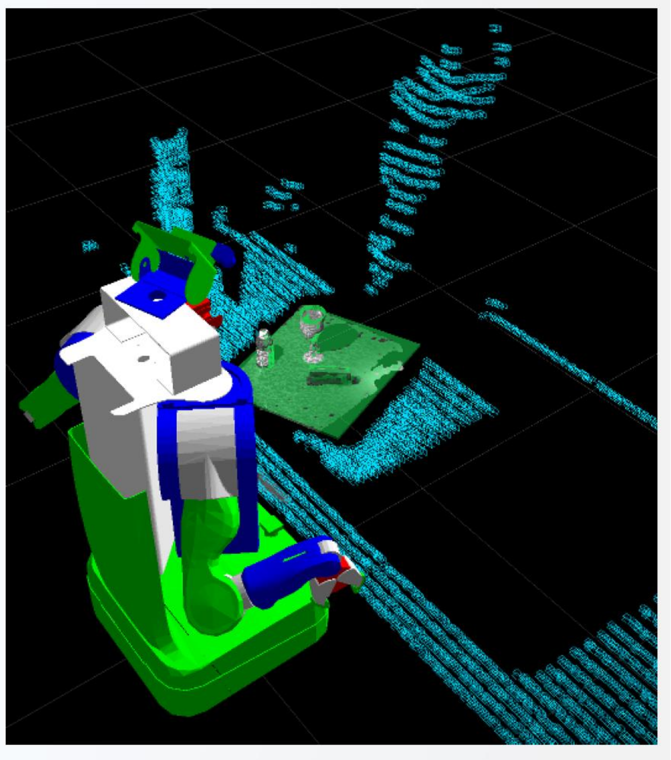


- Data logging:
  - Rosbag record [topics] –o < output_file>
- Data playback:
  - Rosbag play <input_file> --clock

Many ROS Tools

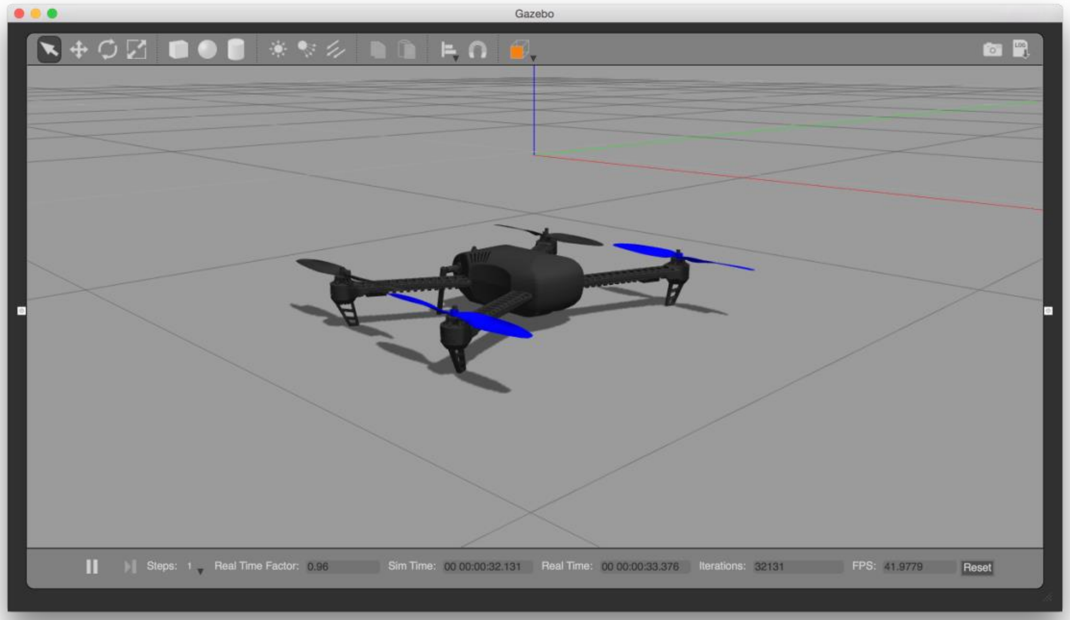
Visualization Tools: RVIZ

- Sensor and robot state data
- Coordinate frames
- Maps, built or in process
- Visual 3D debugging markers



Simulation Tools:

- **Gazebo:** started as grad student project at USC
- Can model and simulate motions/dynamics of different robots
- Can simulate sensory views
- Can build different environments
- Can run simulation from ROS code for testing



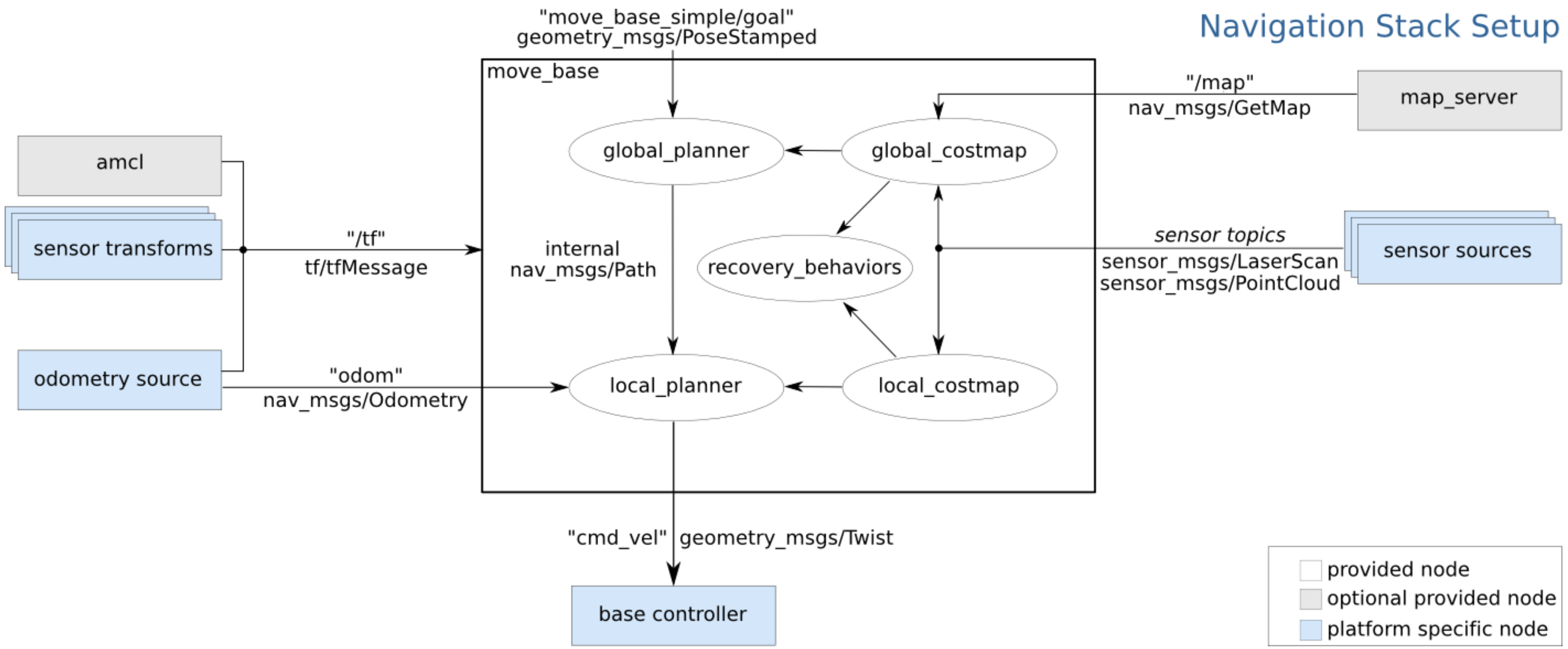
A first look at ***move_base***

move_base is a *package* that implements an *action* in ROS.

- An action can be *preempted*
- An action can provide periodic feedback on its execution

move_base is a node that moves a robot (the “base”) to a goal

- It links a *global* and *local* planner with sensory data and maps that are being built, so that the *navigation stack* can guide the robot to a goal, and have *recovery strategies*



Goals for Next Week

Download ROS distribution.

- Choose how you want to manage Ubuntu on your machine:
  - Dual boot
  - Virtual machine: (one option is the free *virtual box:* https://itsfoss.com/install-linux-in-virtualbox/
  - Try the Windows installation?
- Install ROS (melodic is best, but kinetic might be okay)

GO through the first 2-3 steps of the *Core ROS Tutorial* at the beginner's level.

- You may prefer to start the first few steps of "*A Guided Journey to the Use of ROS*"