Robotic Operating System

Prof. V Muthukumar

ROS IN MOTION

What is ROS?

- ROS is an open-source **robot operating system**
- The primary goal of ROS is to support code *reuse in robotics research and development*
- ROS was originally developed in 2007 at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage
- Today managed by the Open Source R Foundation

ROS

Open Source Robotics Foundation

What do we do with ROS?

- Understand the working of ROS
- ROS simulation with C++ and Python
- Interface ROS with hardware
- Interface ROS with our Mobile Robot
- Collect and visualize data

Robots with ROS


210 Stanley Innovation V3 Segway


220 Stanley Innovation V3 Segway


420 Omni Stanley Innovation V3 Segway


440LE Stanley Innovation V3 Segway


Eddiebot


Enova Robotics MiniLab


Erle-Brain


Erle-Copter


440SE Stanley Innovation V3 Segway


ABB Robotics (ROS-Industrial)


Adept MobileRobots Pioneer family (P3DX, P3AT, ...)


Adept MobileRobots Pioneer LX


Erle-Copter Ubuntu Core special edition


Erle-HexaCopter


Erle-Plane


Erle-Rover


Adept MobileRobots Seekur family (Seekur, Seekur Jr.)


Aldebaran Nao


Allegro Hand SimLab


AMIGO


evarobot


Fanuc Robotics (ROS-Industrial)


Festo Didactic Robotino


Fetch robotics: Fetch


AscTec Quadrotor


Barrett Hand


BipedRobin


Bitcraze Crazyflie


Fetch robotics: Freight


Fraunhofer IPA Care-O-bot 3


Fraunhofer IPA Care-O-bot 4


Gostai Jazz


Clearpath Robotics Grizzly


Clearpath Robotics Husky


Clearpath Robotics Jackal


Clearpath Robotics Kingfisher


GoThere! Robot


i-Cart mini


Innok Heros


Intel Edison

CoroWare Corobot

Cyton-Gamma

Denso VS060

Dr. Robot Jaguar

iRobot Roomba

Kawada Nextage / Hiro

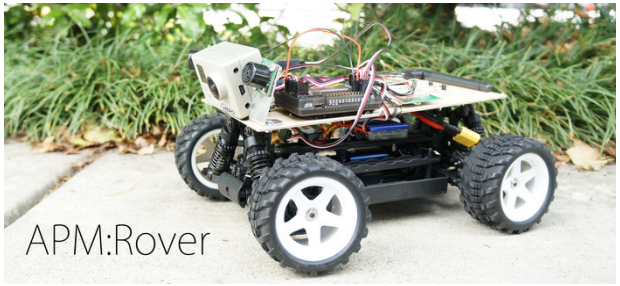
Kinova JACO

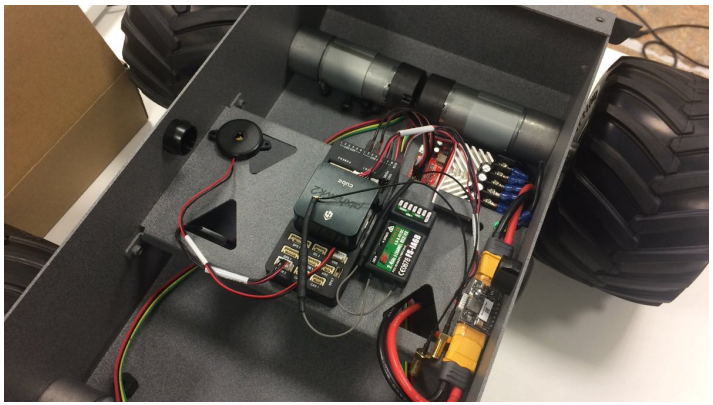
Kinova MICO

ROS IN MOTION

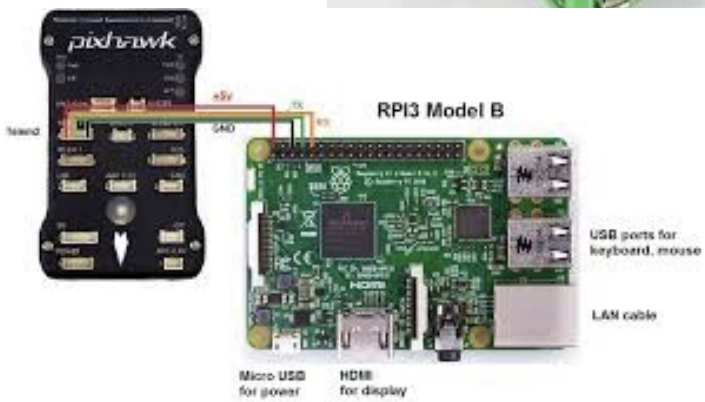
PX4 Offboard Control Using MAVROS on ROS

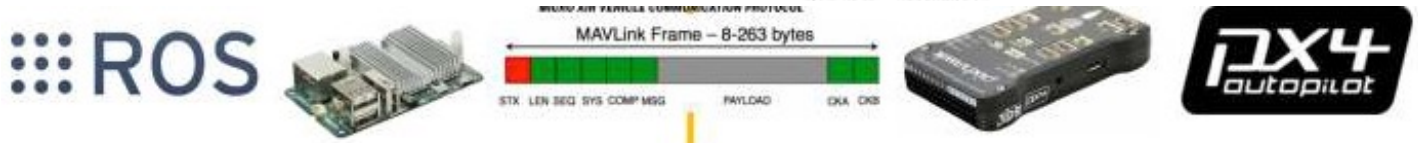
ArduPilot Rover

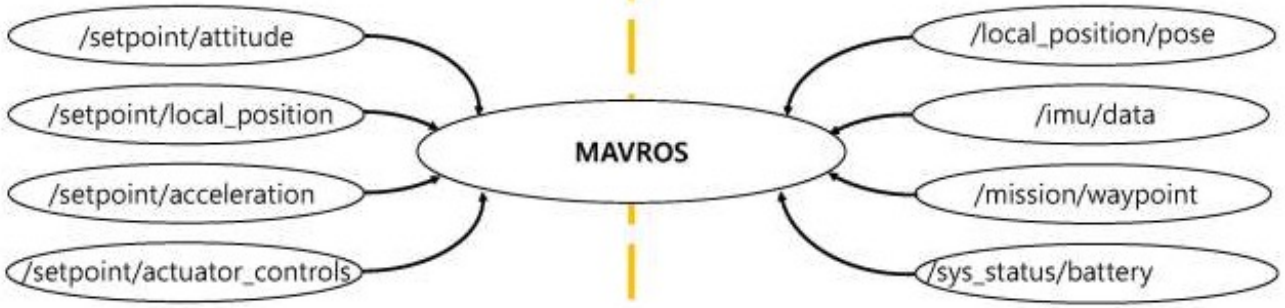




Pixhawk

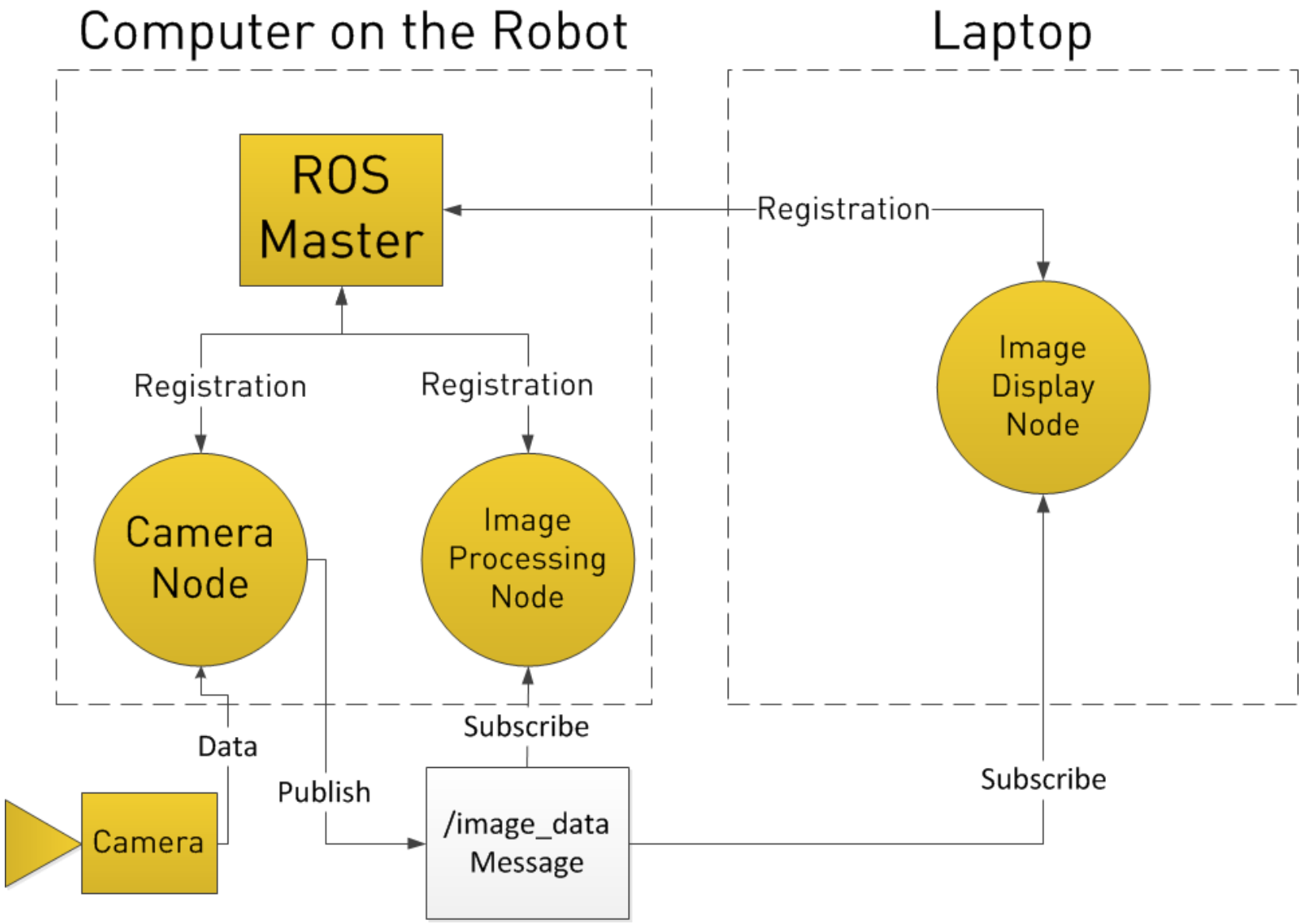






ROS IN MOTION

ROS Example

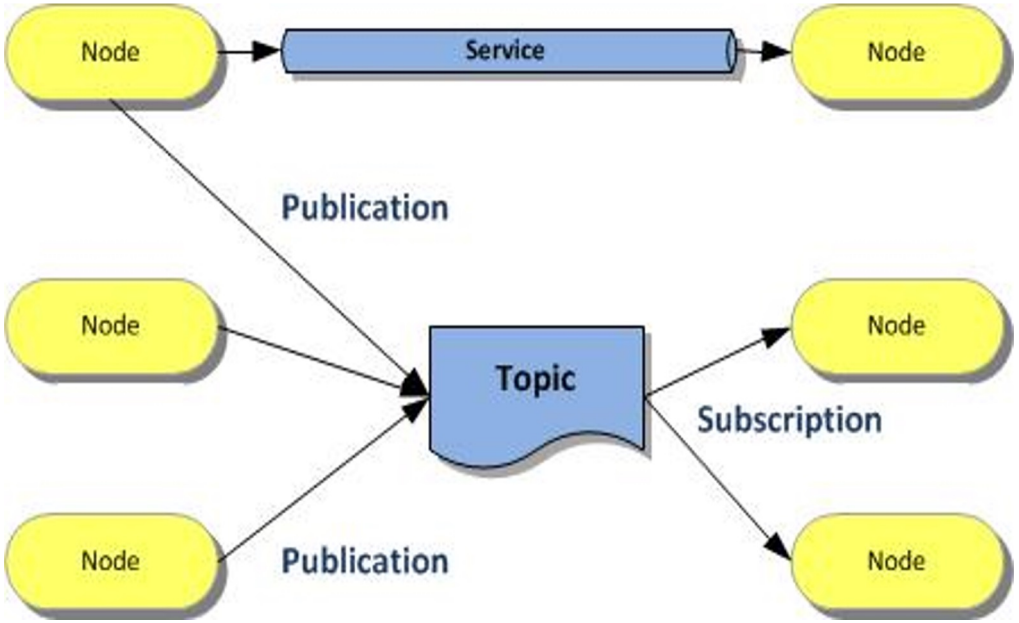

Computer on the Robot
Laptop
ROS Master
Registration
Image Display Node
Registration
Registration
Camera Node
Image Processing Node
Subscribe
Data
Publish
Subscribe
Camera
/image_data Message

Components of ROS

Client Layer	roscpp	rospy	roslisp	rosjava	roslibjs		
Robotics Application	MoveIt!	navigatioin	executive smach	descartes	rospeex		
	teleop pkgs	rocon	mapviz	people	ar track		
Robotics Application Framework	dynamic reconfigure	robot localization	robot pose ekf	Industrial core	robot web tools	ros realtime	mavros
	tf	robot state publisher	robot model	ros control	calibration	octomap mapping	
	vision opencv	image pipeline	laser pipeline	perception pcl	laser filters	ecto	
Communication Layer	common msgs	rosbag	actionlib	pluginlib	rostopic	rosservice	
	rosnode	roslaunch	rosparam	rosmaster	rosout	ros console	
Hardware Interface Layer	camera drivers	GPS/IMU drivers	joystick drivers	range finder drivers	3d sensor drivers	diagnostics	
	audio common	force/torque sensor drivers	power supply drivers	rosserial	ethercat drivers	ros canopen	
Software Development Tools	RViz	rqt	wstool	rospack	catkin	rosdep	
Simulation	gazebo ros pkgs	stage ros					

ROS Core Concepts

- Nodes
- Messages and Topics
- Services
- ROS Master
- Parameters
- Stacks and packages



ROS Computation Graph Level

- **Nodes**: Nodes are processes that perform computation.
- **Master**: The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Messages**: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

ROS Computation Graph Level

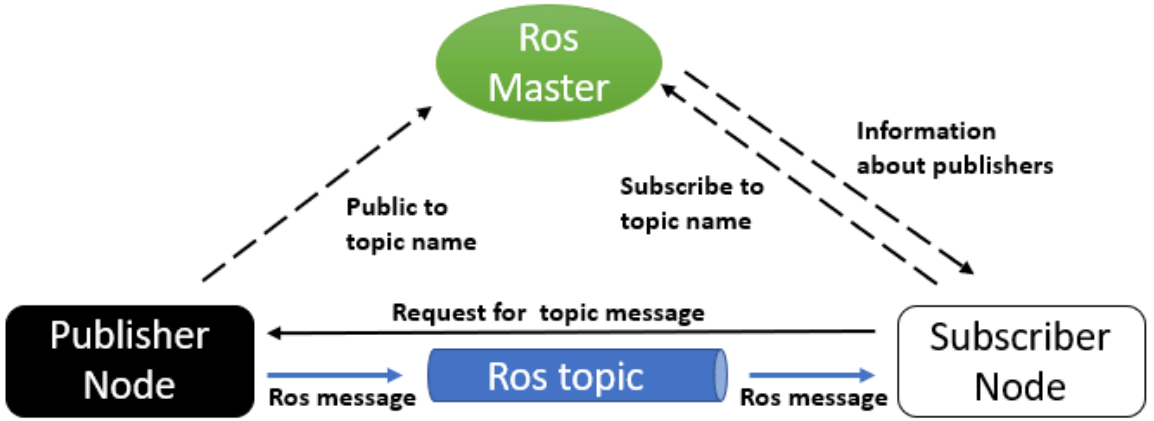
- **Topics (async)**: Messages are routed via a transport system with publish / subscribe semantics.
- A node sends out a message by *publishing* it to a given topic. The topic is a name that is used to identify the content of the message.
- A node that is interested in a certain kind of data will *subscribe* to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics.
- In general, publishers and subscribers are not aware of each others' existence.

ROS Computation Graph Level

- **Services (sync)**: The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system.
- Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.
- Service/Client model: 1-to-1 request-response
- **Bags**: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

ROS Nodes

- Single-purposed executable programs
  - e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc.
- Modular design
  - Individually compiled, executed, and managed
- Nodes are written using a ROS client library
  - roscpp – C++ client library
  - rospy – python client library
- Nodes can pu
- Nodes can als



C++ file of a node

Create a ROS pkg:

```
$ cd ~/ros_workspace/src
$ catkin_create_pkg
tutorial_pkg roscpp
```

```cpp
#include <ros/ros.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "example_node");
    ros::NodeHandle n("~");
    ros::Rate loop_rate(50);
    while (ros::ok())
    {
        ros::spinOnce();
        loop_rate.sleep();
    }
}
```

CMakeLists.txt

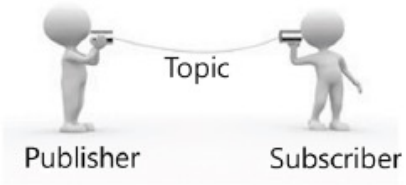
```
cmake_minimum_required(VERSION 2.8.3)
project(tutorial_pkg)

add_compile_options(-std=c++11)

find_package(catkin REQUIRED COMPONENTS
  roscpp
)

catkin_package(
  CATKIN_DEPENDS
)

include_directories(${catkin_INCLUDE_DIRS})

add_executable(${PROJECT_NAME}_node src/tutorial_pkg_node.cpp)

target_link_libraries(${PROJECT_NAME}_node
  ${catkin_LIBRARIES}
)
```

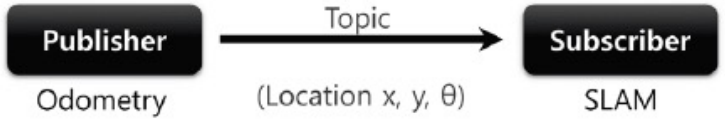
Build a ROS pkg:

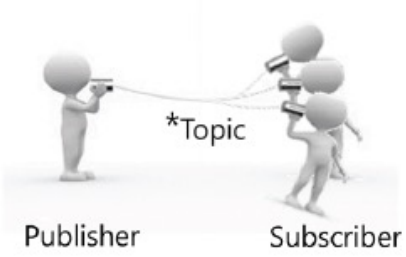
```
cd ~/ros_workspace
catkin_make
```

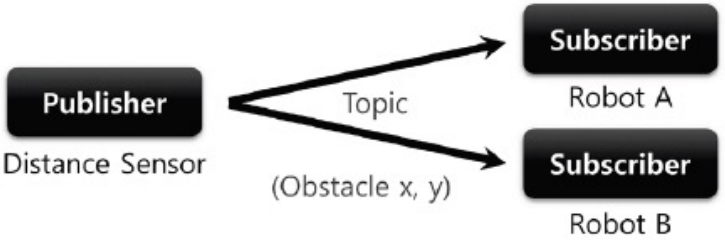

ROS Topics

- Nodes communicate with each other by publishing messages to topics
- Unidirectional
- Publish/Subscribe model: 1-to-N broadcasting
- A shared topic can be used to send messages b









ROS Messages

- Strictly-typed data structures for inter-node communication
- For example, geometry_msgs/Twist is used to express velocity broken into linear and angular parts:

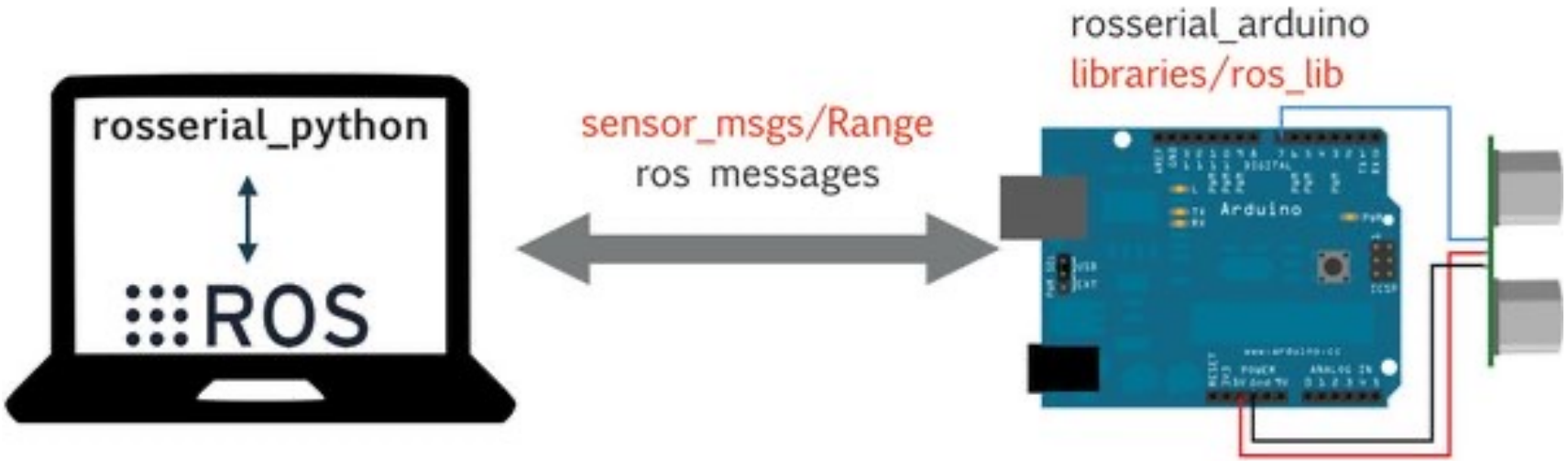
```
Vector3 linear
Vector3 angular
```

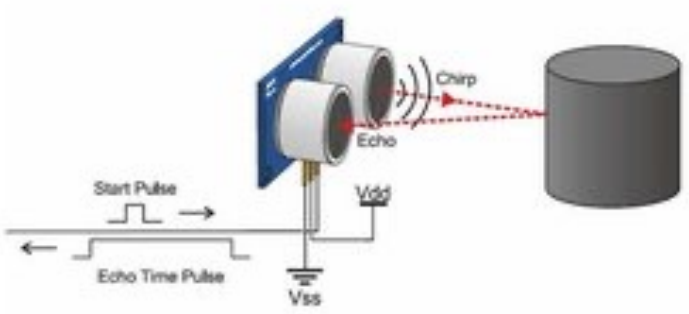
- Vecto… ge type (

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

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  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
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

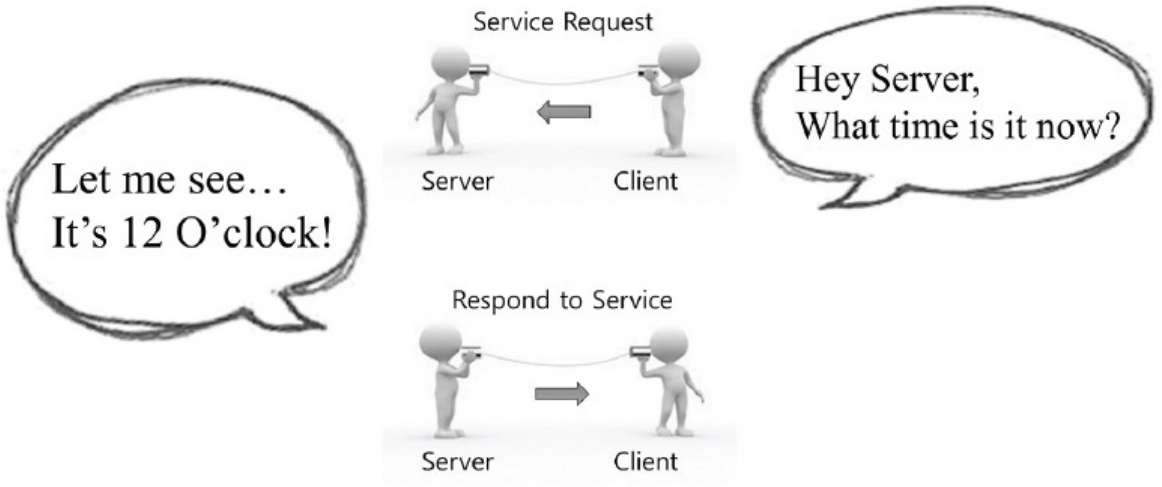

Practical example of Nodes and Topics





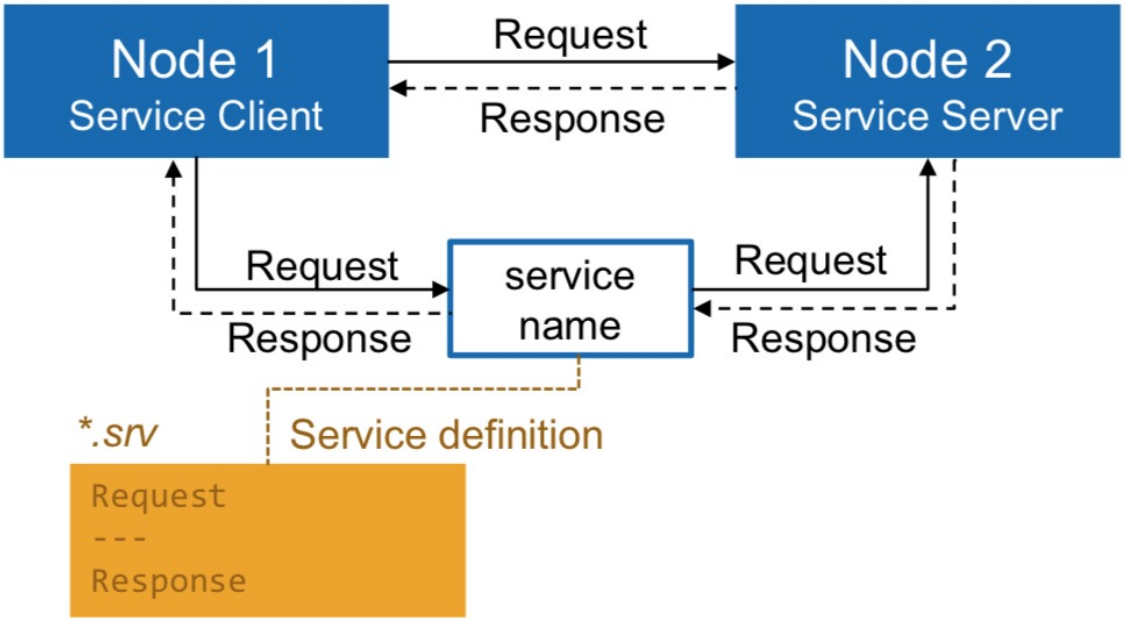
ROS Services

- Synchronous Bi-directional inter-node transactions
- Service roles:
  - carry out remote computation
  - trigger functionality / behavior

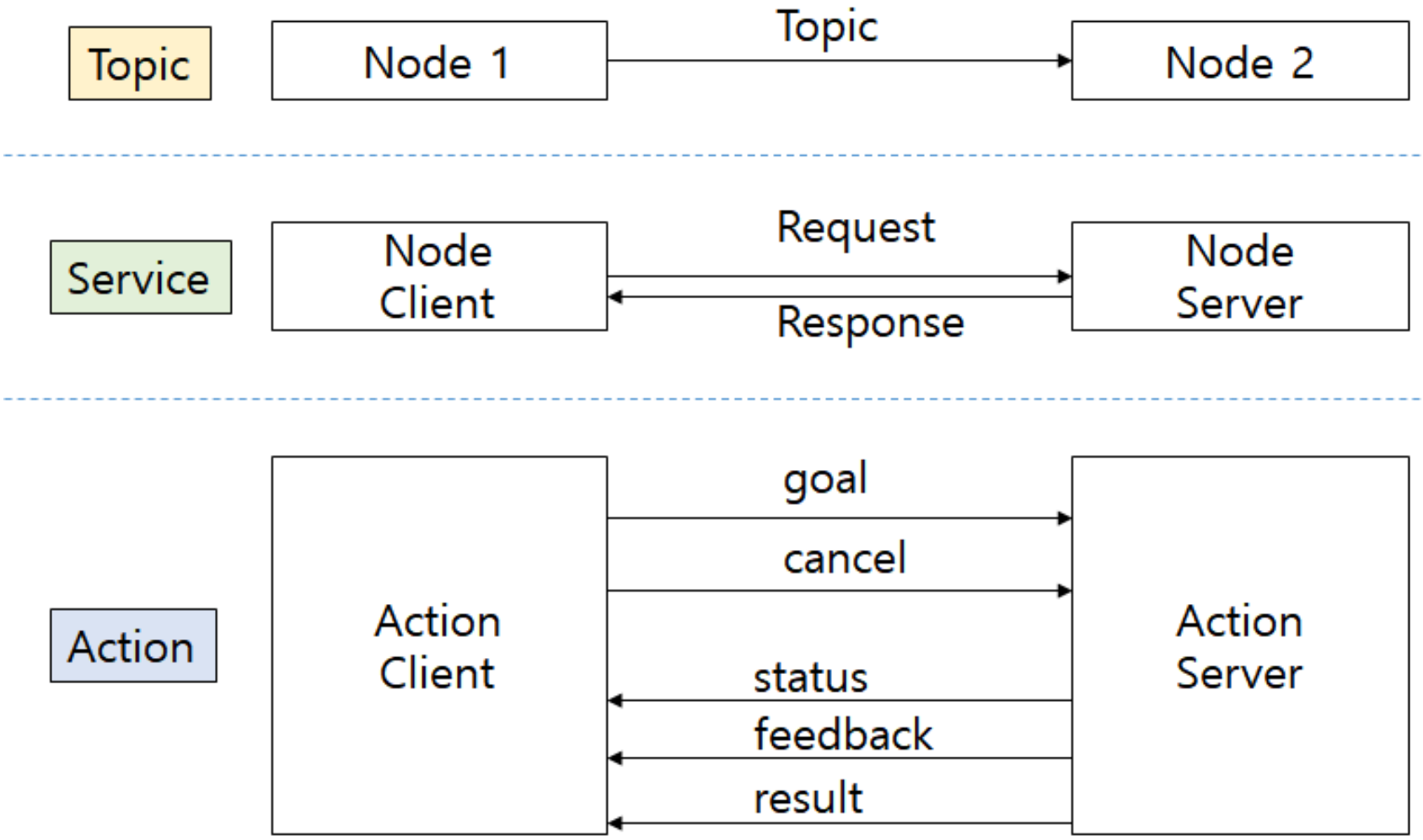


- Action
  - Asynchronous Bi-directional
  - Used when it is difficult to use the service due to long response times after the request or when an intermediate feedback value is needed

Service Server & Action Server

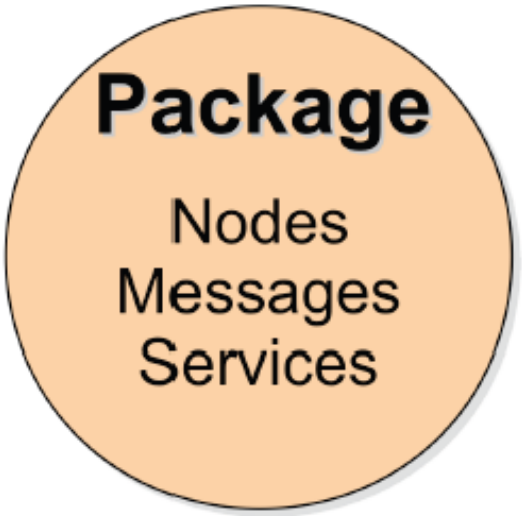


Difference b/w Nodes, Services, & Action

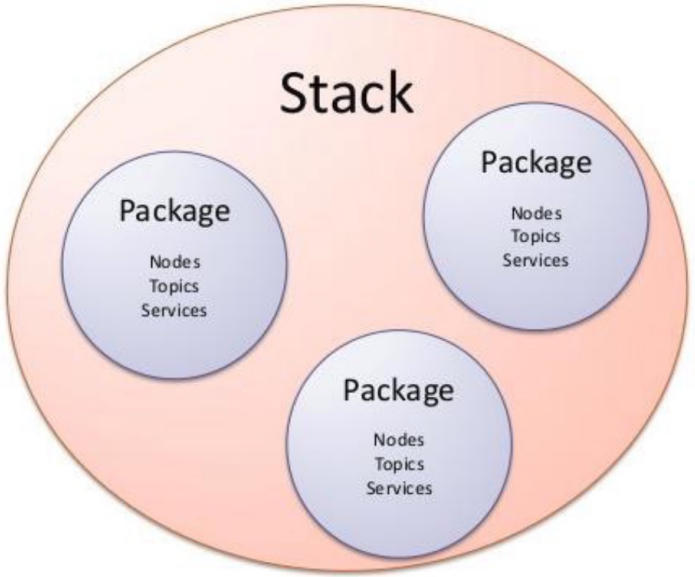


ROS Packages

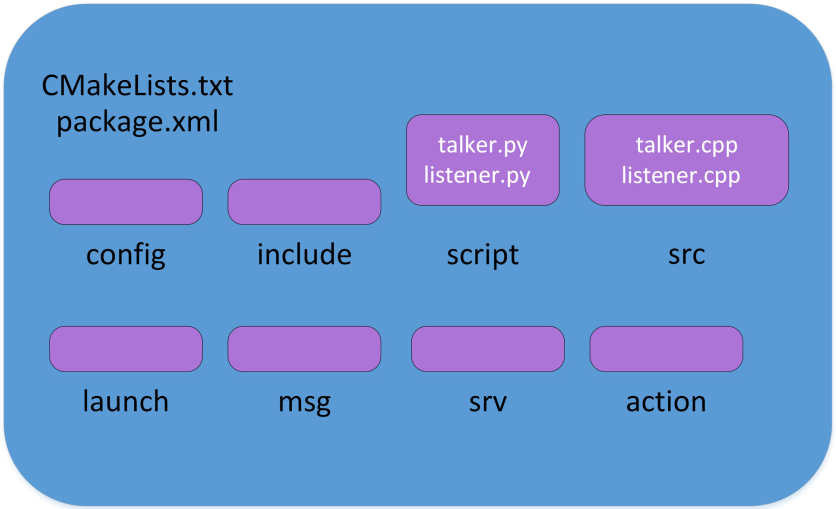
- Software in ROS is organized in *packages*.
- A package contains one or more nodes and provides a ROS interface
- **ROS Stack**
  - Packages in ROS are organized into *ROS stacks*
- **ROS Repository**



ackages



ROS Package Structure



ROS Package Creation:

```
ros_workspace
├── build
│   └── ...
├── devel
│   └── ...
└── src
    ├── CMakeLists.txt
    └── tutorial_pkg
        ├── CMakeLists.txt
        ├── include
        │   └── tutorial_pkg
        ├── package.xml
        └── src
```


ROS Important Packages

Package	
TF	Maintains the relationship between multiple coordinate frames over time
actionlib	Provides a standardized interface for interfacing with pre-emptable tasks.
gmapping	Provides laser-based SLAM (Simultaneous Localization and Mapping) using a grid map
amcl	a probabilistic localization system for a robot moving in 2D
move_base	Print information about a node
stage_ros	Stage 2-D multi-robot simulator

ROS Command-Line Tools

- rostopic (Topics)
- rosservice (Services)
- rosnode (Nodes)
- rosparam (Parameters)
- rosmsg (Messages)
- rossrv (Services)
- roswtf (General debugging)

Command	Description
rostopic list	Show the list of active topics
rostopic echo [TOPIC_NAME]	Show the content of a message in real-time for a specific topic
rostopic find [TYPE_NAME]	Show the topics that use specific message type
rostopic type [TOPIC_NAME]	Show the message type of a specific topic
rostopic bw [TOPIC_NAME]	Show the message data bandwidth of a specific topic
rostopic hz [TOPIC_NAME]	Show the message data publishing period of a specific topic

Command	Description
rosnode list	Check the list of active nodes
rosnode ping [NODE_NAME]	Test connection with a specific node
rosnode info [NODE_NAME]	Check information of a specific node
rosnode machine [PC_NAME OR IP]	Check the list of nodes running on the corresponding PC
rosnode kill [NODE_NAME]	Stop running a specific node
rosnode cleanup	Delete the registered information of the ghost nodes for which the connection information cannot be checked

Command	Description
rosservice list	Display information of active services
rosservice info [SERVICE_NAME]	Display information of a specific service
rosservice type [SERVICE_NAME]	Display service type
rosservice find [SERVICE_TYPE]	Search services with a specific service type
rosservice uri [SERVICE_NAME]	Display the ROSRPC URI service
rosservice args [SERVICE_NAME]	Display the service parameters
rosservice call [SERVICE_NAME] [PARAMETER]	Request service with the input parameter

Command	Description
rosparam list	View parameter list
rosparam get [PARAMETER_NAME]	Get parameter value
rosparam set [PARAMETER_NAME]	Set parameter value
rosparam dump [FILE_NAME]	Save parameter to a specific file
rosparam load [FILE_NAME]	Load parameter that is saved in a specific file
rosparam delete [PARAMETER_NAME]	Delete parameter

ROS commands

- roscore is the first thing you should run when using ROS
  - $ roscore
  - roscore will start up:
    - a ROS Master
    - a ROS Parameter Server
    - a rosout logging node
- rosrun allows you to use the package name to directly run a node within a package
  - $ rosrun turtlesim turtlesim_node
- To display the list of current topics, use the following command:
  - $ rostopic list

Publish to ROS Topic

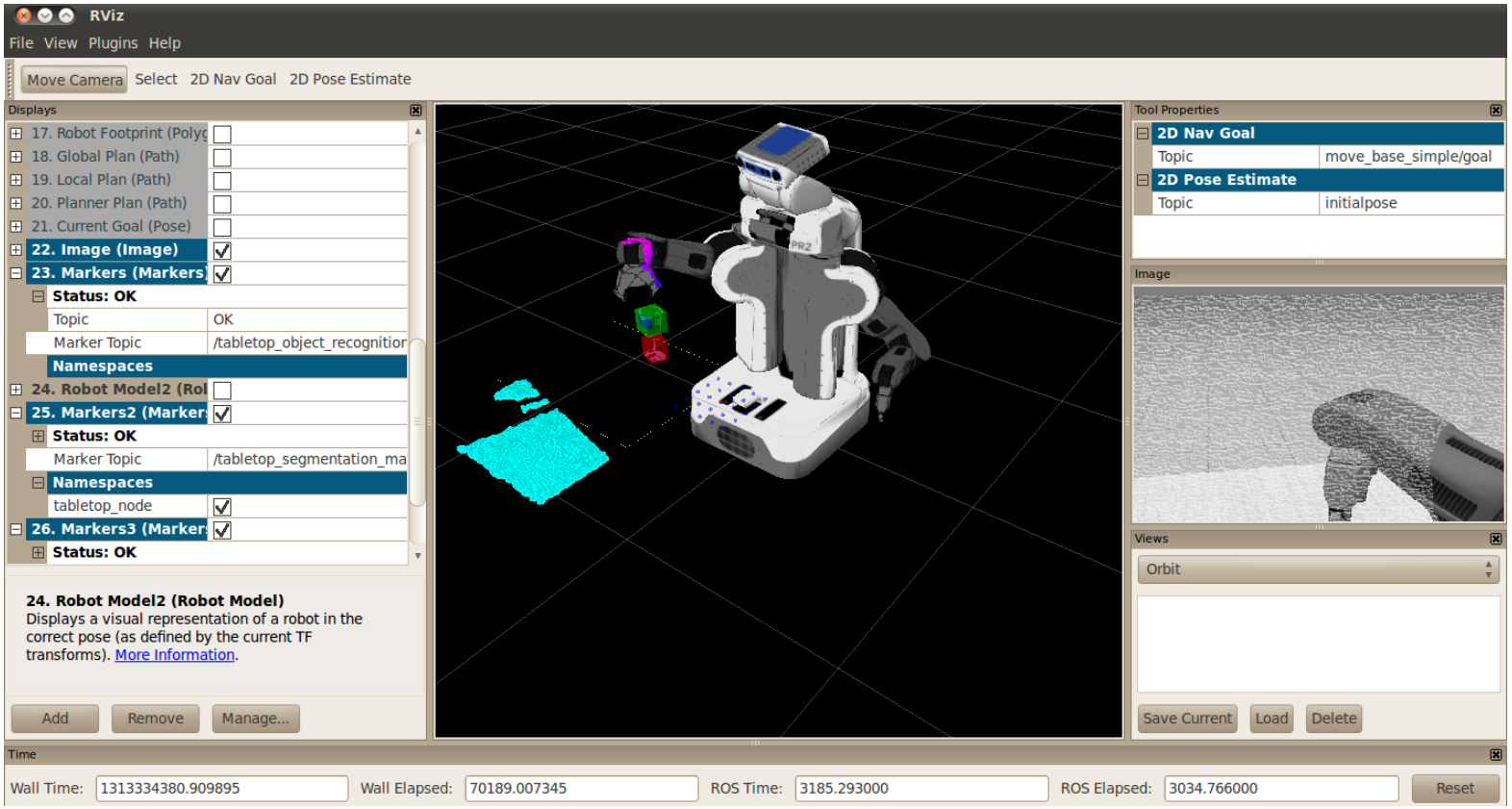
- Use the rostopic pub command to publish messages to a topic
  - $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.2, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
  - predefined timeout
  - Use argument –r (-r 20 (10Hz)) with the loop rate in Hz
- rqt provides the main to start an instance of the ROS integrated graphical user
  - $rqt
- Recording and playing back data - record data from a running ROS system into a .bag file, and then to play back the data to produce similar behavior in a running system.
  - mkdir ~/bagfiles
  - cd ~/bagfiles
  - rosbag record -a

Creating a ROS Package

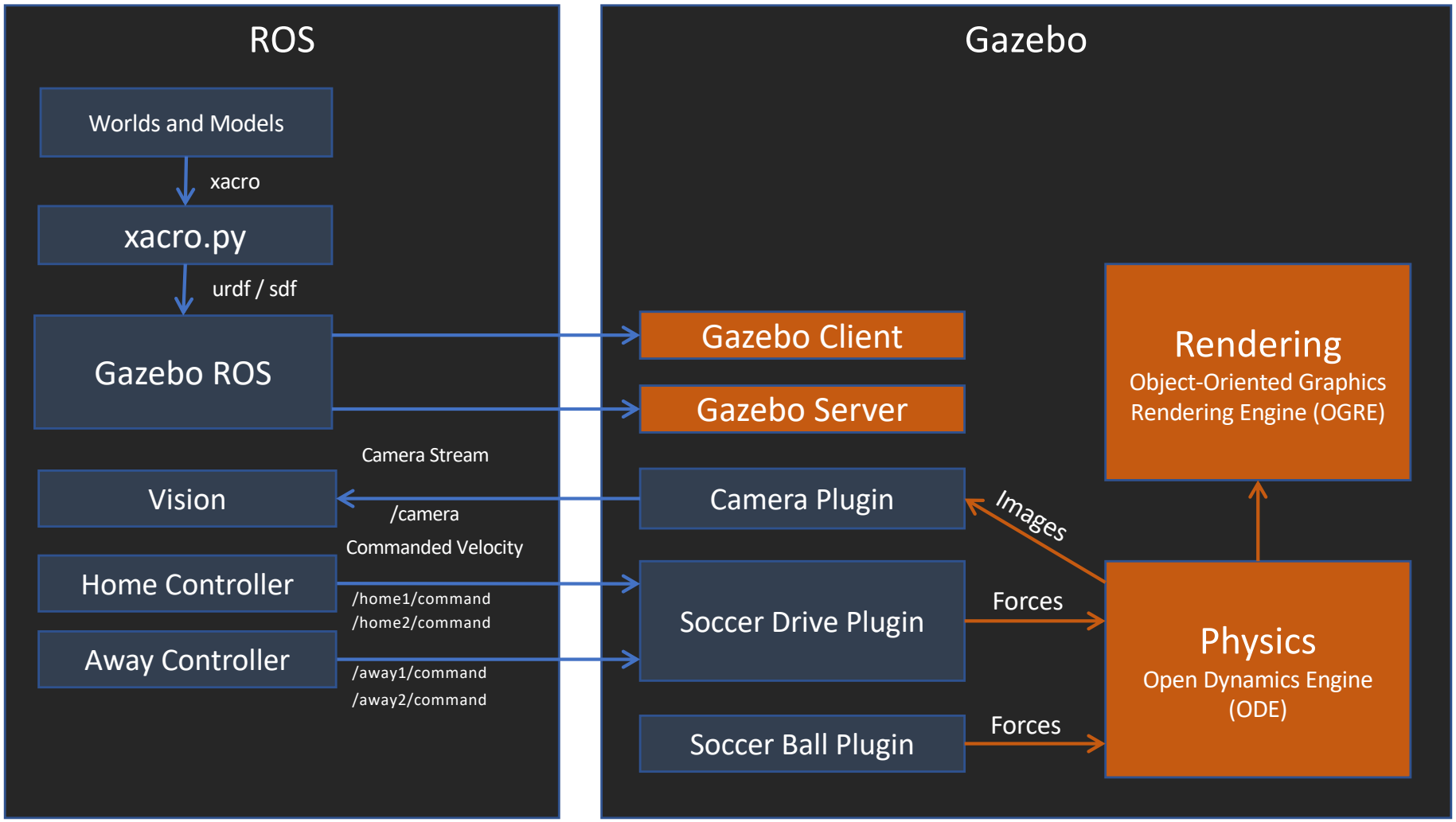
- catkin Workspace
  - $ mkdir catkin_ws catkin_ws/src
  - $ cd catkin_ws/src
  - $ catkin_init_workspace
  - $ cd ..
  - $ catkin_make
  - $ source ~/catkin_ws/devel/setup.bash
- Create a new ROS package called stage_multi
  - $ cd ~/catkin_ws/src
  - $ catkin_create_pkg stage_multi std_msgs rospy roscpp
  - $ mkdir ~/catkin_ws/src/stage_multi/world
  - $ cp ~/willow-multi-erratic.world ~/catkin_ws/src/stage_multi/world
  - $ cp ~/willow-full.pgm ~/catkin_ws/src/stage_multi/world
- Now compile the package and create an Eclipse project file for it:
  - $ cd ~/catkin_ws
  - $ catkin_make --force-cmake -G"Eclipse CDT4 - Unix Makefiles"

ROS Visualization Tools

- rqt – ROS integrated graphical user interface
- rviz – 3D visualization tool for ROS

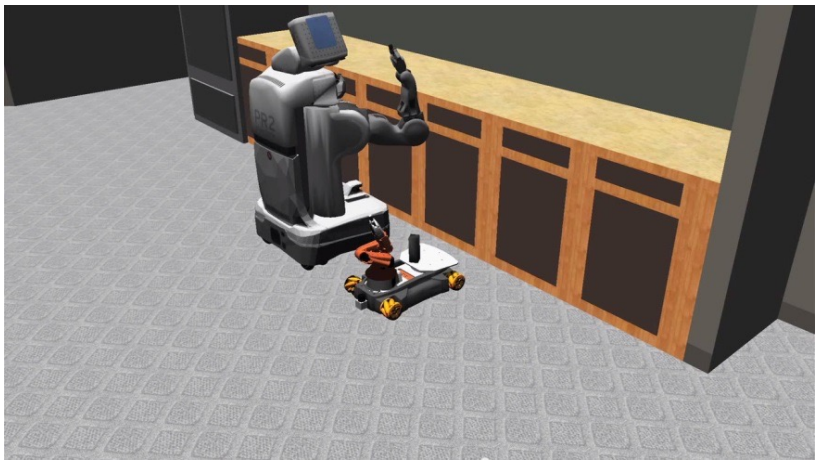


Gazebo Flow Diagram



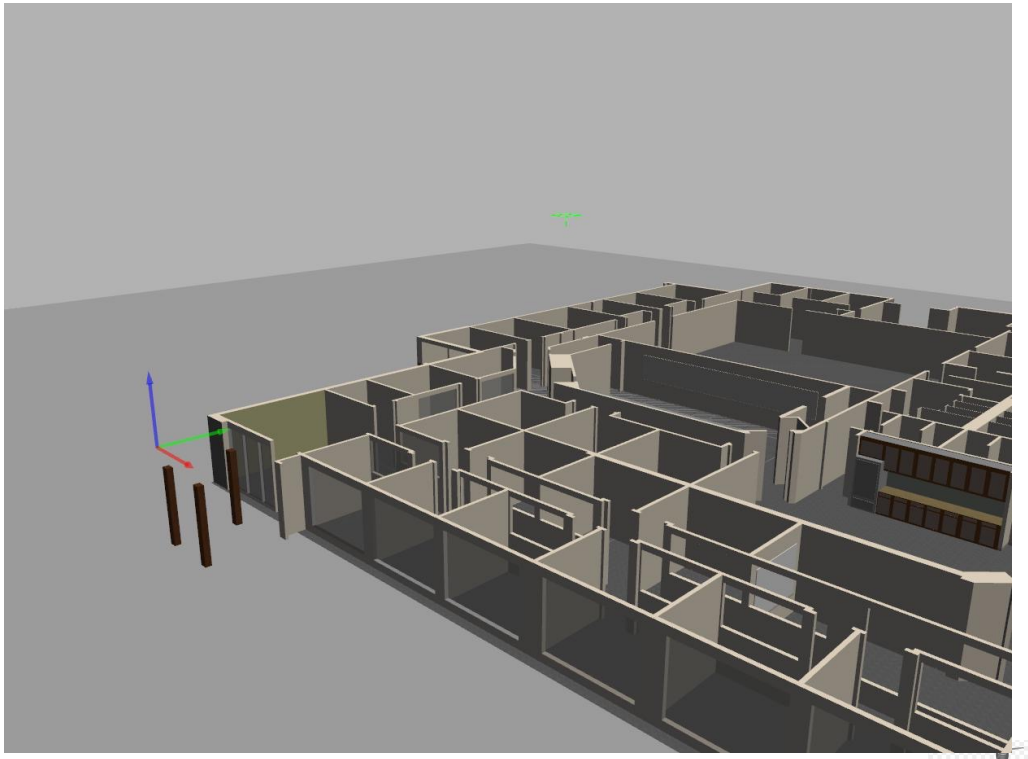
Robot Description Formats

- URDF
  - Legacy Format
  - Usually used by ROS
- SDF
  - Newer
  - More flexible
- Xacro
  - XML Macros
  - Use with URDF or SDF



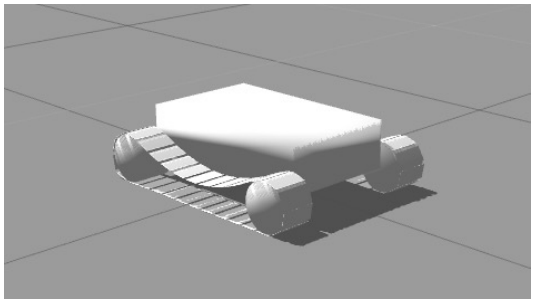
Worlds

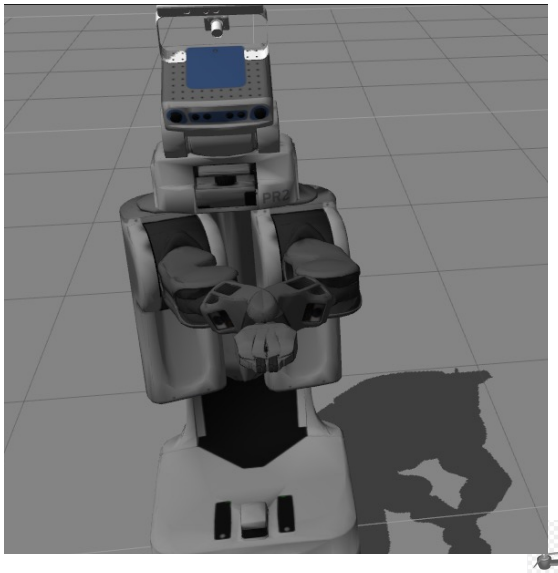
- SDF, URDF, or Xacro file describing world
- Physics Properties
- Static Models
- Lighting
- World Plugins



Models

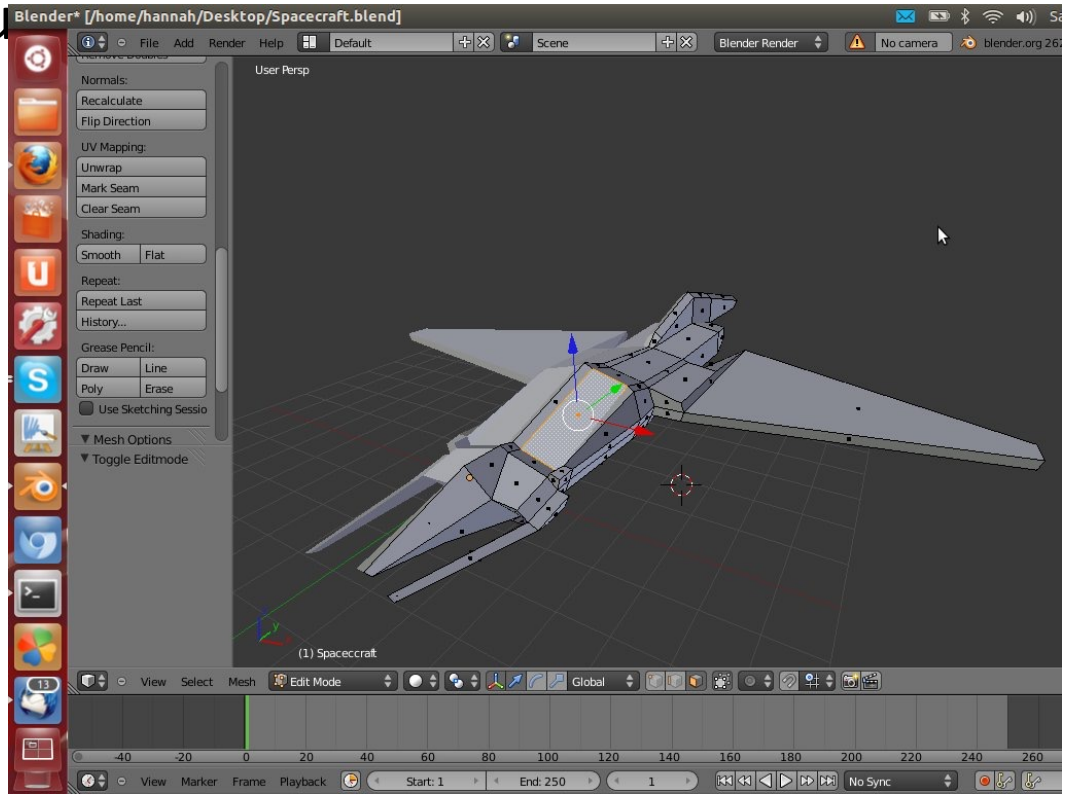
- Complete robot or any other physical object
- SDF, URDF, or Xacro file describing model
- Pose (xyz, rpy, or quaternions)
- Link
  - Inertia
  - Collision
  - Visual
- Joint
- Plugins
  - Sensors, Motor Controllers, etc





Meshes

- 3D models
  - Blender
  - Google Sketchu
  - 123 Design
  - Many others



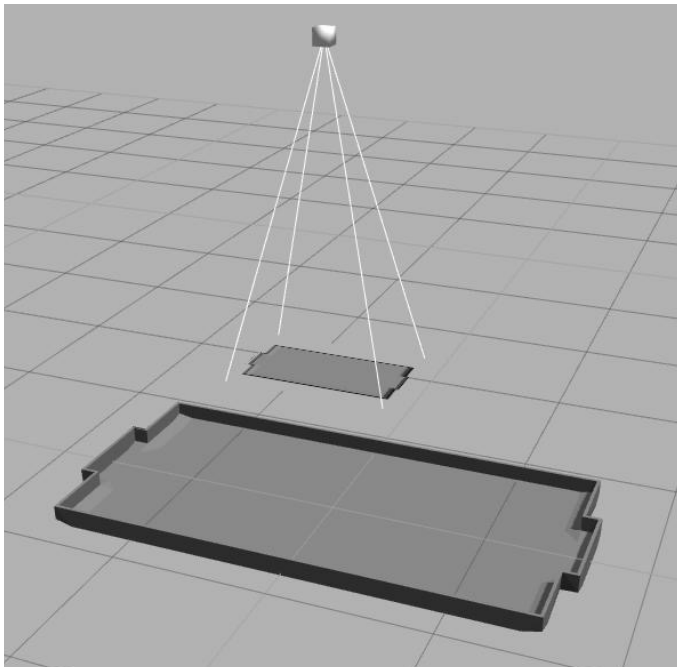
Launch Files

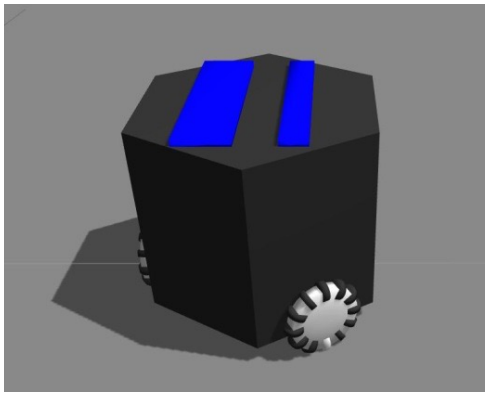
- Launch file will allow you to start everything you

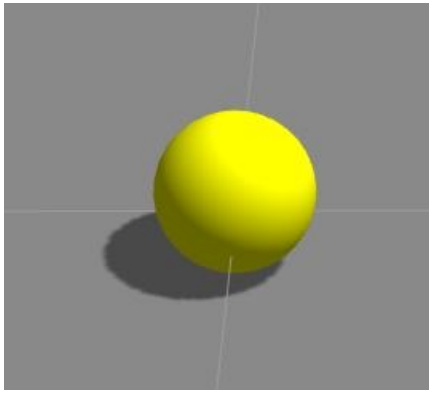
```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" />
  <node name="move_turtle" pkg="my_turtle" type="move_turtle" output="screen" />
</launch>
```


Plugins

- Sensors
  - Camera
  - Laser Scanner
- Control
  - ROS Control (PID)
  - Planar Move
  - Soccer Drive
  - Soccer Ball







Install ROS

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

ROS Kinetic Kame
Released May, 2016
LTS, supported until April, 2021
This version isn't recommended for new installs.

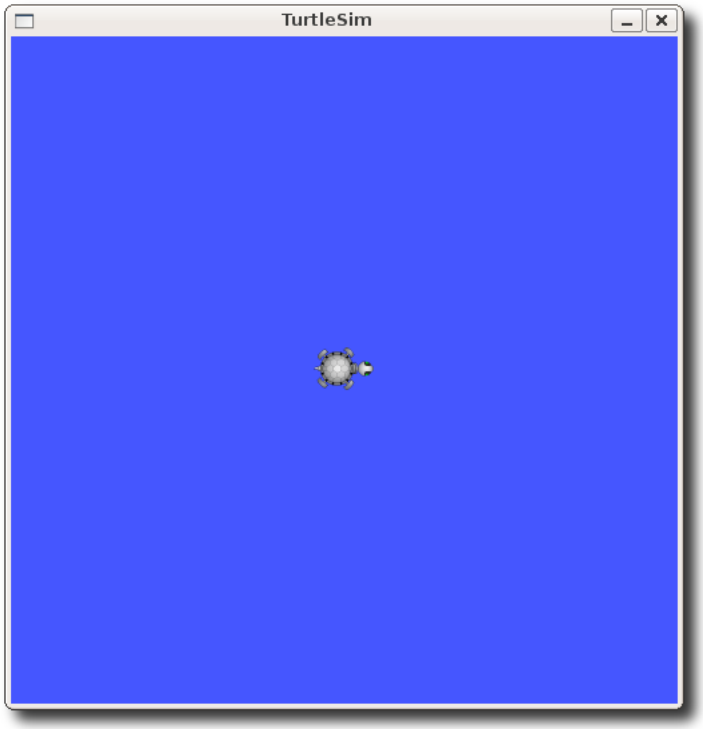
ROS Melodic Morenia
Released May, 2018
LTS, supported until May, 2023
Recommended for Ubuntu 18.04

ROS Noetic Ninjemys
Released May, 2020
Latest LTS, supported until May, 2025
Recommended for Ubuntu 20.04

- https://wiki.ros.org/noetic/Installation/Ubuntu

Testing ROS with Turtlesim

- http://wiki.ros.org/turtlesim
- Install Turtlesim
  - $ sudo apt-get install ros-$(rosversion -d)-turtlesim
- Run Turtlesim
  - $ rosrun turtlesim turtlesim_node



Turtlesim Tutorials

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

Getting Started with ROS

- Basic Tutorials
  - Understanding ROS Core Topics
  - http://wiki.ros.org/ROS/Tutorials
  - https://github.com/ros/ros_tutorials
- What to cover?
  - Navigating ROS environment
  - ROS workspace
  - Creating a ROS package from scratch
  - Using a ROS package
  - Components of a ROS package
  - Step through each tutorials in ROS wiki page