NICF - System integration with Robot Operating System (ROS) v15

SINGAPORE WORKFORCE SKILLS QUALIFICATIONS

Trainer: Man Guo Chang
Email: gc.man.sg@gmail.com
Mobile: 90403176

TINFOTECH

Website:www.tertiarycourses.com.sg
Email: enquiry@tertiaryinfotech.com

About the Trainer

Mr. Man Guo Chang graduated from Nanyang Technological University, School of Electrical and Electronic Engineering, major in Computer Engineering.

Mr. Man has more than 25 years of working experience in the Semiconductor field, specialized in IC Testing, Product Engineering, Data Analysis, and Software Development.

Mr. Man is an ACTA certified trainer. His skill set includes Website Development, Software Development, Machine Vision, Internet of Things, ROS, Cyber Security, etc.

Let's Know Each Other...

Say a bit about yourself
- Name
- What Industry you are from?
- Do you have any prior knowledge in Python and Linux?
- Why do you want to learn ROS?
- What do you want to learn from this course?

Ground Rules

- Set your mobile phone to silent mode
- Actively participate in the class. No question is stupid.
- Respect each other view
- Exit the class silently if you need to step out for phone call, toilet break

Ground Rules for Virtual Training

- Upon entering, mute your mic and turn on the video. Use a headset if you can
- Use the 'raise hand' function to indicate when you want to speak
- Participate actively. Feel free to ask questions on the chat whenever.
- Facilitators can use breakout rooms for private sessions.

WSQ and SSG TG Application Form

Please fill up the following WSQ and SSG TG Application Form for TRACOM survey, e-cert generation and WSQ funding

https://forms.gle/pJ2WxHZ3fyRbDLVu6

Digital Attendance

- You need to take digital attendance in the AM and PM.
- Please download the mySkillsFuture apps below to take your digital attendance for WSQ and SFC courses.

Guidelines for Facilitators

1. Once all the participants are in and introduce themselves
2. Goto gallery mode, take a snapshot of the class photo - makes sure capture the date and time
3. Start the video recording (only for WSQ courses)
4. Continue the class
5. Before the class end on that day, take another snapshot of the class photo - makes sure capture the date and time
6. For NRIC verification, facilitator to create breakout room for individual participant to check (only for WSQ courses)
7. Before the assessment start, take another snapshot of the class photo - makes sure capture the date and time (only for WSQ courses)
8. For Oral Questioning assessment, facilitator to create breakout room for individual participant to OQ (only for WSQ courses)
9. End the video recording and upload to cloud (only for WSQ courses)
10. Assessor to send all the assessment records, assessment plan and photo and video to the staff (only for WSQ courses).

Prerequisite

The following knowledge is assumed

- Basic Linux

Learning Outcomes

By the end of the course, learners will be able to

- conduct assessment on robotics system integration using ROS
- utilise ROS tools and protocols
- test LIDAR and SLAM with ROS
- perform robotics system integration with ROS

Agenda

Topic 1 Overview of System Integration with ROS

- Introduction to Robot Operating System (ROS)
- System Integration with ROS

Topic 2 Basic ROS Tools

- ROS Workspace and Package
- ROS Nodes and Launch
- ROS Topics and Messages
- Publishers and Subscribers Protocol
- ROS Services and Actions
- ROS Bags

Topic 3 LIDAR and SLAM

- Overview of LIDAR and SLAM
- Testing LIDAR system using ROS

Agenda

Topic 4 Robotics System Integration

- Overview ROS TF and URDF
- ROS system Integration on a robot

Final Assessment

- Written Assessment(Q&A)
- Practical Performance

Download Course Material

- Your can download the course material from Google Classroom https://classroom.google.com
- Enter the class code below to join the class on the top right.
- Goto Classwork >> Course Material
- If you cannot access the google classroom, please inform the trainer or staff.

i4nzy25

CERTIFICATE

Two e-certificates will be awarded to trainees who have demonstrated competency in the WSQ Arduino microcontroller assessment and achieved at least 75% attendance.

- A SkillsFuture WSQ Statement of Attainment (SOA) ICT-DIT-3016-1.1 System Integration TSC under the ICT Skills Framework issued by WSG
- Certification of Completion issued by Tertiary Infotech Pte Ltd

Topic 1
Overview of System Integration with ROS

What is ROS

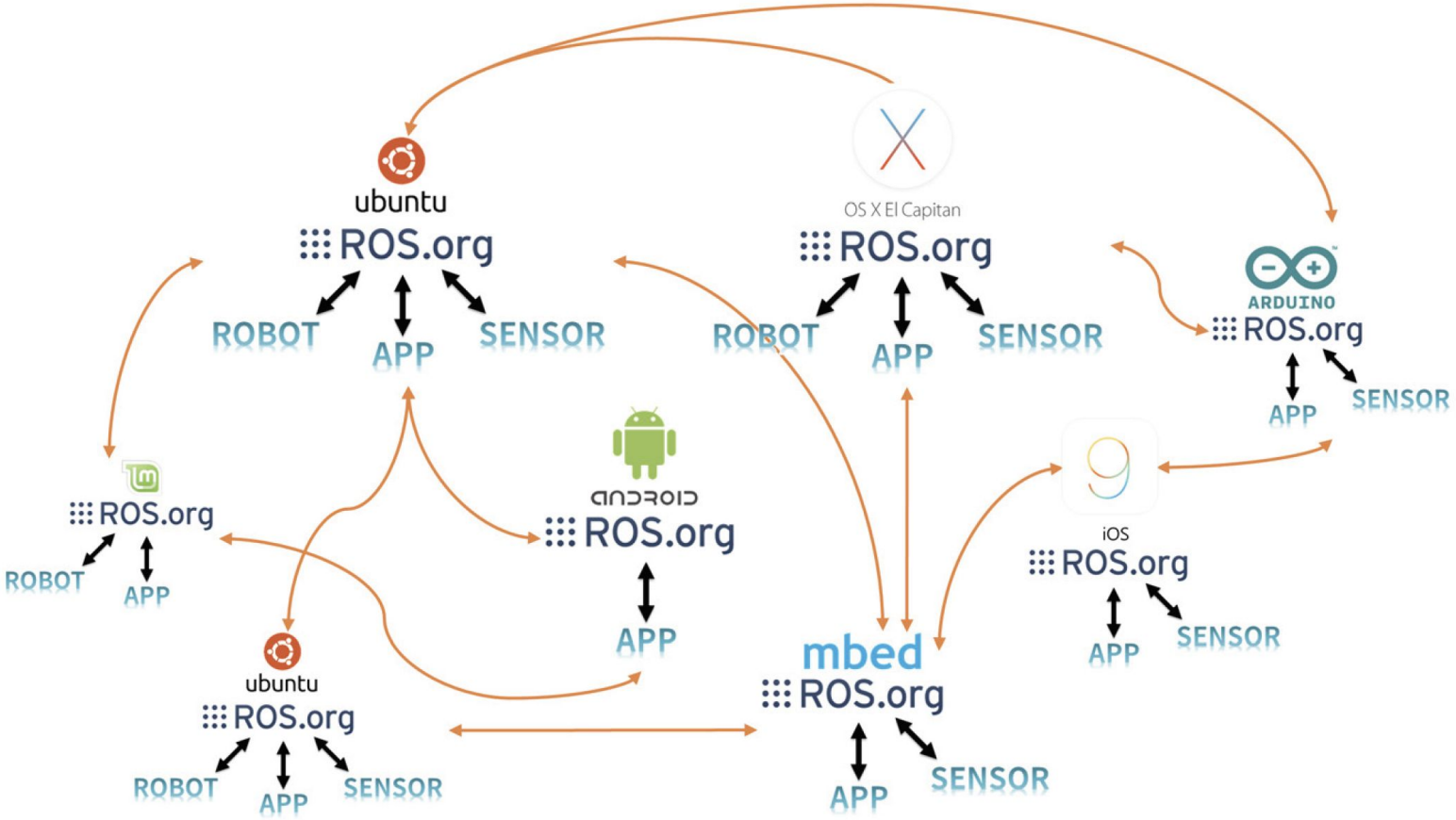
- ROS = Robot Operating System
- ROS is an open-source, meta-operating system for your robot.
- It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.
- It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers

What is Meta Operating System?

- ROS is NOT a conventional OS; ROS is a meta operating system that runs on existing OS such as Window, Mac, Linux etc.

ROS Data Communication

- ROS data communication is supported by multiple operating systems, hardware, and programs, making it highly suitable for robot development where various hardware are combined



ROS Robots

There are hundreds of ROS robots in the market.
https://robots.ros.org/all/

10 Years of ROS

Team MAXed Out

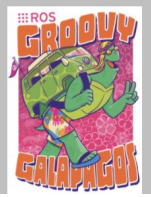
Learning Algorithms and Systems Laboratory (LASA) | EPFL

History of ROS

- 2000's - Robotics projects started at Stanford AI Lab
- 2007 - In May 2007, ROS was started by borrowing the early open source robotic software frameworks including switchyard, which is developed by Dr. Morgan Quigley by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR (STanford AI Robot) project
- 2007 - In November 2007, U.S. robot company Willow Garage succeeded the development of ROS. Willow Garage is a well-known company in the field for personal robots and service robots. It is also famous for developing and supporting the Point Cloud Library (PCL), which is widely used for 3D devices such as Kinect and the image processing open source library OpenCV.
- 2010 - On January 22, 2010, ROS 1.0 was released
- 2013 - Open Source Robotics Foundation (OSRF) later changed to Open Robotics
- 2013 - Release of Kinetic Kame
- 2014 - Release of Indigo Igloo
- 2017 - Release of Lunar Loggerhead
- 2017 - Release of ROS 2.0
- 2018 - Release of Melodic

ROS Distributions

http://wiki.ros.org/Distributions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014

ROS Philosophy

- **Peer to peer**
  Individual programs communicate over defined API (ROS messages, services, etc.).
- **Distributed**
  Programs can be run on multiple computers and communicate over the network.
- **Multi-lingual**
  ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Light-weight**
  Stand-alone libraries are wrapped around with a thin ROS layer.
- **Free and open-source**
  Most ROS software is open-source and free to use.

ROS Components

ROS Ecosystem

- More than 5,000 packages
- 2,818 Official Packages (Indigo, March 2017)
- 13,441,711 .deb Packages Download (July 2017)
- 18,839 Wiki page (July 2017)

Developers, Users

APP

ROS

ROBOT, SENSOR

Robot, Sensor Manufacturer

Applications	fetch beer, elevator ···
Simulators	gazebo, player/stage, STDR Simulator ···
Intelligent Modules	navigation, action, grasping ···
Libraries	tf, PCL, OpenCV, OpenRave ···
Device Drivers	camera_drivers, urg_node ···
Debugging Tools	rviz, rqt_graph, rosbag, rostopic ···
Message Passing	rosmaster, rosmsg, rosservice ···
Execution Tools	rosrun, roslaunch ···
Compile Tools	catkin_make, rosbuild ···
File Systems	roscd, rosls ···
Installer	rosinstall ···
Programming Languages	C++, Python, Lisp, Java, Ruby, MATLAB, etc

- Supports more than 90 robots and 80 sensors

Install ROS

- To install ROS Kinetic distribution on Ubuntu 16.04:

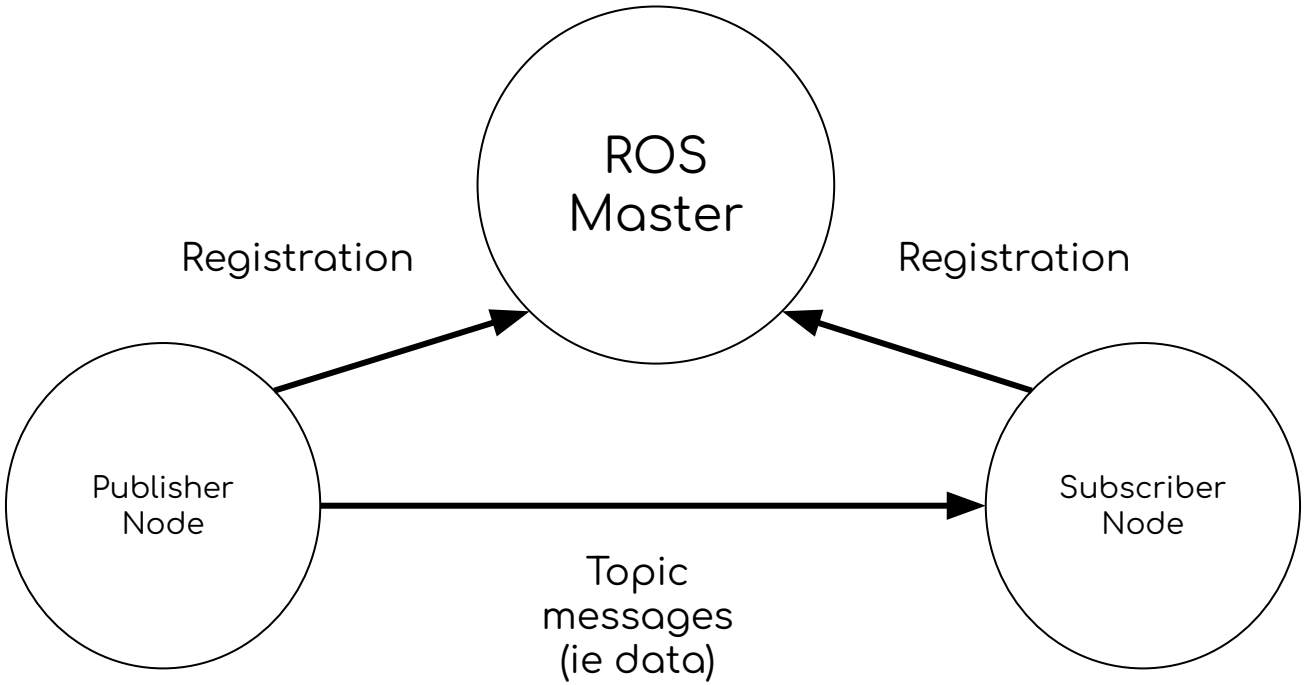
  https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

- To install ROS Kinetic distribution on VirtualBox

  - Download and install VirtualBox from https://www.virtualbox.org
  - Download Ubuntu 16.04 image from http://releases.ubuntu.com/16.04/?_ga=2.245247622.286617492.1585822024-1634746839.1585822024.
  - In VirtualBox, click "New" to install Ubuntu
  - Install ROS kinetic distribution: https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

ROS Master

- The ROS Master provides naming and registration services to all other nodes in the ROS system.
- It tracks publishers and subscribers to topics and services.
- It enables individual nodes to locate one another.
- Once these nodes have located each other, they communicate with each other peer-to-peer.



Start a ROS Master

- To start a ROS master, type on terminal:

  roscore

- roscore will start up:
  - a ROS Master
  - a ROS Parameter Server
  - a rosout logging node

- roscore is the command that runs the ROS master. If multiple computers are within the same network, it can be run from another computer in the network. However, except for special case that supports multiple roscore, only one roscore should be running in the network.

Topic 2
Basic ROS Tools

ROS Workspace & Catkin

- Any ROS project begins with making a workspace.
- catkin is the official build system of ROS to generate executables, libraries, and interfaces.
- catkin is the successor to the original ROS build system, rosbuild.
- catkin combines CMake macros and Python scripts to provide some functionality on top of CMake's normal workflow

Create a Workspace

- To create a catkin workspace, type from the following command lines

  mkdir -p catkin_ws/src

  cd catkin_ws

  catkin_make

- After catkin_make, there are two 'build' and 'devel' folders created
- To make the workspace visible to ROS, run the following command
  - cd ~/catkin_ws
  - source devel/setup.bash

Catkin Build System

catkin_make will create the following folders in the workspace:

Work here

The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

The development (devel) space is where built targets are placed (prior to being installed).

Activity: Create a New Workspace

- Create a new workspace on the root directory

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws
$ catkin_make
$ source ~/.bashrc
```

OR (after creating the workspace)

```
$ cm
$ sb
```

Workspace Folder Structure

A typical workspace folder structure might look like this:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

ROS Package

- A ROS package is a project. It 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.
- A ROS package is resided in the src folder of the ROS workspace.
- For a ROS package to be considered a catkin package it must meet a few requirements:
  - The package must contain a catkin compliant package.xml file. The package must contain a CMakeLists.txt which uses catkin.
  - Each package must have its own folder

Steps to Create a ROS Package

- Change to the source space directory of the catkin workspace:
  $ cd ~/catkin_ws/src

- Use the catkin_create_pkg script to create a new package
  catkin_create_pkg <package_name> [depend1], [depend2],....
  $ catkin_create_pkg demo1 std_msgs rospy roscpp

- Build the packages in the catkin workspace
  $ cd ~/catkin_ws
  $ catkin_make

- Add the workspace to your ROS environment by sourcing the generated setup file:
  $. ~/catkin_ws/devel/setup.bash

- catkin_make creates two files - CMakeList.txt and package.xml and two folders include and src in the package folder

Activity: Create a ROS Package

Create a ROS package 'my_robotics' as follows:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg my_robotics std_msgs rospy roscpp
$ cd ~/catkin_ws
$ catkin_make
$ .  ~/catkin_ws/devel/setup.bash
```

OR

```
$ cs
$ catkin_create_pkg my_robotics std_msgs rospy roscpp
$ cm
$ sb
```

ROS Packages File Structure

ROS packages tend to follow a common structure:

- **src:** Source files (C++)
- **scripts:** Script files (Python)
- **msg:** Folder containing Message (msg) types
- **srv:** Folder containing Service (srv) types
- **CMakeLists.txt:** CMake build file
- **package.xml:** Package catkin/package.xml
- **include:** C++ include headers

Install a ROS Package

- You can enhance the capabilities of ROS by installing packages
- These packages are either downloadable debian packages or github repo
- To install debian packages, use apt-get install, Eg

  sudo apt install <package-name>
- To install github repo, cd to src folder, do a git clone, follow by catkin build and source setup.bash

Activity: Install Github ROS Package

Try to install a github repo for ROS package as follows:

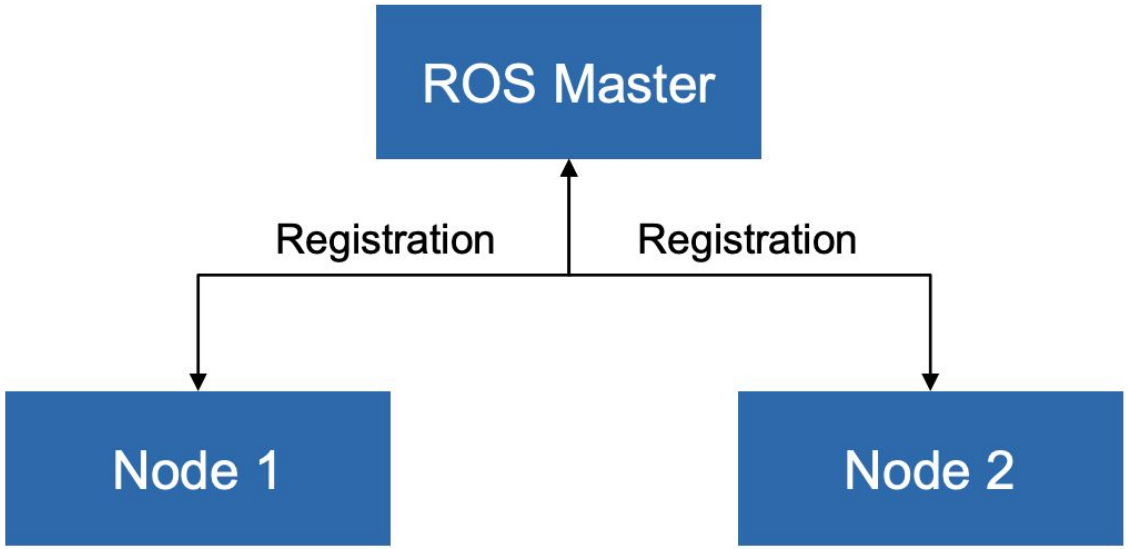
$ cd ~/<your own ws>/src
$ git clone
https://github.com/tertiarycourses/beginner_tutorials
$ cd ~/<your own ws>
$ catkin_make
$. devel/setup.bash

OR

$ cs
$ git clone
https://github.com/tertiarycourses/beginner_tutorials
$ cm
$ sb

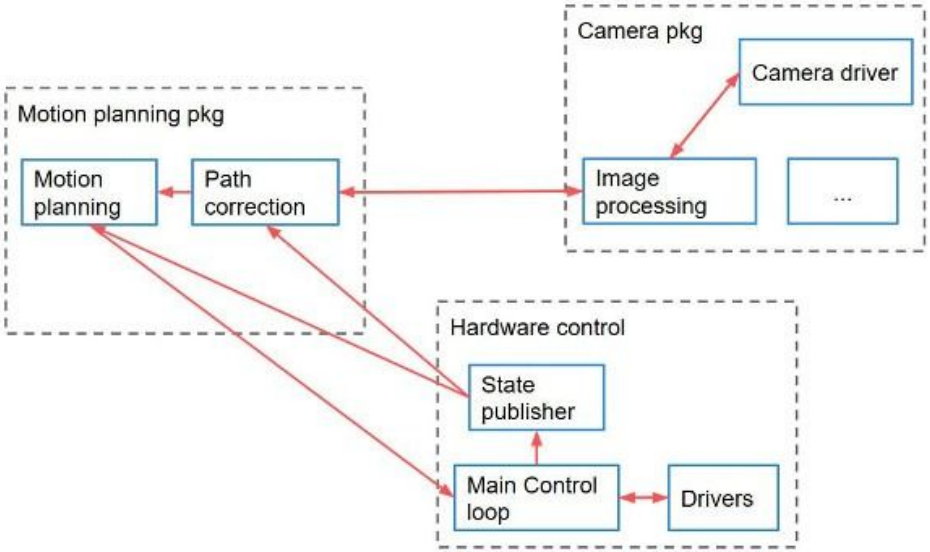
ROS Nodes

- The basis of ROS communication is that multiple executables called nodes are running in an environment and communicating with each other in various ways.
- A node is a single purpose executable program
- These nodes exist within a structure called a package



ROS Node Case Study

- Take an example of a standard robotics application which involves a mobile robot and a camera. The robot has 3 ROS packages :
  - Hardware control package: directly controls the hardware (wheels and other actuators)
  - Motion planning package: monitors and controls the robot trajectory
  - Camera package: processes images and give useful info and commands to the robot
- Each package would have multiple nodes for some processes such as image process, path correction.
- Nodes are combined into a graph and communicate with each other using ROS topics, services, actions, etc.



Start a Node

- To start a node, type the following command
  rosrun <package_name> <node_name>
  Eg: **rosrun turtlesim turtlesim_node**

Node Information

- To list all the active nodes,run
  rosnode list
- To retrieve information about a node with
  rosnode info <node_name>
  Eg: **rosnode info /turtlesim**

Activity: ROS Node

Run the following commands

(terminal 1) $ roscore

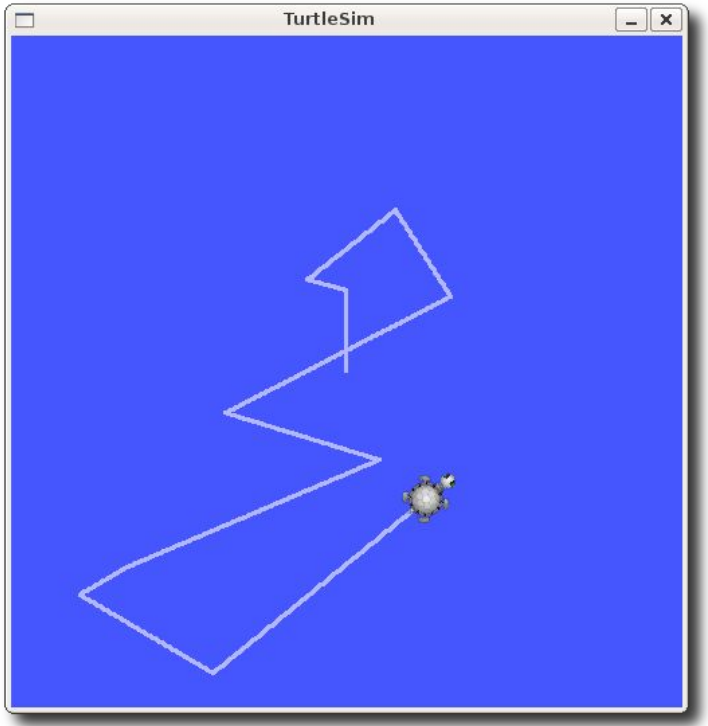
(terminal 2) $ rosrun turtlesim turtlesim_node

(terminal 3) $ rosrun turtlesim turtle_teleop_key

Use the arrow keys to move the turtle

(terminal 4) $ rosnode list

(terminal 4) $ rosnode info /teleop_turtle



ROS Launch

- ROS launch is a tool for launching multiple nodes (as well as setting parameters)
- Launch files are written in XML as *.launch files
- To launch the ROS nodes
  roslaunch <package_name> <launch file name>

Activity: ROS Launch

(terminal 1)
$ cs
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cm
$ sb

(terminal 1)
$ roscore

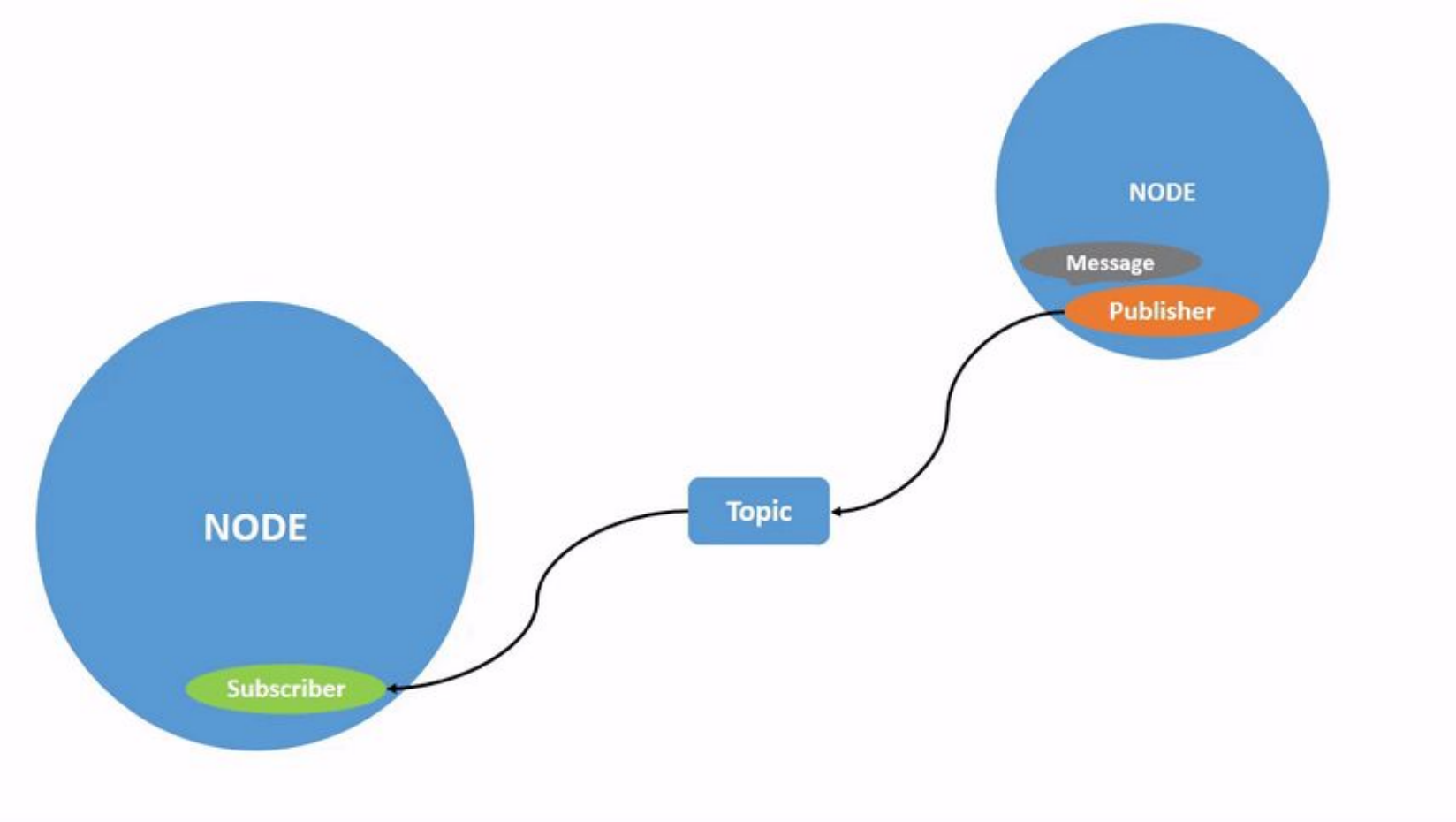
(terminal 2)
$ sb
$ roslaunch turtlebot3_fake turtlebot3_fake.launch

(terminal 3)
$ sb
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

(terminal 4)
$ sb
$ rosnode list

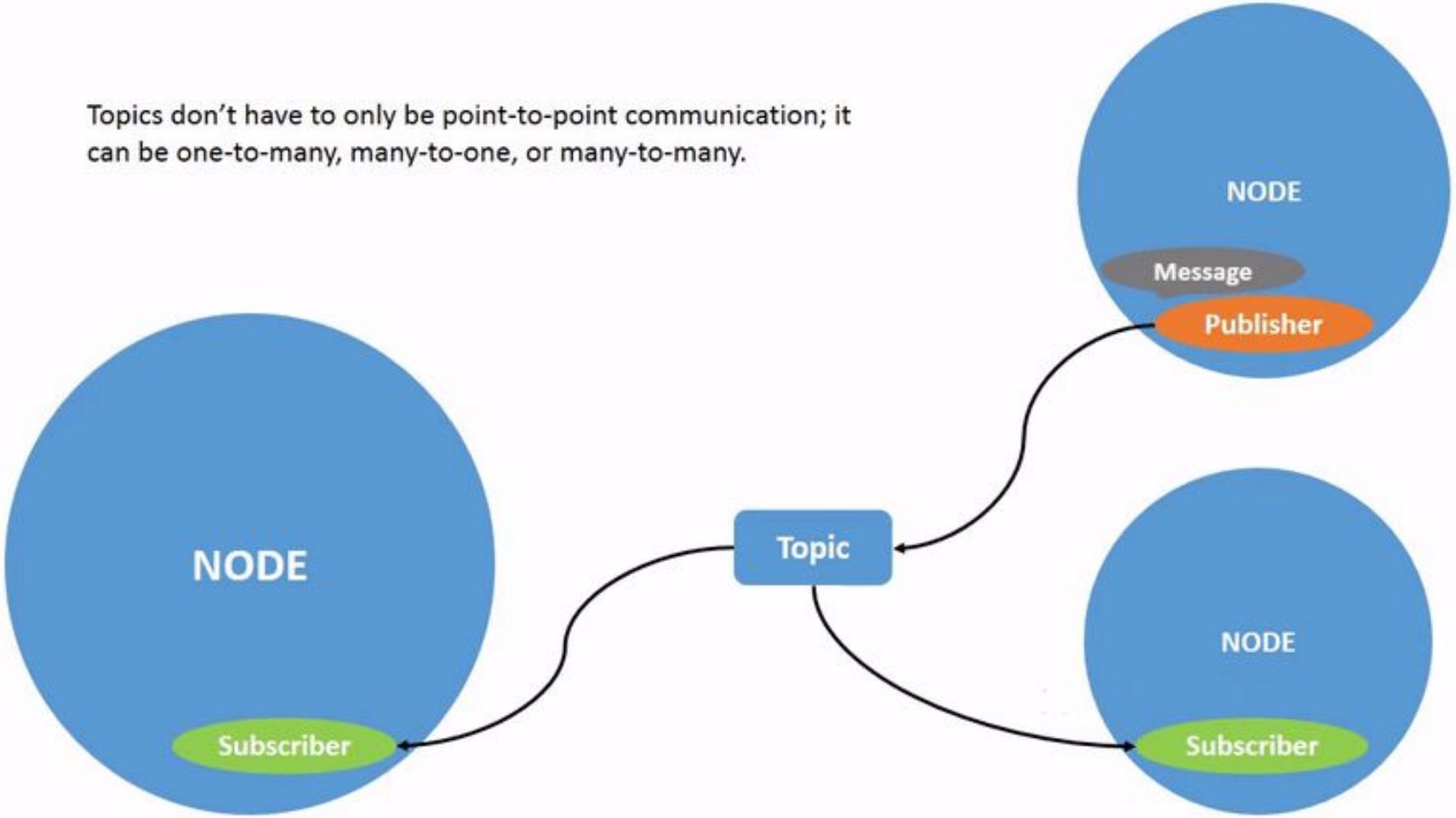
ROS Topics

- A ROS Topic is a channel for communicating messages among the nodes
- A ROS Topic has a fixed Message type
- A Node publishes on a Topic to broadcast Messages
- A Node subscribes to a Topic to get Messages from other Nodes

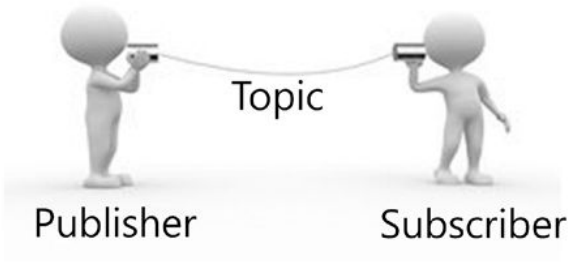


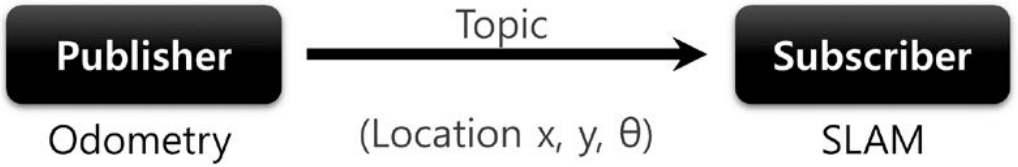
Topics Many-to-Many Communication

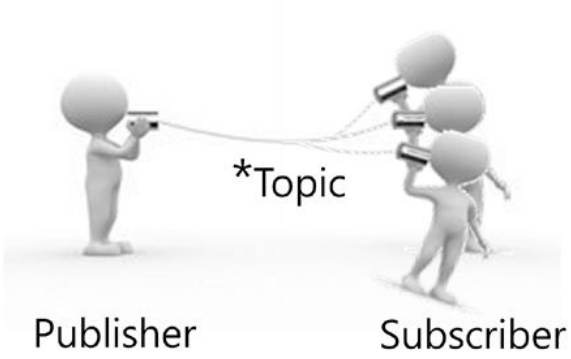
Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.

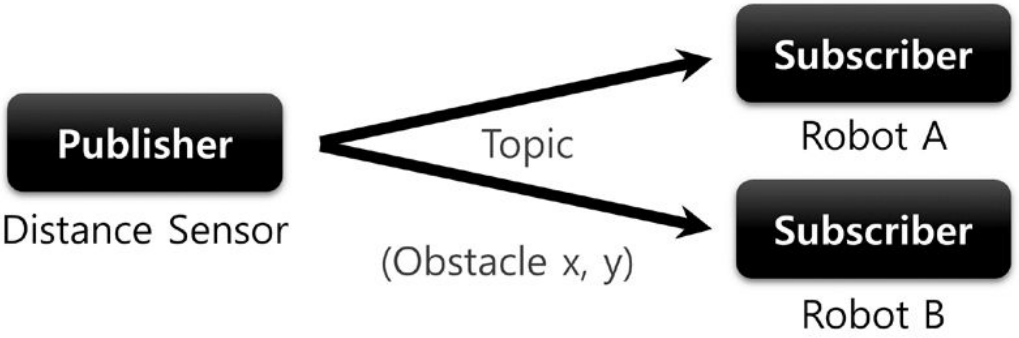


Topic Message Communication









*Topic not only allows 1:1 Publisher and Subscriber communication, but also supports 1:N, N:1 and N:N depending on the purpose.

Topics information

- To list all the active topics, run
  rostopic list
- To show the information on a topic
  rostopic info <topic_name>
- To see the data being published on a topic, use:
- **rostopic echo <topic_name>**

Eg
rostopic info /turtle1/cmd_vel

Activity: ROS Topic

Run the following commands

(terminal 1) $ roscore

(terminal 2) $ rosrun turtlesim turtlesim_node

(terminal 3) $ rosrun turtlesim turtle_teleop_key

Use the arrow keys to move the turtle

(terminal 4)

$ rostopic list

$ rostopic info /turtle1/cmd_vel

$ rostopic echo /turtle1/cmd_vel

Visualize Nodes and Topics

- To visualize the nodes and topics relationship, you can use the package rqt_graph
  rosrun rqt_graph rqt_graph

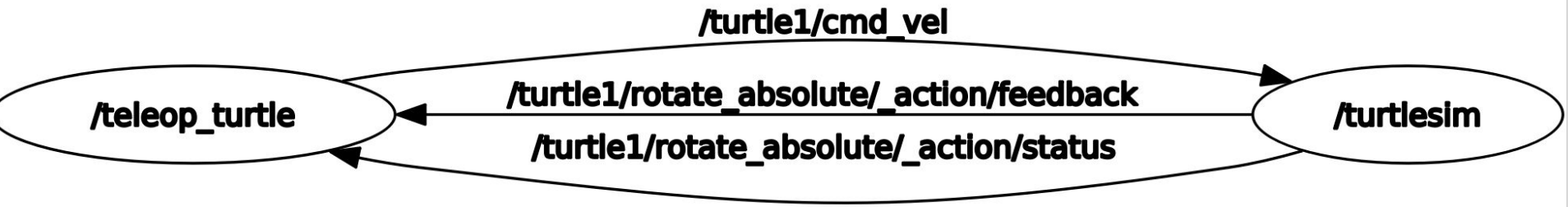
As you can see, the turtlesim_node and the turtle_teleop_key nodes are communicating on the topic named /turtle1/command_velocity.

RQt

- RQt is a graphical user interface framework that implements various tools and interfaces in the form of plugins.
- One can run all the existing GUI tools as dockable windows within RQt! The tools can still run in a traditional standalone method, but RQt makes it easier to manage all the various windows in a single screen layout.
- You can run any RQt tools/plugins easily by:
  rqt

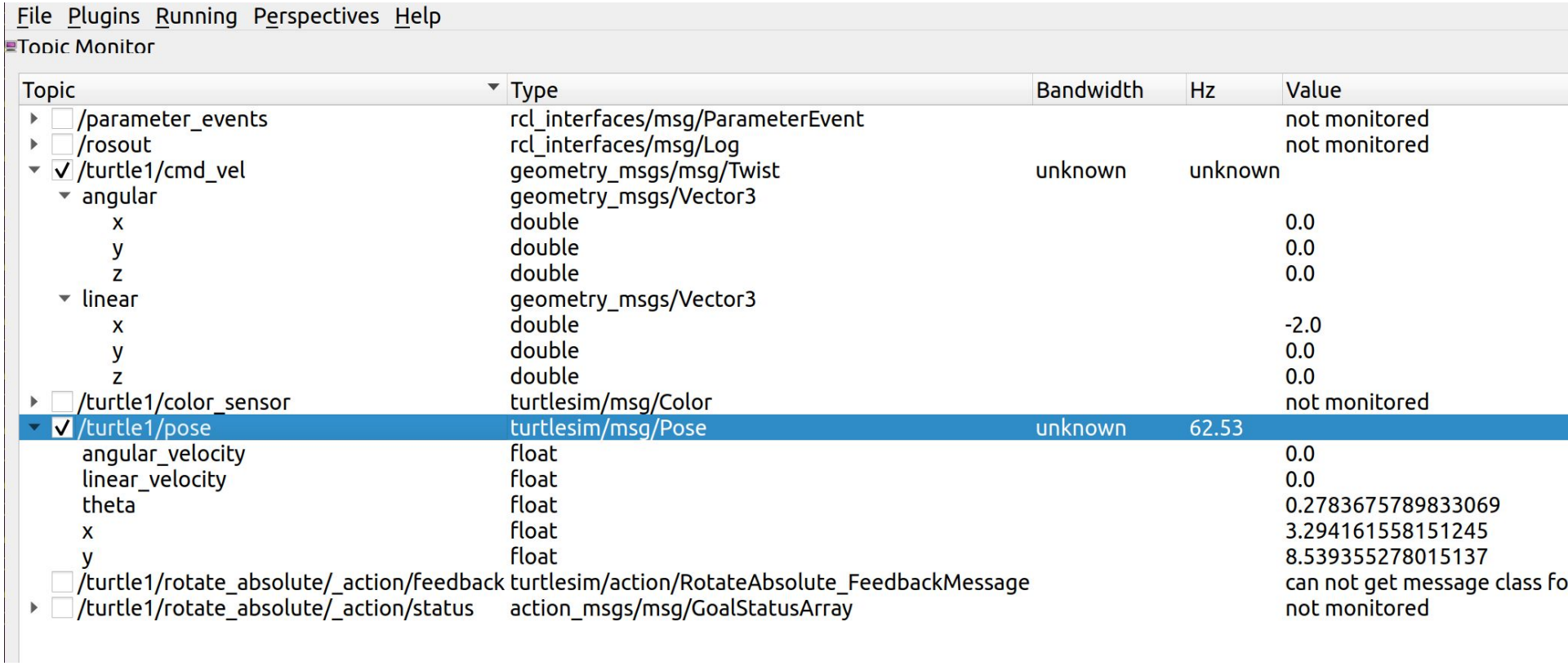
Visualize Nodes and Topics

- To visualize the nodes and topics relationship, you can run rqt and selecting Plugins > Introspection > Nodes Graph.
- As you can see, the turtlesim_node and the turtle_teleop_key nodes are communicating on the topic named /turtle1/cmd_vel



Activity: Monitor Topic with RQt

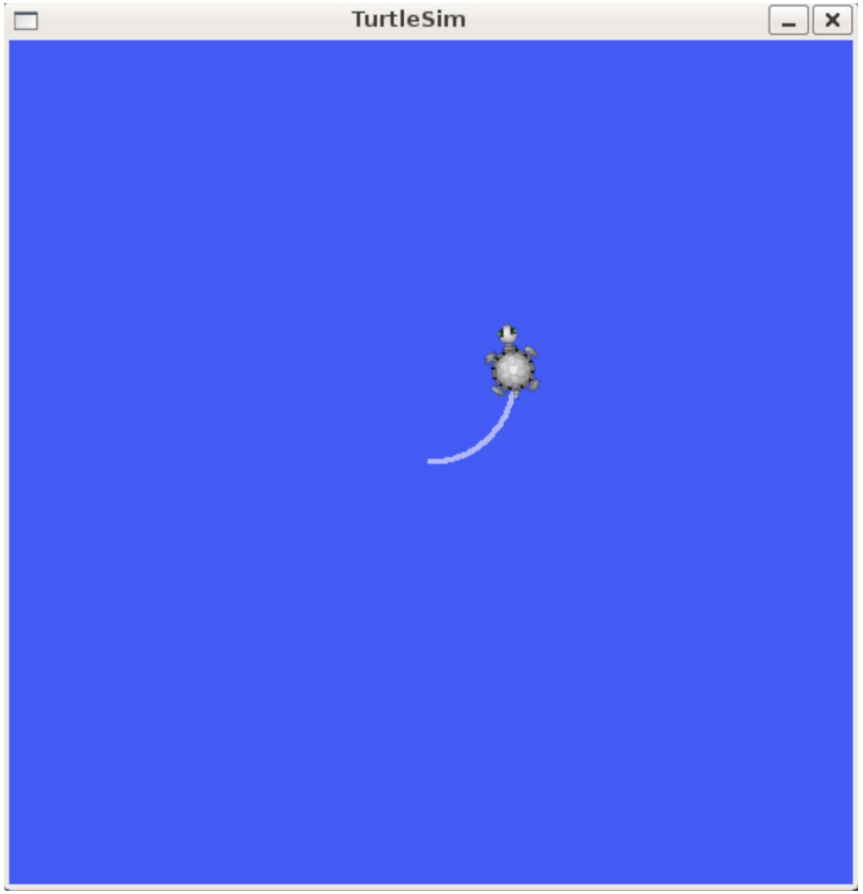
- You can open rqt and select Plugins > Topics > Topics Monitor to monitor the topics information.
- Move the keys to see how the value changes.



Publish ROS Message Data

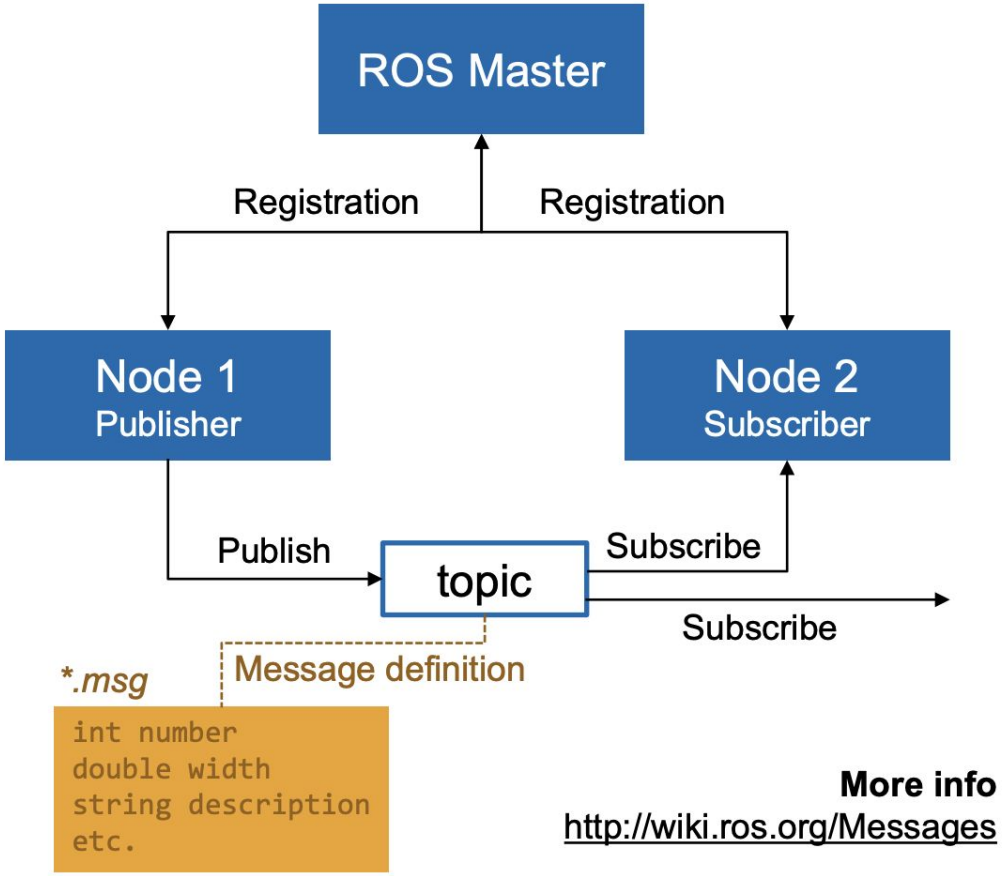
rostopic pub publishes data on to a topic currently advertised. Eg

rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'



ROS Messages

- Communication on topics happens by sending ROS messages between nodes.
- A ROS message is a data type for communication among the nodes



ROS Message Types

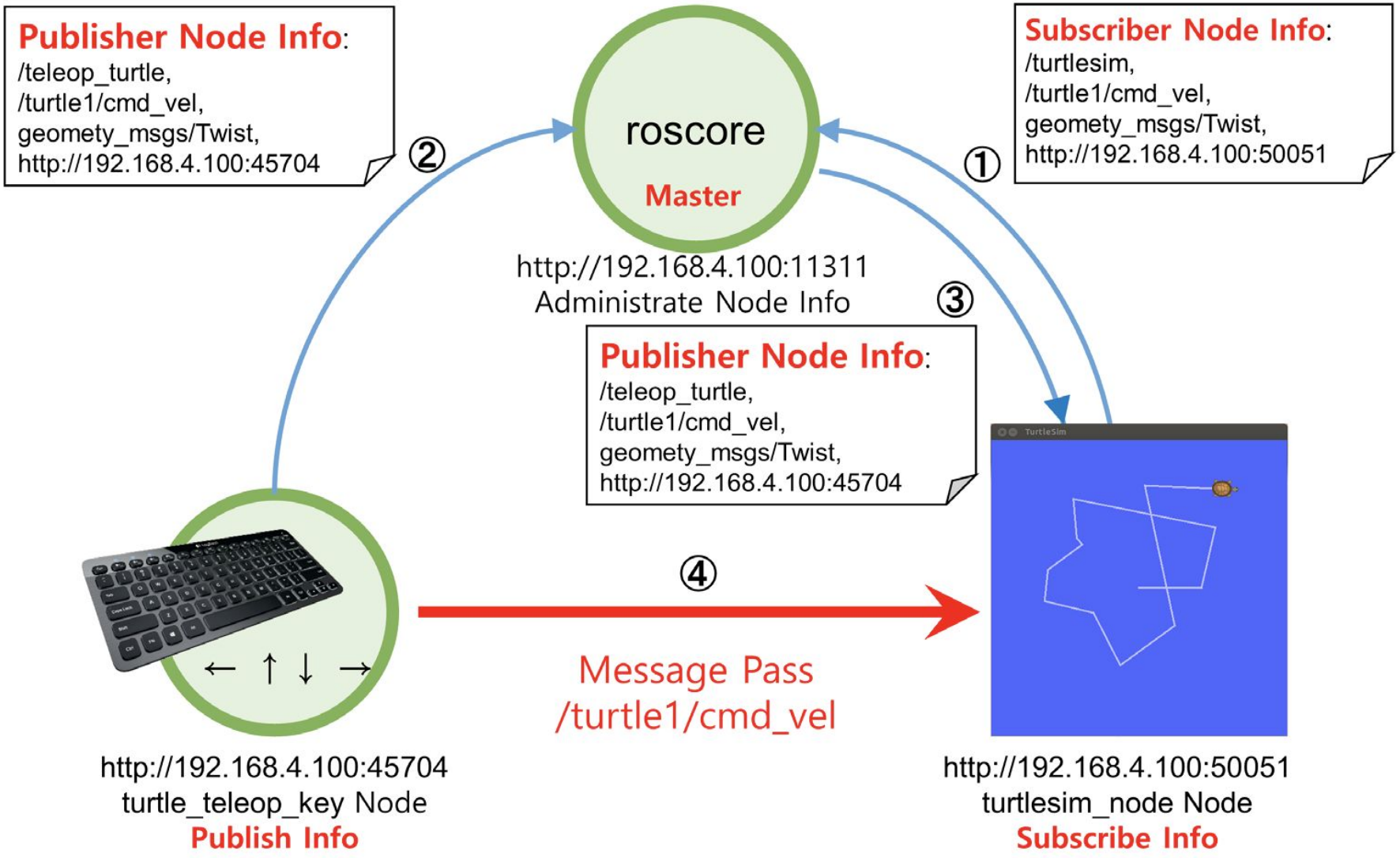
- ROS built in default message types
  - std_msgs - basic data types, such as Int , Float, Char, String, Bool, Byte, Array
  - geometry_msgs - geometry related data types, such as
    - Pose-related: Point, PointStamped, Quaternion, QuaternionStamped, Pose PostStamped, PosewithCovariance, PosewithCovarianceStamped, PoseArray
    - Control-related: Twist, Vector3, Transform
  - sensor_msgs - sensory data types, such as Image, PointCloud, BatteryState, LaserScan
- Customized message type
  - Developer can define a new message type using the existing message types

Show Message Types

- To show information of a message
  rosmsg show <type>
- Eg **rosmsg show geometry_msgs/Twist**
  Its show the following info:

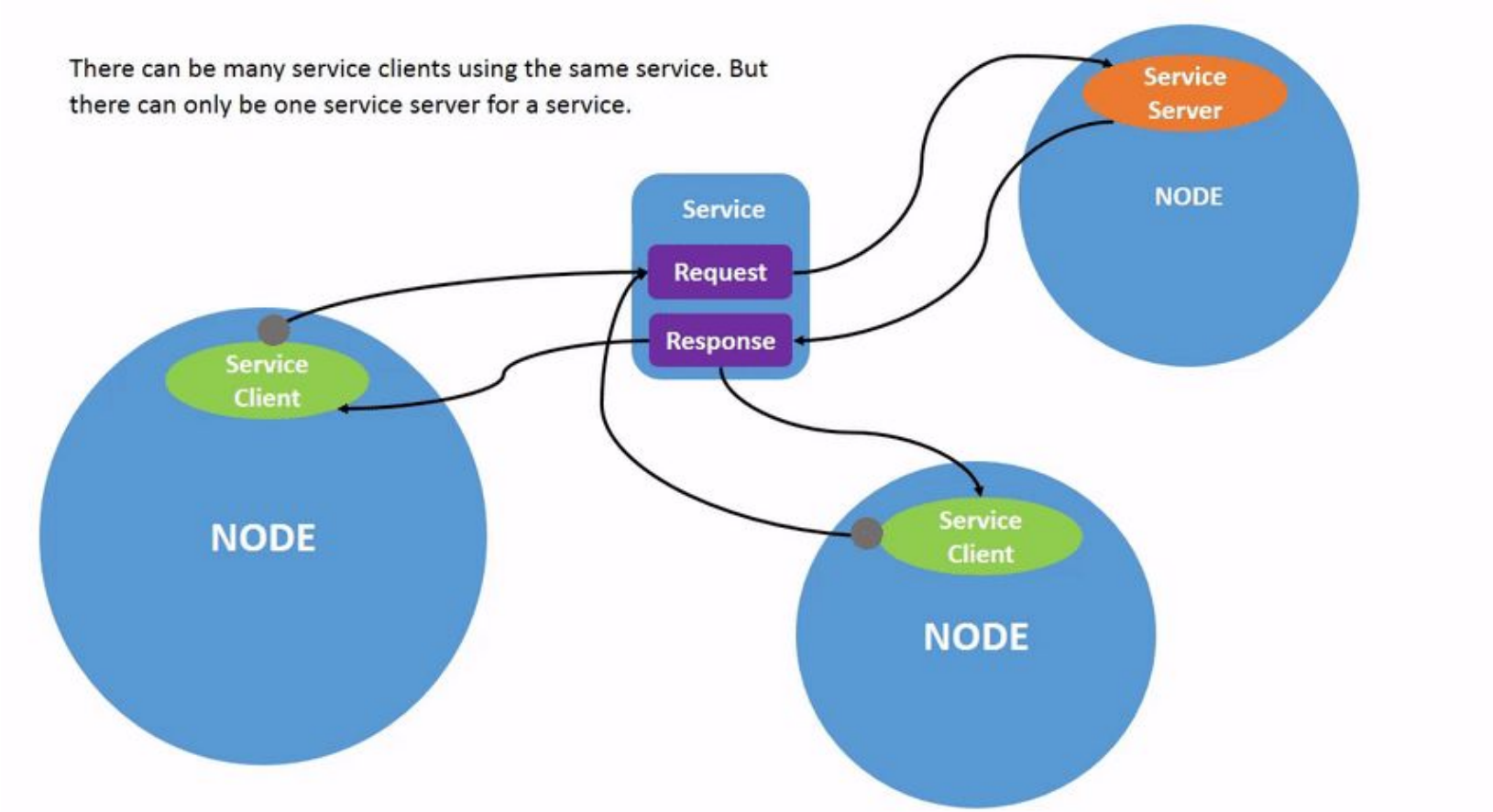
```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Turtlesim Message Communication

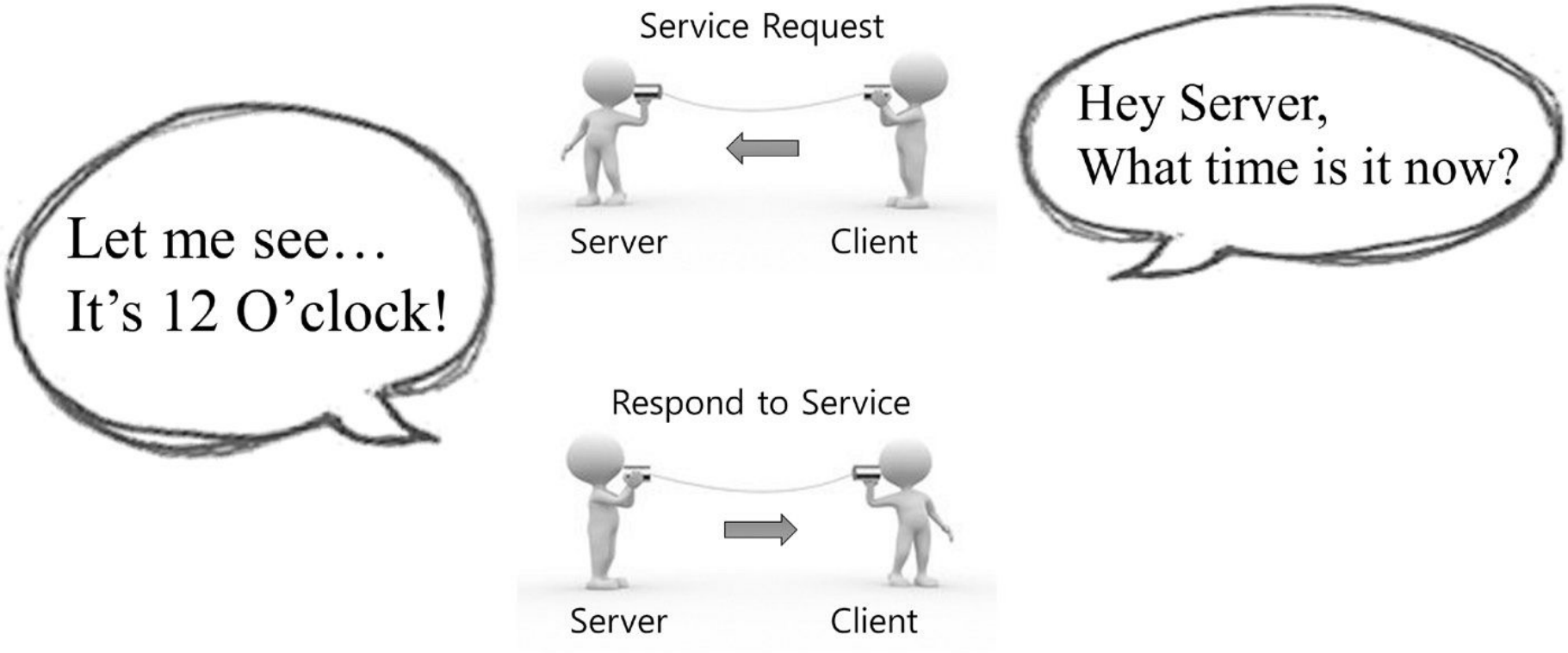


ROS Services

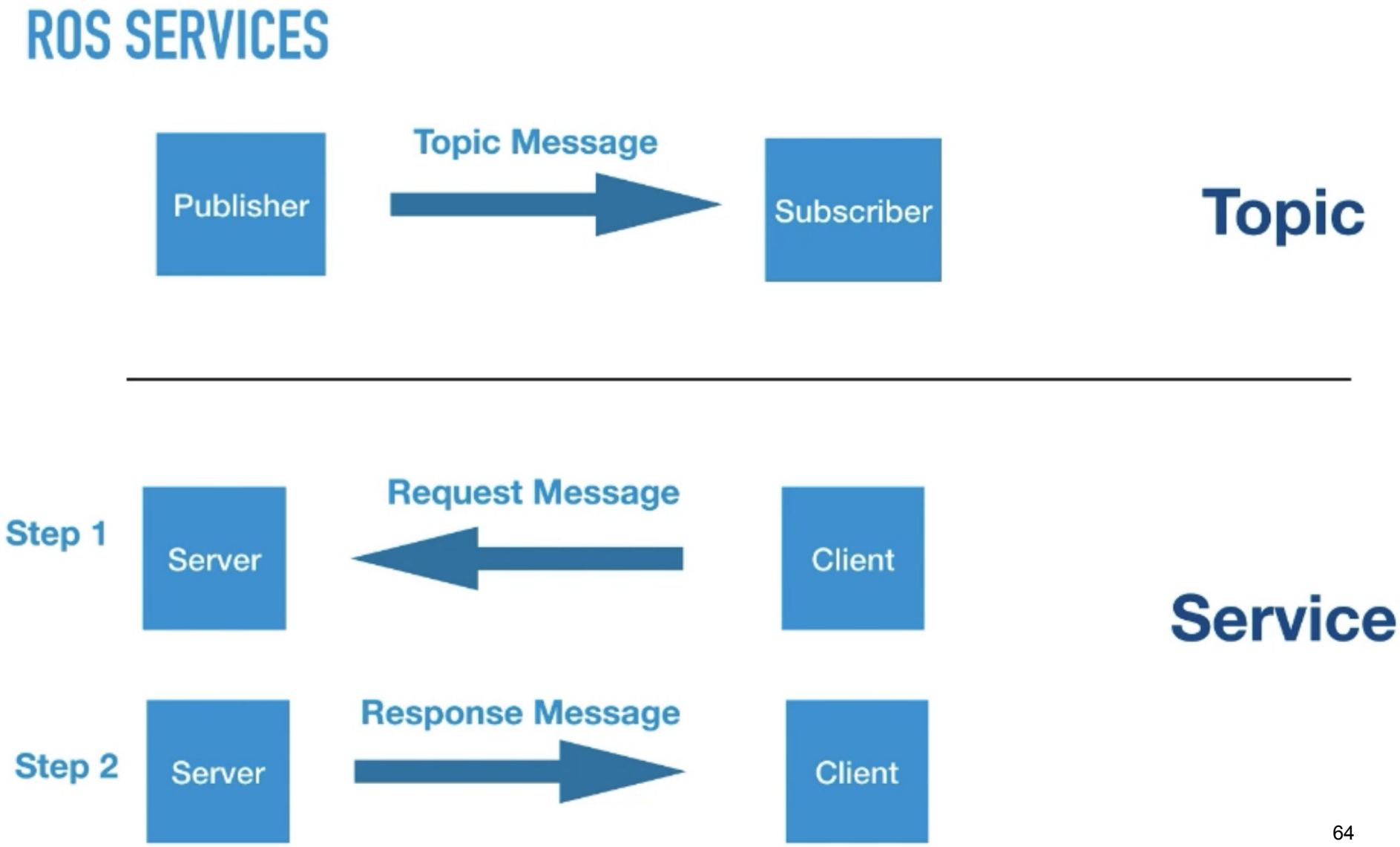
- What is ROS Service?
  - Unlike topic, service is a one time communication
  - A client sends a request, the server sends back a response.
- When to use ROS Service?
  - Request the robot to perform a certain task eg pick up a cup, move from point A to B



Service Message Communication

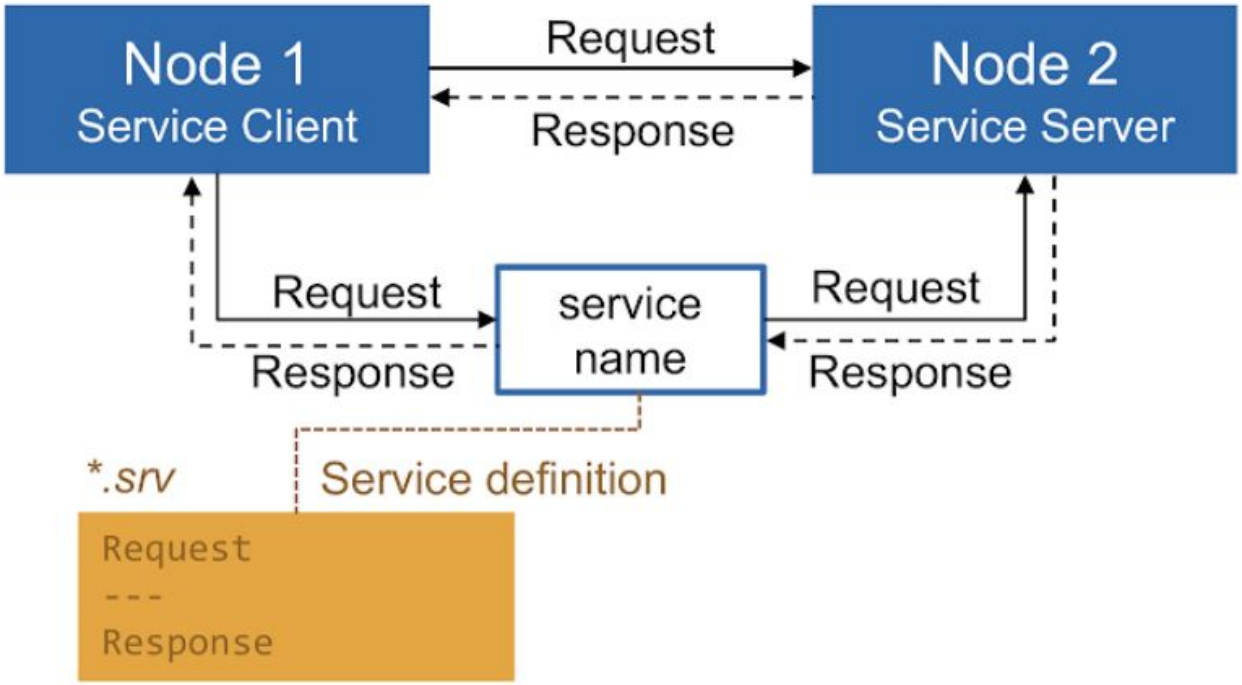


ROS Topic vs Service



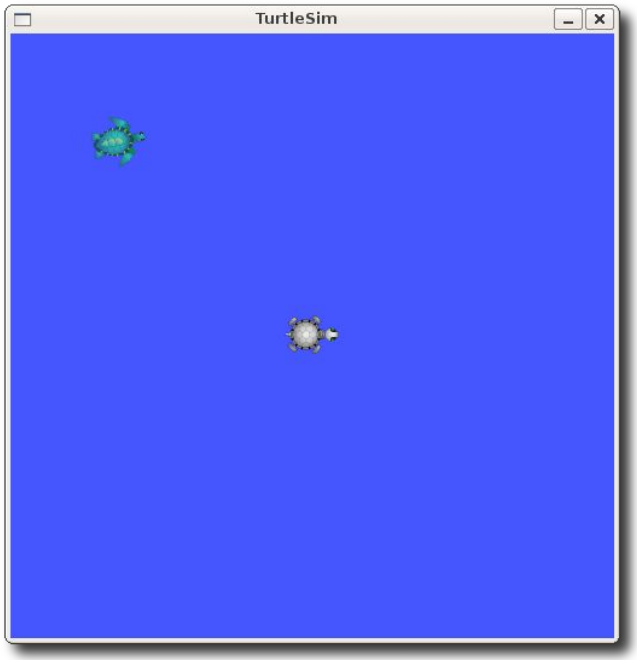
ROS Service Command

- List all available ROS services
  rosservice list
- Show the type of ROS service
  rosservice type /service_name
- Call a service with request
  rosservice call /service_name args



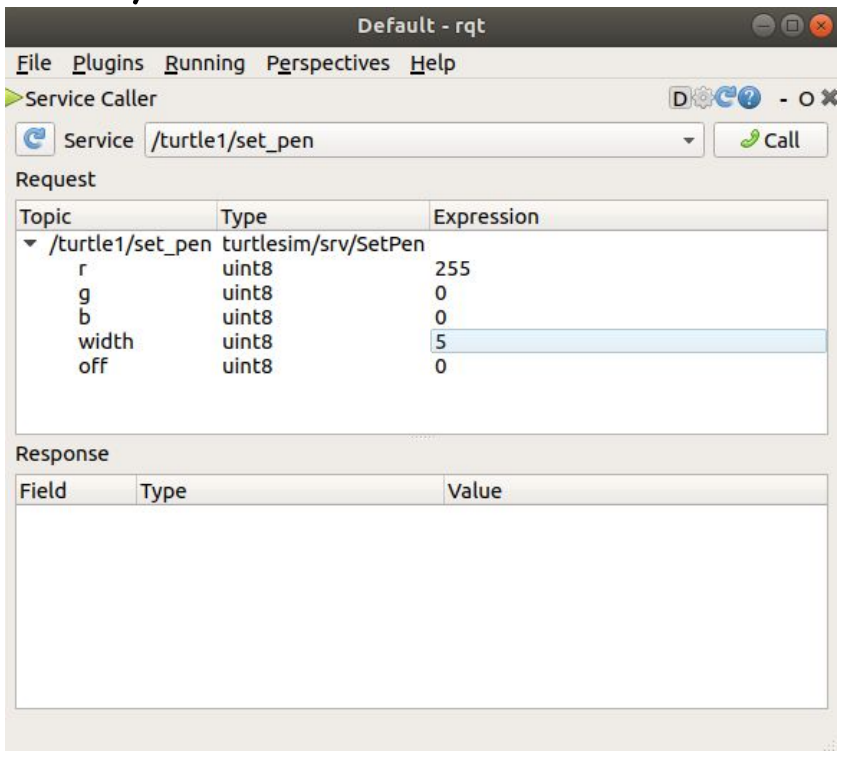
Activity: ROS Service

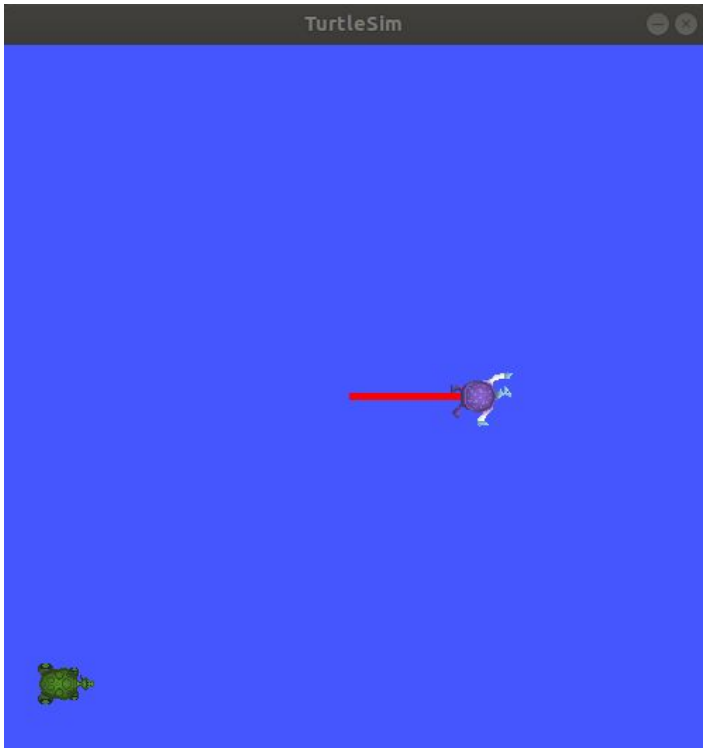
Run the following commands
(terminal 1) $ roscore
(terminal 2) $ rosrun turtlesim turtlesim_node
(terminal 3)
$ rosservice list
$ rosservice call /spawn 2 2 180 t2
$ rosservice call /spawn 8 8 0 t3



Call ROS Services from RQt

- Select Plugins > Services > Service Caller from the menu bar at the top.
- Use rqt to call the /spawn service.
- Give turtle1 a unique pen using the /set_pen service
- To have turtle1 draw with a distinct red line, change the value of r to 255, and the value of width to 50 and y = 1.0





ROS Parameters

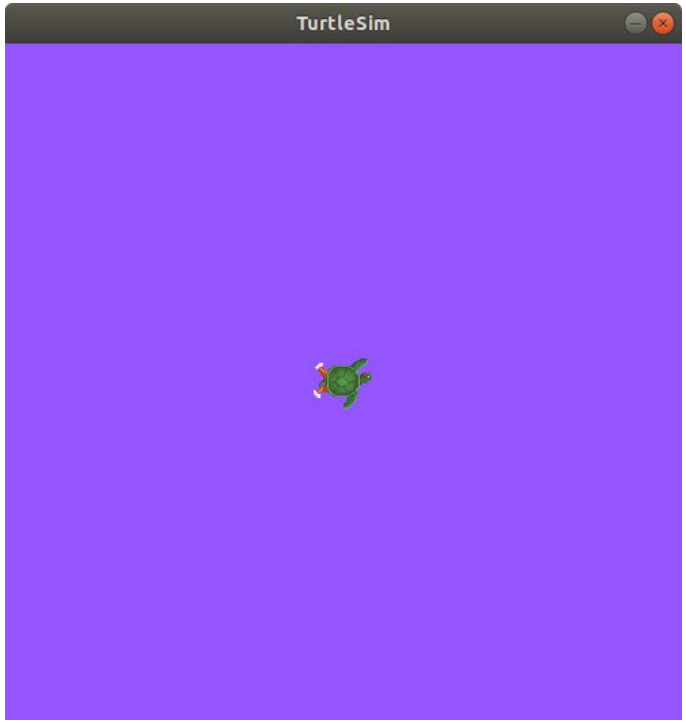
- A parameter server is a shared, multi-variate dictionary that is accessible via network APIs.
- Nodes use this server to store and retrieve parameters at runtime.
- As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters.
- It is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify if necessary.
- The Parameter Server is implemented using XMLRPC and runs inside of the ROS Master, which means that its API is accessible via normal XMLRPC libraries

ROS Parameter Commands

- To see the parameters belonging to your nodes, open a new terminal and enter the command:
  rosparam list
- To get the current value of a parameter, use the command:
  rosparam get <node_name> <parameter_name>
- To change a parameter's value at runtime, use the command:
  rosparam set <node_name> <parameter_name> <value>

Activity: ROS Parameters

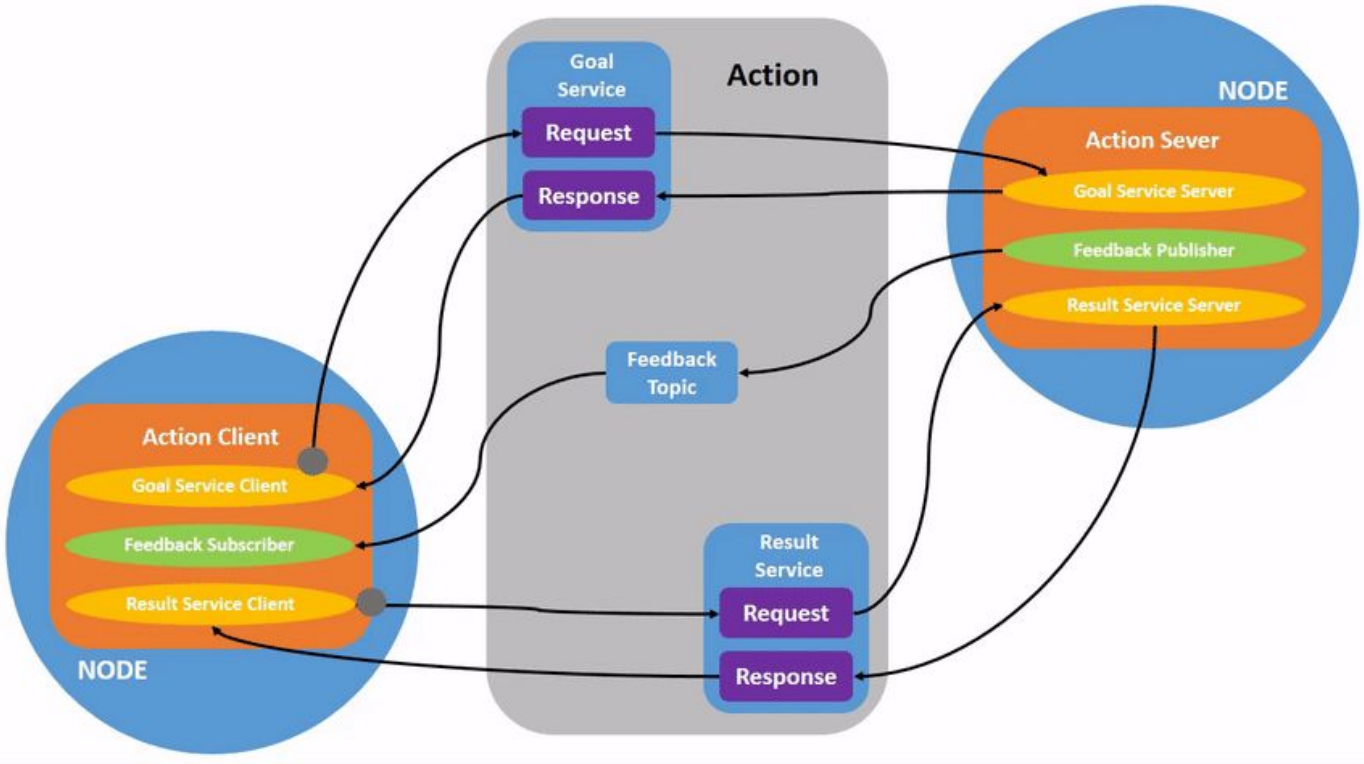
Run the following commands:
(terminal 1) $ roscore
(terminal 2) $ rosrun turtlesim turtlesim_node
(terminal 3)
$ rosparam list
$ rosparam get background_g
$ rosparam set /background_r 150
$ rosservice call /clear



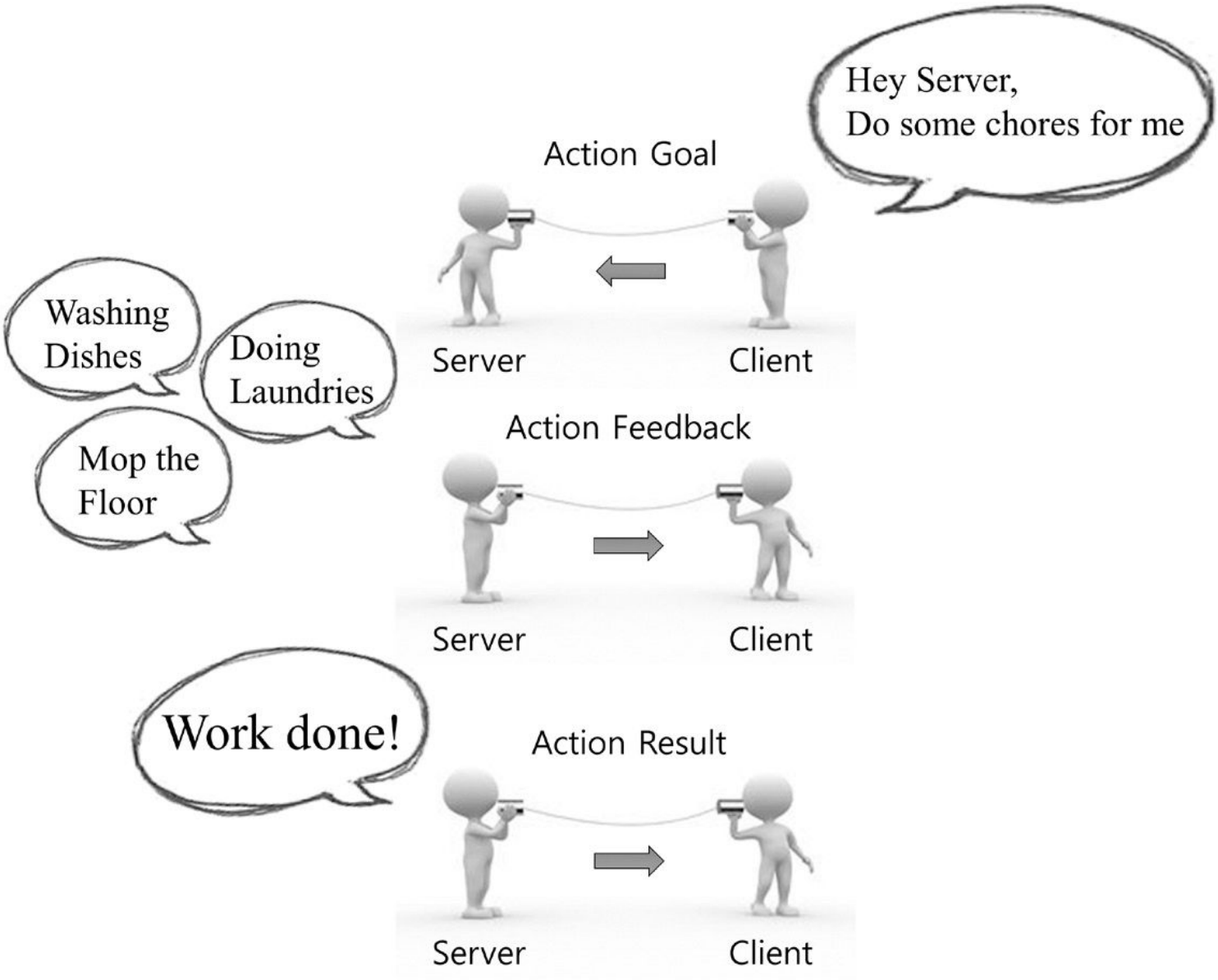
If not working, try
$ rosparam set /turtlesim/background_r 150
$ rosservice call /clear

ROS Actions

- Actions are built on topics and services. Their functionality is similar to services, except actions are preemptable (you can cancel them while executing). They also provide steady feedback, as opposed to services which return a single response.
- Actions use a client-server model, similar to the publisher-subscriber model (described in the topics tutorial). The “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result



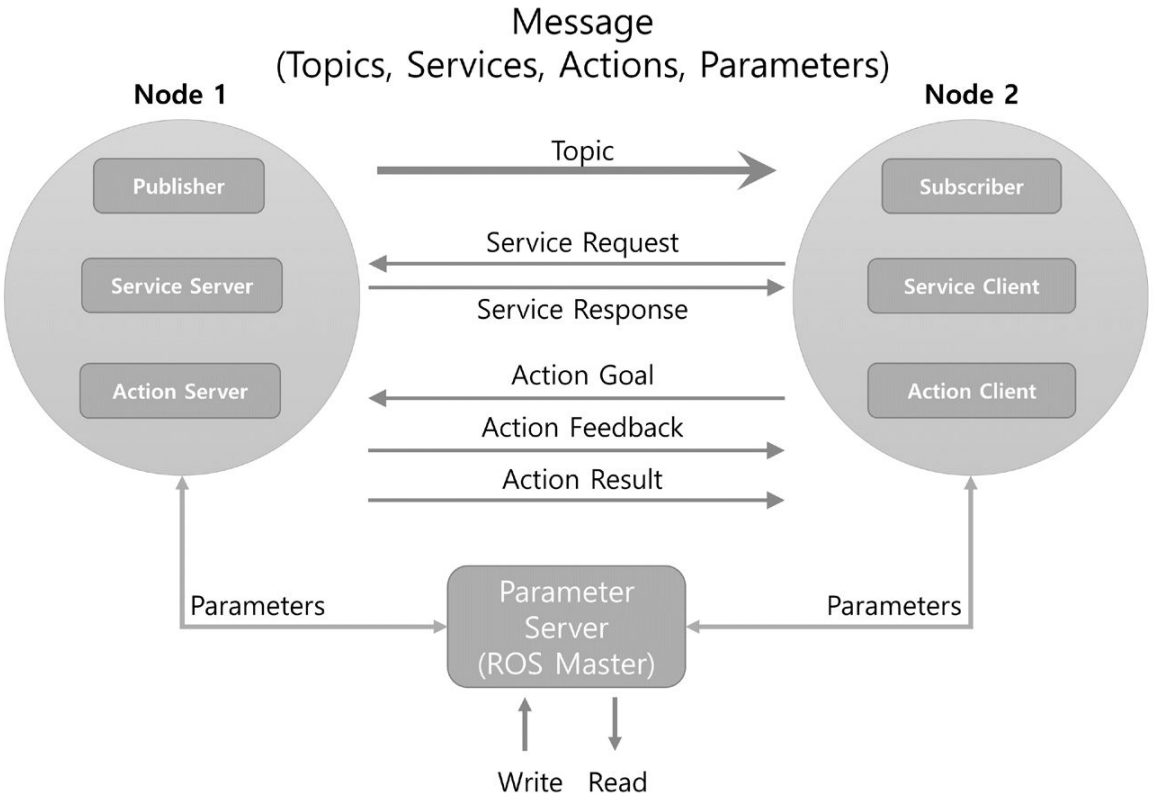
Action Message Communication



ROS Action Commands

- To identify all the actions in the ROS graph, run the commands:
  rosaction list
- To send an action goal from the command line with the following syntax:
  rosaction send_goal <action_name> <action_type> <values>

ROS Communication Summary



Type	Features		Description
Topic	Asynchronous	Unidirectional	Used when exchanging data continuously
Service	Synchronous	Bi-directional	Used when request processing requests and responds current states
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

ROSBag

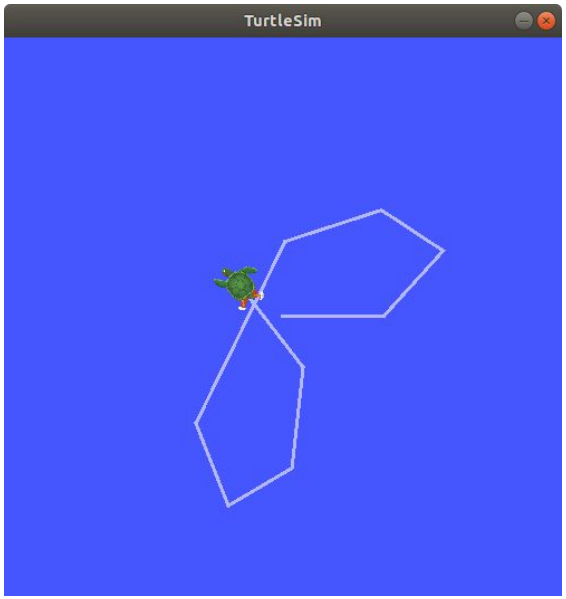
- A rosbag or bag is a file format in ROS for storing message data.
- These bags are often created by subscribing to one or more ROS topics, and storing the received message data in an efficient file structure
- ROS bags is suited for logging and recording dataset for later visualization and analysis

ROS Bag Commands

- To record the data published to a topic:
  rosbag record <topic_name>
- To record and output a specified bag name
  rosbag record <topic_name> -o <bag_file_name>
- To see details about your recording by running:
  rosbag info <bag_file_name>
- To play the recorded data
  rosbag play <bag_file_name>

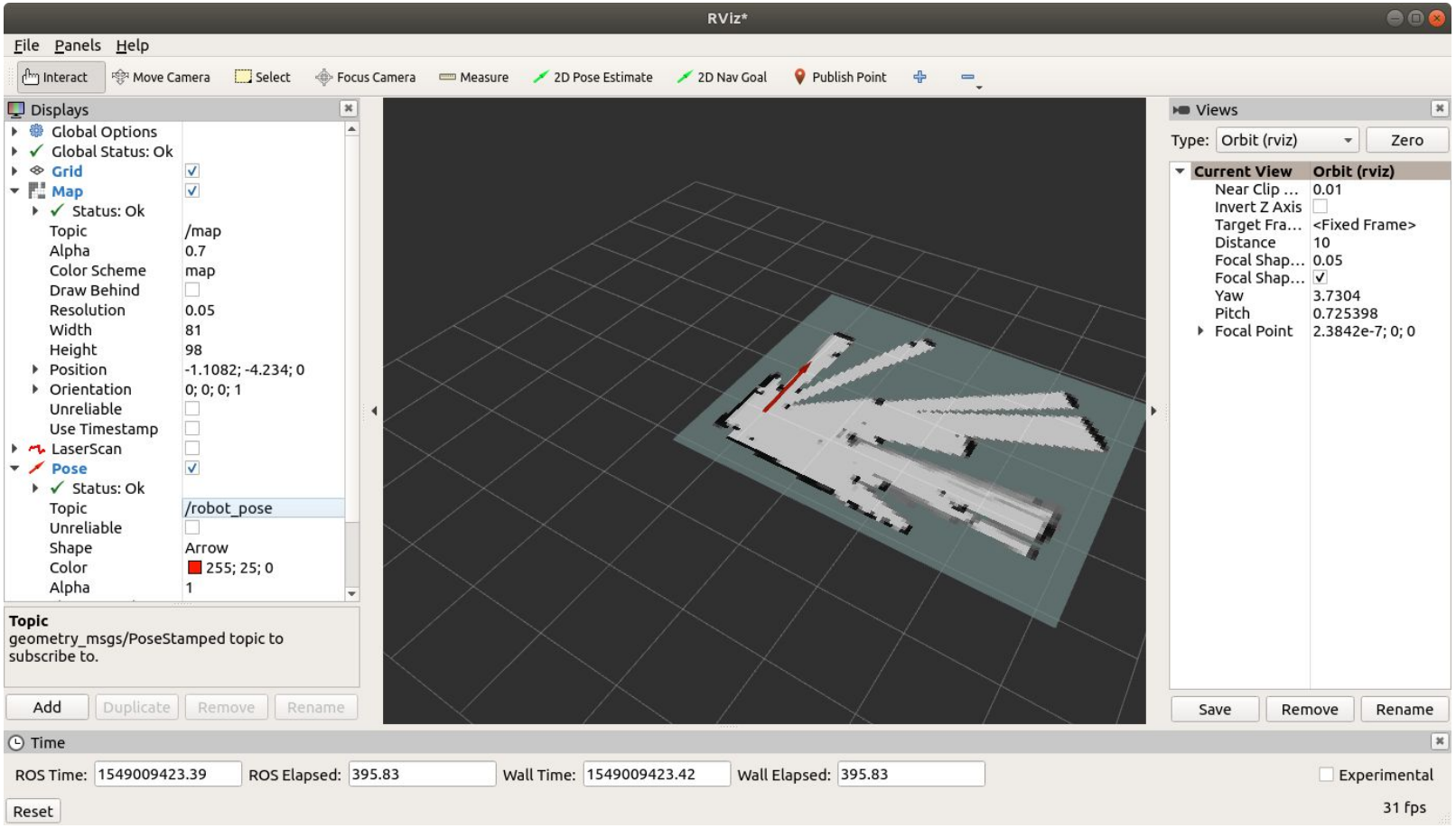
Activity: Record and Play ROS Bag

- (terminal 1) $ roscore
- (terminal 2) $ rosrun turtlesim turtlesim_node
- (terminal 3) $ rosrun turtlesim turtle_teleop_key
- (terminal 4) running rosbag record
- $ mkdir bagfiles
- $ cd bagfiles
- $ rosbag record /turtle1/cmd_vel
- (terminal 3) move the turtle with arrow keys
- (terminal 4) exit with a Ctrl-C.
- (terminal 4) $ rosbag info <your bagfile>
- (terminal 4) $ rosbag play <your bagfile>

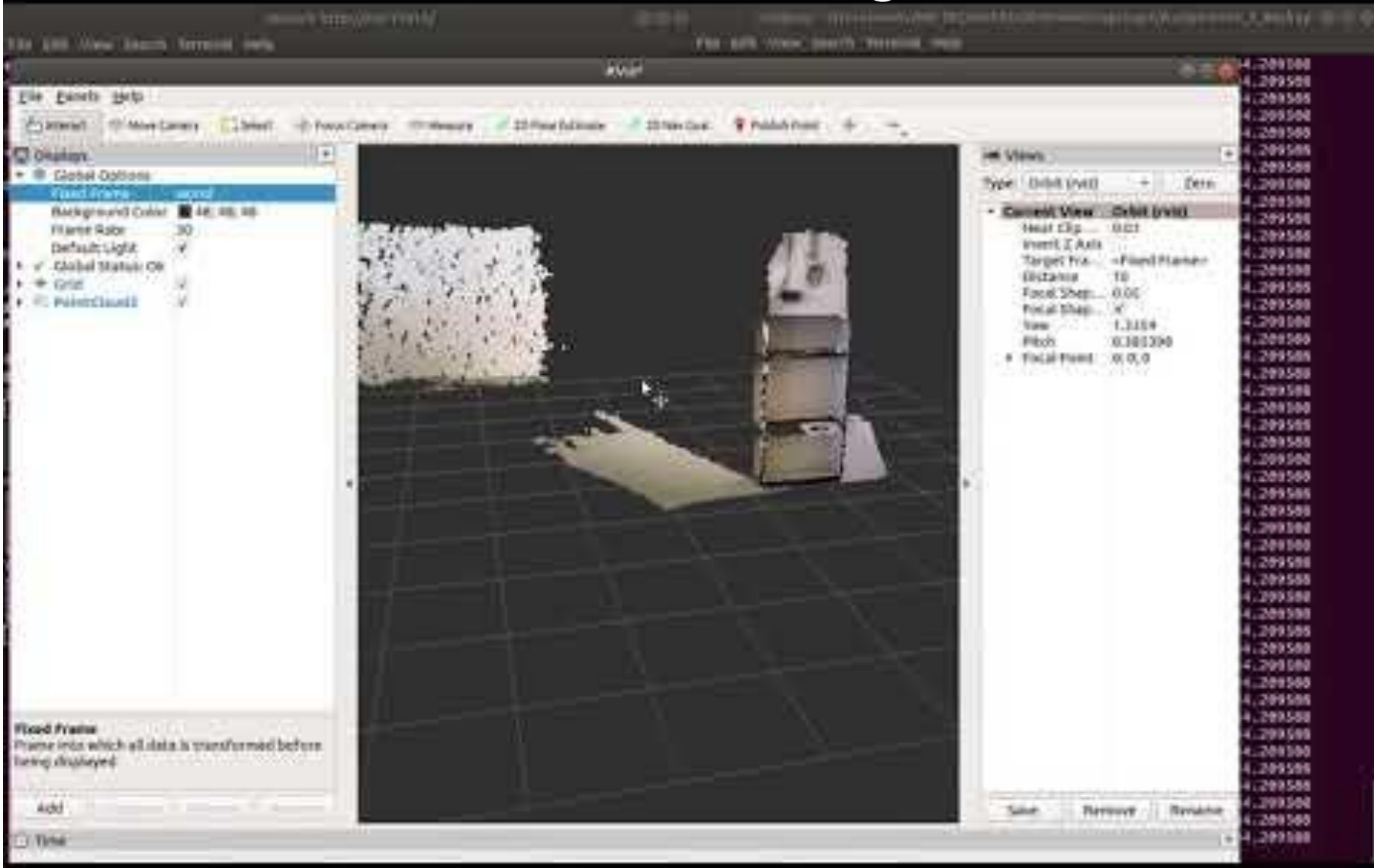


RViz

- RViz (http://wiki.ros.org/rviz) is a 3D visualization tool for ROS
- Subscribes to topics and visualize the message content
- Different camera views
- To run rviz package, type the following command on terminal **rosrun rviz rviz (or rviz)**



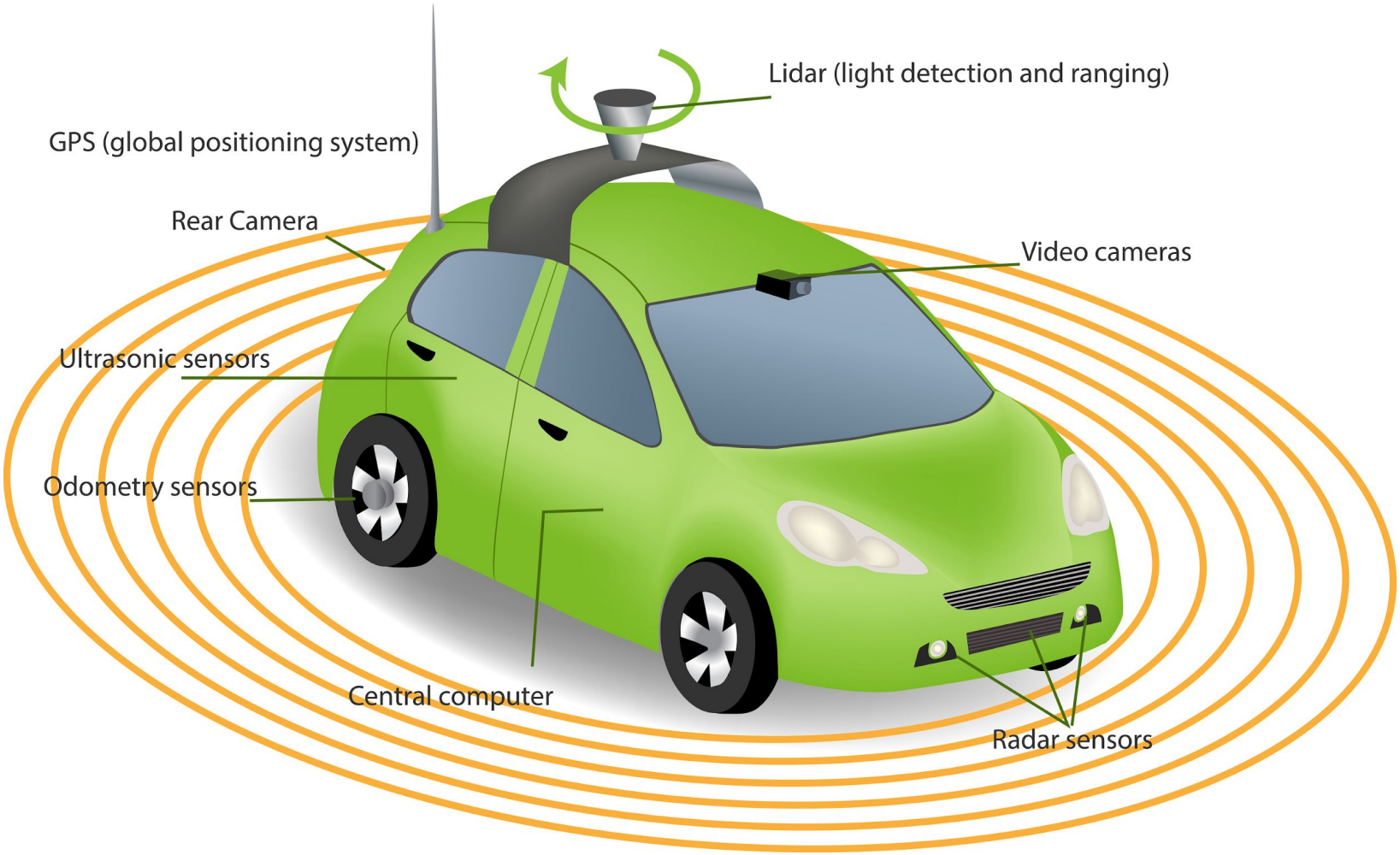
Visual Point Cloud Bags with RViz



Topic 3
LIDAR and SLAM

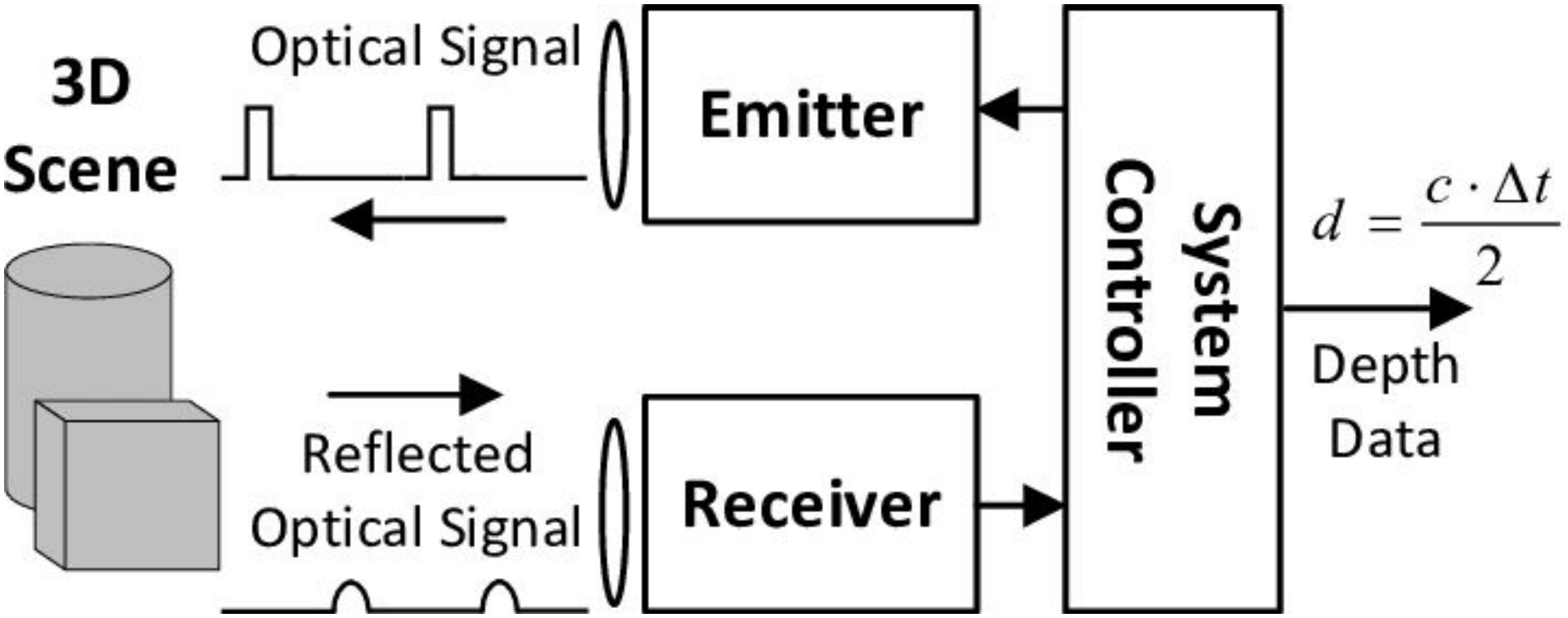
What is LiDAR

- LiDAR (Light Detection and Ranging) is an active remote sensing system that can be used in autonomous vehicle.



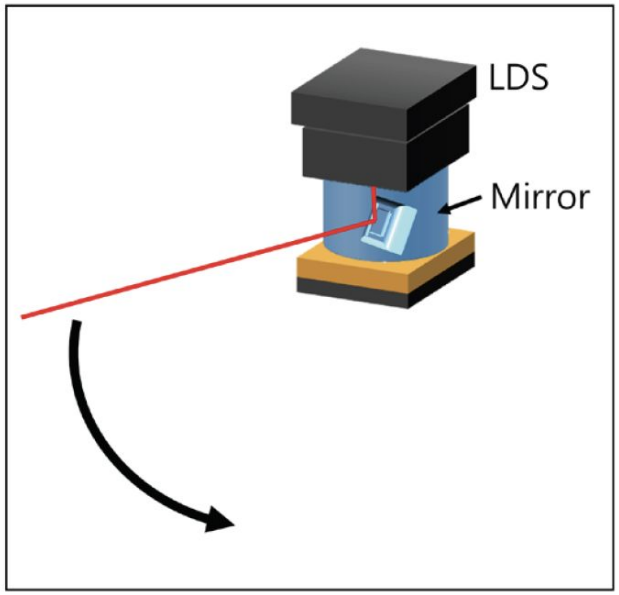
Principle of LIDAR -1

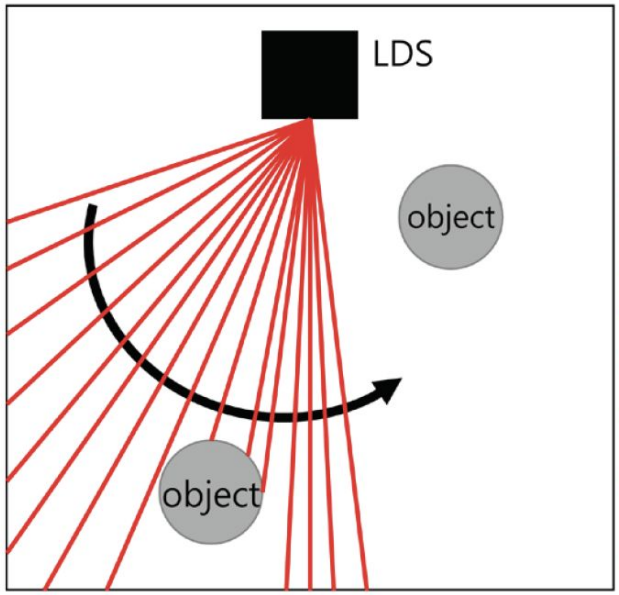
LiDAR (Light Detection And Ranging) sensors work on the same principle as RADAR, firing a wavelength at an object and timing the delay in its return to the source to measure the distance between the two points.

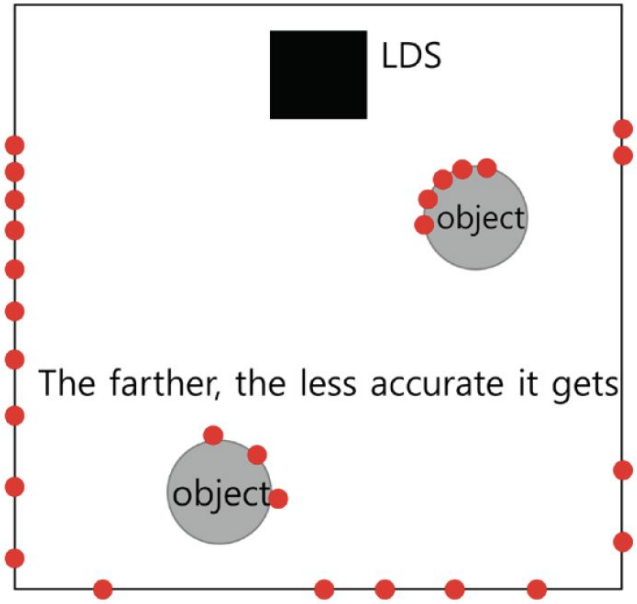


Principle of LIDAR - 2

- The left image shows the LDS with a laser inside and a mirror that is tilted at an angle. The motor rotates the mirror and sensor measures the return time of the laser
- The sensor scans objects in a horizontal
- plane around the LDS as shown in the center image. The accuracy is dropped as the distance becomes longer as shown in the right image.

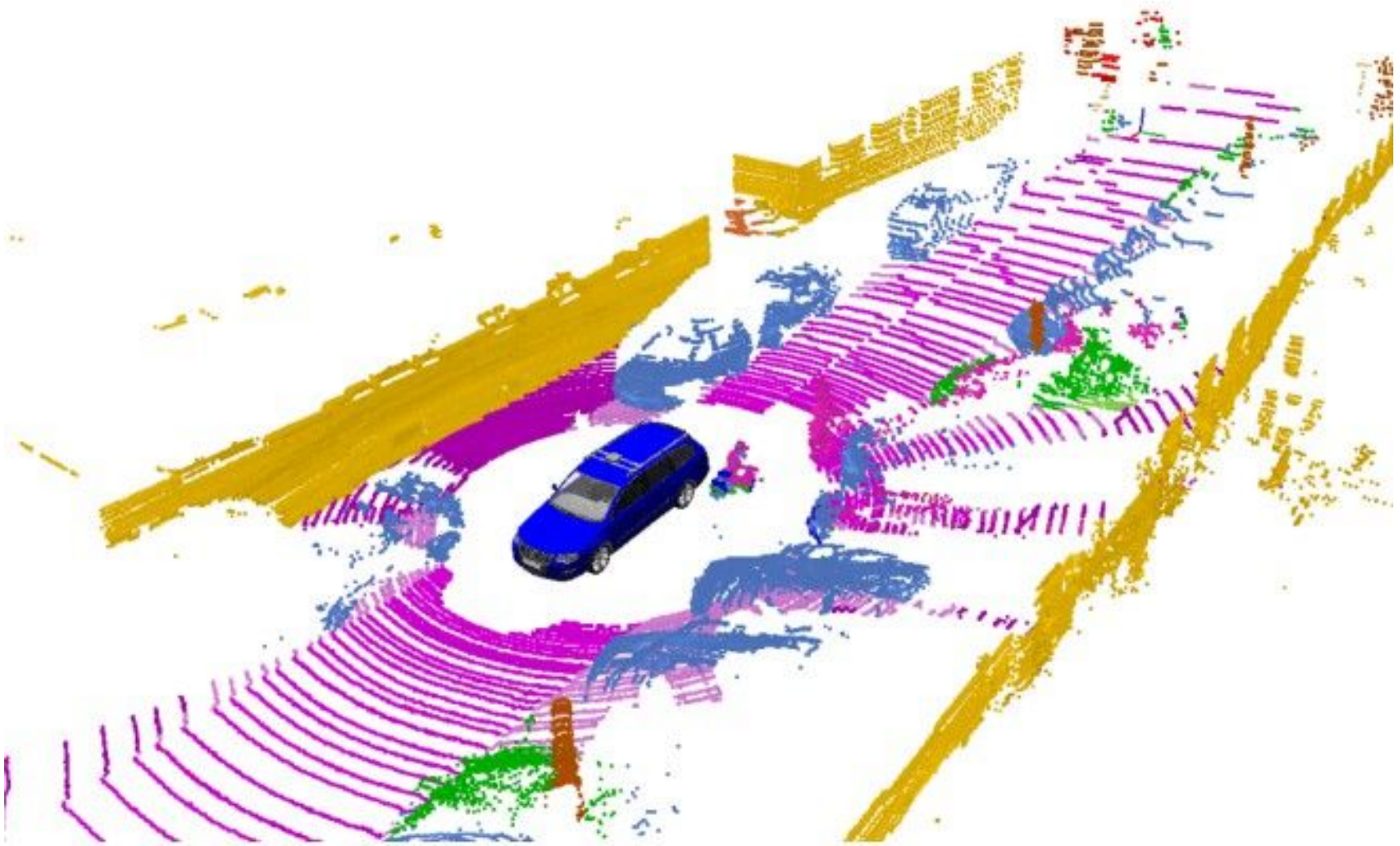






Point Cloud

Point clouds are sets of points that describe an object or surface. To create a point cloud, laser scanning technology like LiDAR can be used



LIDAR for Self Driving Car

LIDAR vs RADAR

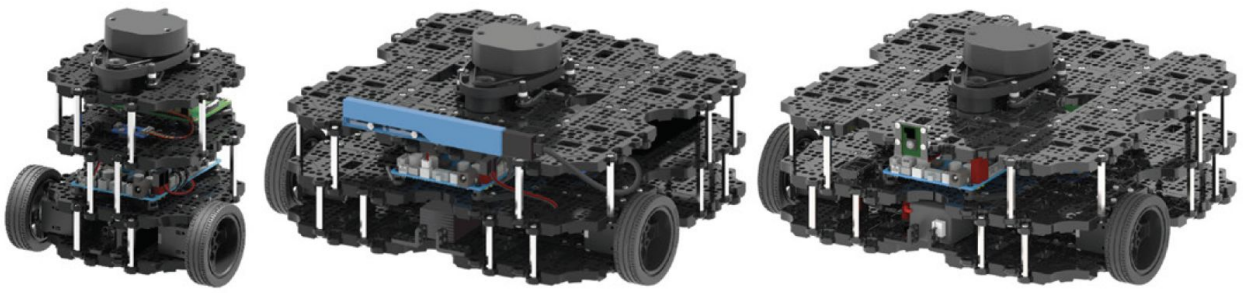
- The RADAR system works in much the same way as the LIDAR, with the only difference being that it uses radio waves instead of laser. In the RADAR instrument, the antenna doubles up as a radar receiver as well as a transmitter. However, radio waves have less absorption compared to the light waves when contacting objects. Thus, they can work over a relatively long distance.
- The LIDAR system can readily detect objects located in the range of 30 meters to 200 meters. But, when it comes to identifying objects in the vicinity, the system is a big letdown. It works well in all light conditions, but the performance starts to dwindle in the snow, fog, rain, and dusty weather conditions.
- Both LIDAR and RADAR have poor optical recognition. That's why, self-driving car manufacturers often use LIDAR along with secondary sensors such as cameras and ultrasonic sensors to identify objects.
- LIDAR system is more expensive than RADAR. However, it is more accurate than RADAR and it can distinguish multiple objects are placed very close to each other.
- Unlike LIDAR, RADAR can determine the velocity of an object using Doppler effect.

TurtleBot

- TurtleBot is a ROS standard platform robot.
- There are three versions of the TurtleBot series.
- TurtleBot1 was developed in 2010 by Tully (Platform Manager at Open Robotics) and Melonee (CEO of Fetch Robotics) from Willow Garage
- In 2012, TurtleBot2 was developed by Yujin Robot based on the research robot, iClebo Kobuki.
- In 2017, TurtleBot3 was developed with features to supplement the lacking functions of its predecessors, and the demands of users. The TurtleBot3 adopts ROBOTIS smart actuator DYNAMIXEL for driving.
- TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability.

TurtleBot1

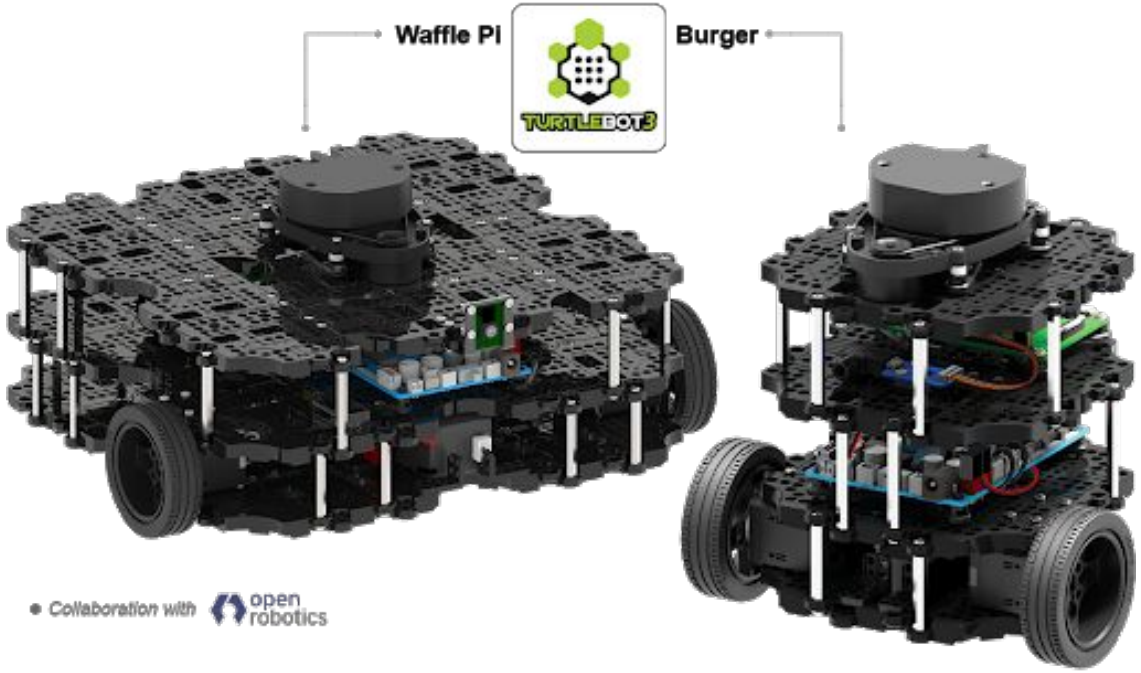
TurtleBot2



TurtleBot3

TurtleBot3

- The two basic types of model structures of TurtleBot3 are Burger and Waffle Pi.
- The basic components of TurtleBot3 are actuators, an SBC for operating ROS, a sensor for SLAM and navigation, restructuring mechanism, an OpenCR embedded board used as a sub-controller, sprocket wheels that can be used with tire and caterpillar, and a 3 cell lithium-poly battery.
- TurtleBot3 Waffle Pi is the same shape as the Waffle model, but this model is used the Raspberry Pi as the Burger model, and the Raspberry Pi Camera to make it more affordable.



TurtleBot3 Burger and Waffle

Turtlebot3 ROS Packages

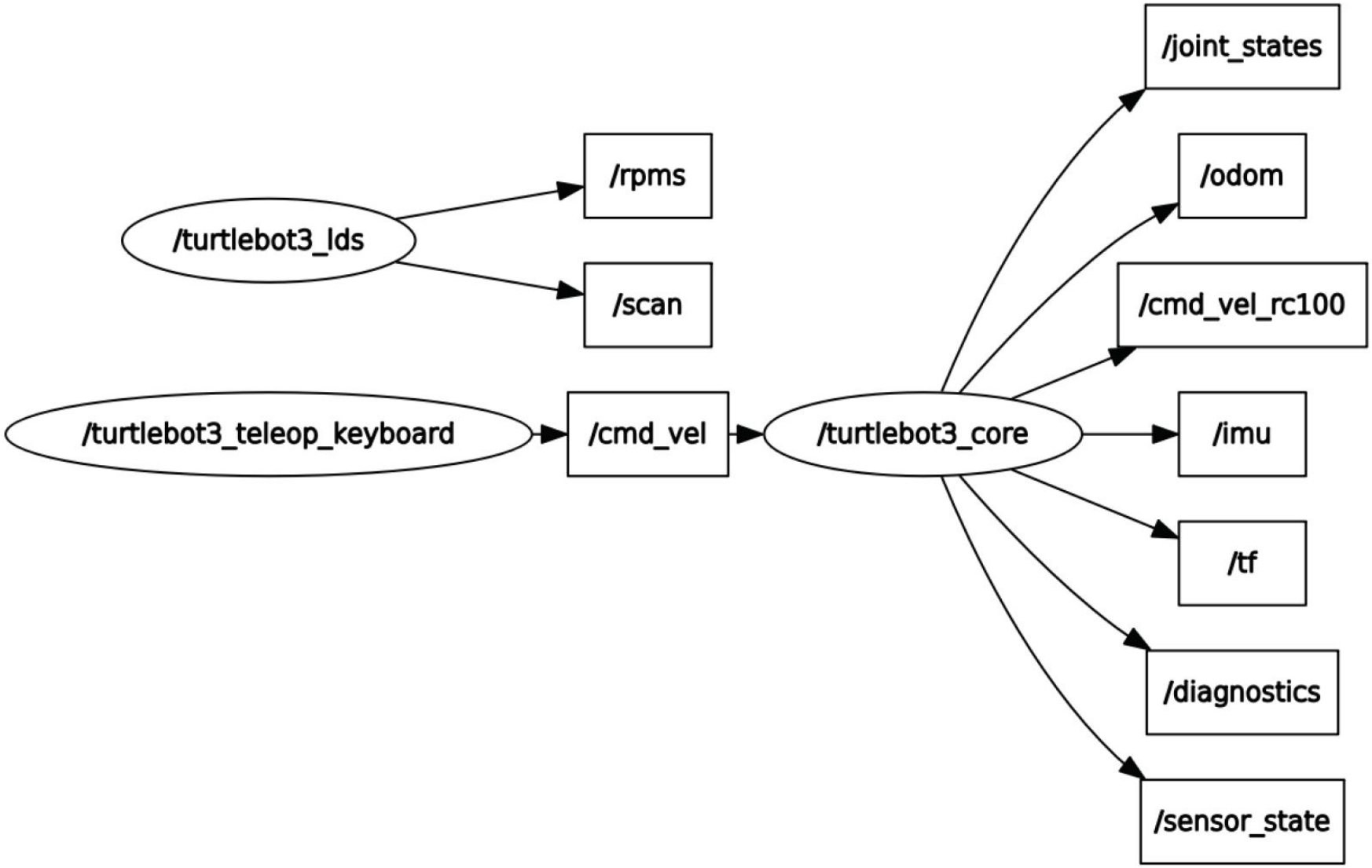
- TurtleBot3's ROS package includes 4 packages which are 'turtlebot3', 'turtlebot3_msgs', 'turtlebot3_simulations', and 'turtlebot3_applications'.
  The 'turtleBot3' package contains TurtleBot3's robot model, SLAM and navigation package, remote control package, and bringup package.
- The 'turtlebot3_msgs' package contains message files used in turtlebot3.
- The 'turtlebot3_simulations' contains packages related to simulation
- The 'turtlebot3_applications' package contains applications.
-

Activity: Install Turtlebot3 Packages on Remote PC

(terminal 1)
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server ros-kinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf ros-kinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqt-image-view ros-kinetic-gmapping ros-kinetic-navigation ros-kinetic-interactive-markers

$ cd ~/catkin_ws/src/
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws && catkin_make
$ source devel/setup.bash

Turtlebot3 Nodes and Topics



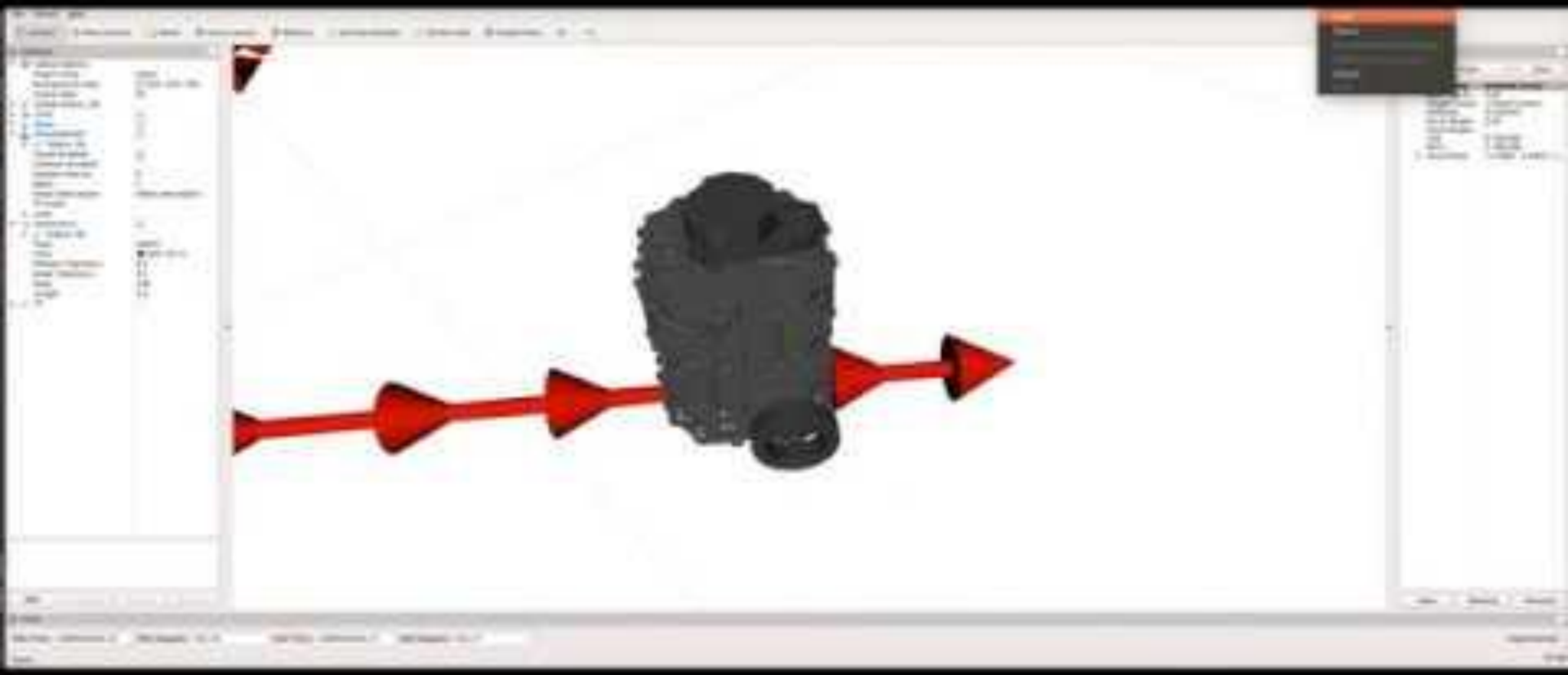
Turtlebot3 Topics

Topic Name	Format	Function
sensor_state	turtlebot3_msgs/SensorState	Topic that contains the values of the sensors mounted on the TurtleBot3
scan	sensor_msgs/LaserScan	Topic that confirms the scan values of the LiDAR mounted on the TurtleBot3
imu	sensor_msgs/Imu	Topic that includes the attitude of the robot based on the acceleration and gyro sensor.
odom	nav_msgs/Odometry	Contains the TurtleBot3's odometry information based on the encoder and IMU
tf	tf2_msgs/TFMessage	Contains the coordinate transformation such as base_footprint and odom
joint_states	sensor_msgs/JointState	Checks the position (m), velocity (m/s) and effort (N·m) when the wheels are considered as joints.
cmd_vel	geometry_msgs/Twist	This topic is published when the Bluetoothbased controller, RC100, is connected to control the velocity (m/s) and angular speed (rad/s) of mobile robot.

Turtlebot3 Simulation

- TurtleBot3 supports development environment that can be programmed and developed with a virtual robot in the simulation.
- There are two development environments to do this, one is using fake node and 3D visualization tool RViz and the other is using the 3D robot simulator Gazebo.
- The fake node method is suitable for testing with the robot model and movement, but it can not use sensors.
- If you need to test SLAM and Navigation, we recommend using Gazebo, which can use sensors such as IMU, LDS, and camera in the simulation.

Fake Node Simulation Demo



Activity: Burger Simulation

(terminal 1)
$ roscore

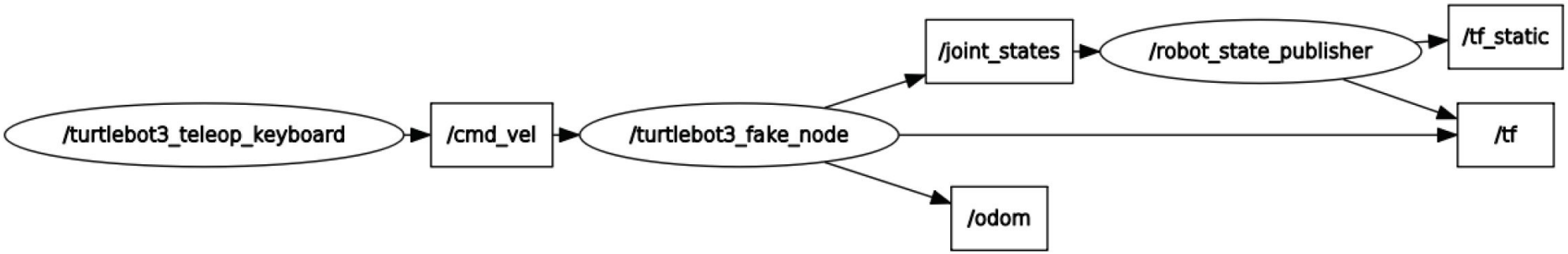
(terminal 2)
$ nano ~/.bashrc
$ export TURTLEBOT3_MODEL=burger
$ source ~/.bashrc
$ roslaunch turtlebot3_fake turtlebot3_fake.launch

(terminal 3)
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

You can automate by edit the .bashrc file
$ nano ~/.bashrc
and add the following lines to .bashrc file
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
After that,
$ source ~/.bashrc

Odometry and TF

- The ‘turtlebot3_fake_node’ receives the speed command to generate odometry information.
- The node publishes ‘odometry’, ‘joint_state’, and ‘tf’ via topic so they can be used to visualize the movement of TurtleBot3 in RViz.



Gazebo

- Gazebo is a 3D simulator that provides robots, sensors, environment models for 3D simulation required for robot development, and offers realistic simulation with its physics engine.
- Gazebo is one of the most popular simulators for open source in recent years, and has been selected as the official simulator of the DARPA Robotics Challenge in the US.
- It is a very popular simulator in the field of robotics because of its high performance even though it is open source.
- Moreover, Gazebo is developed and distributed by Open Robotics which is in charge of ROS and its community, so it is very compatible with ROS.

"rviz shows you what the robot thinks is happening, while Gazebo shows you what is really happening."

Gazebo Simulation

TURTLEBOT3

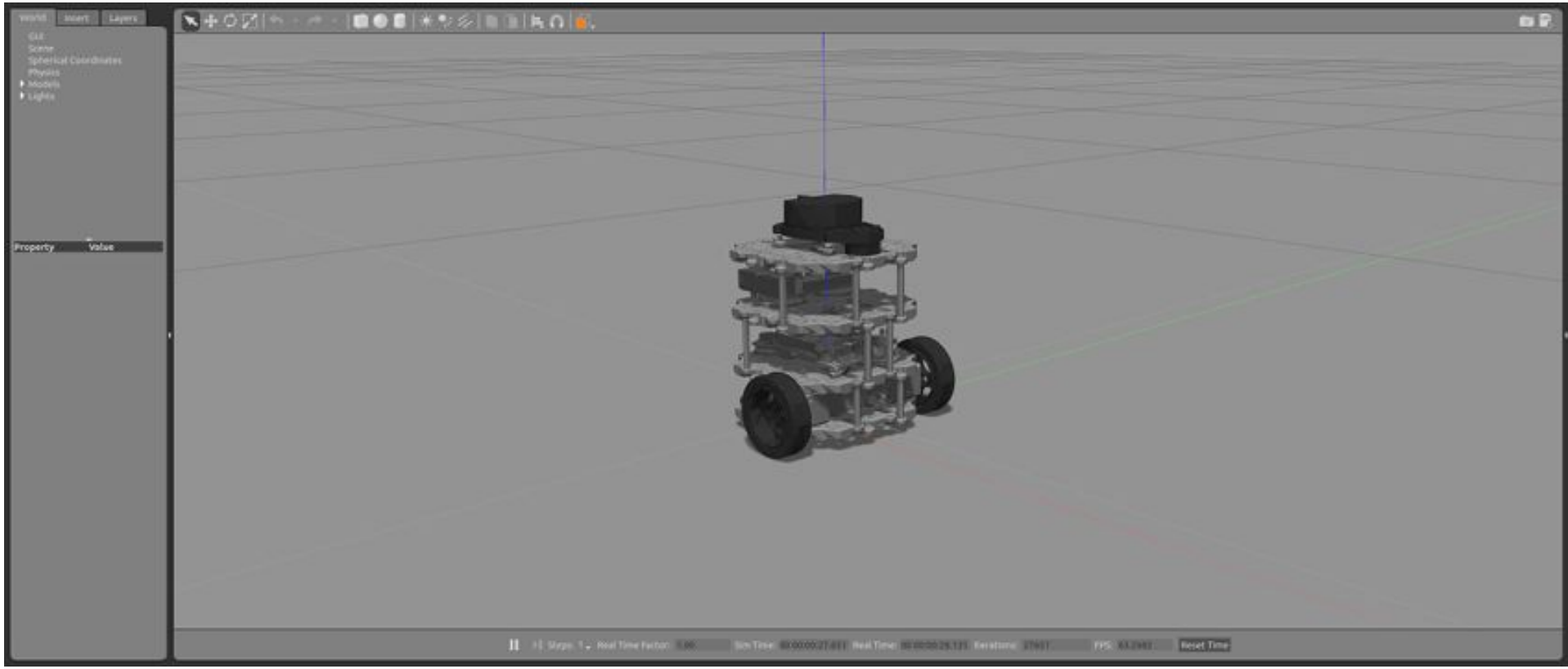
TurtleBot3
Burger

TurtleBot3
Waffle Pi

Gazebo

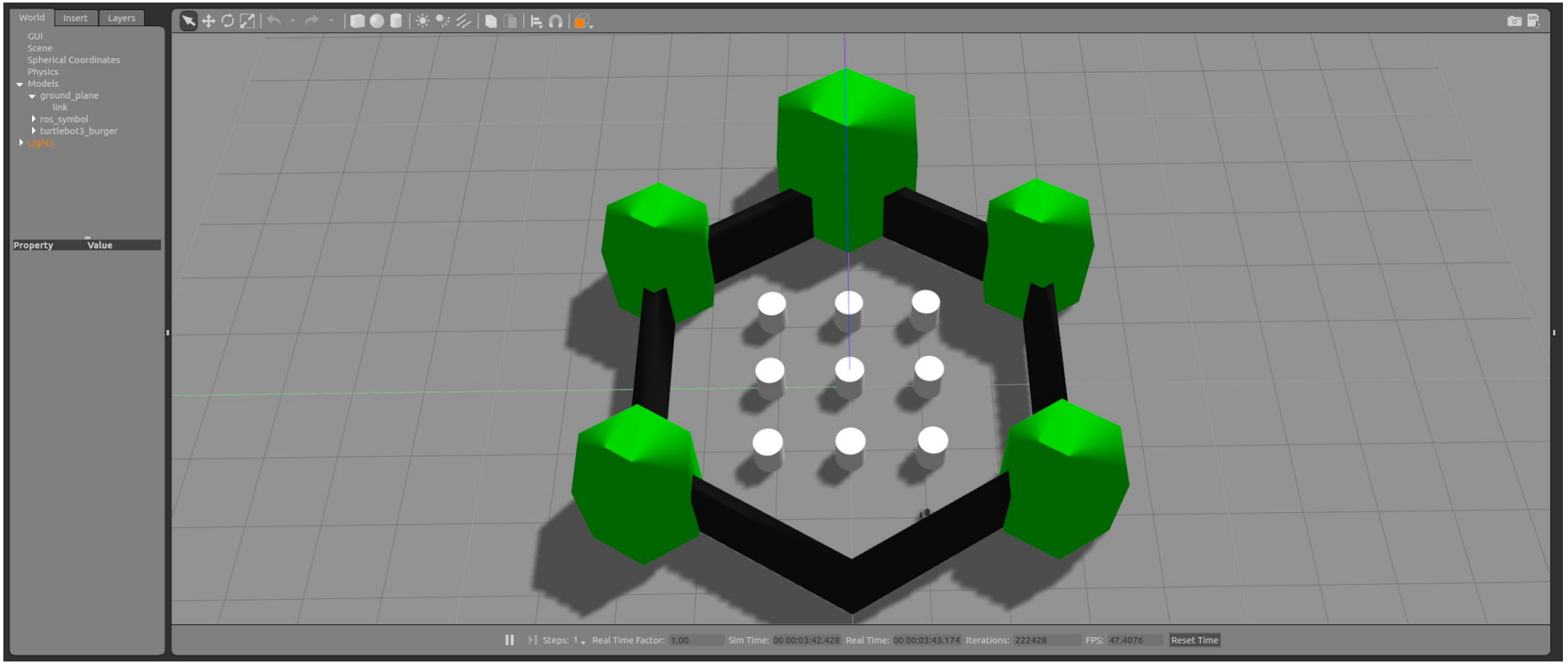
Activity: Gazebo - Empty World

$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



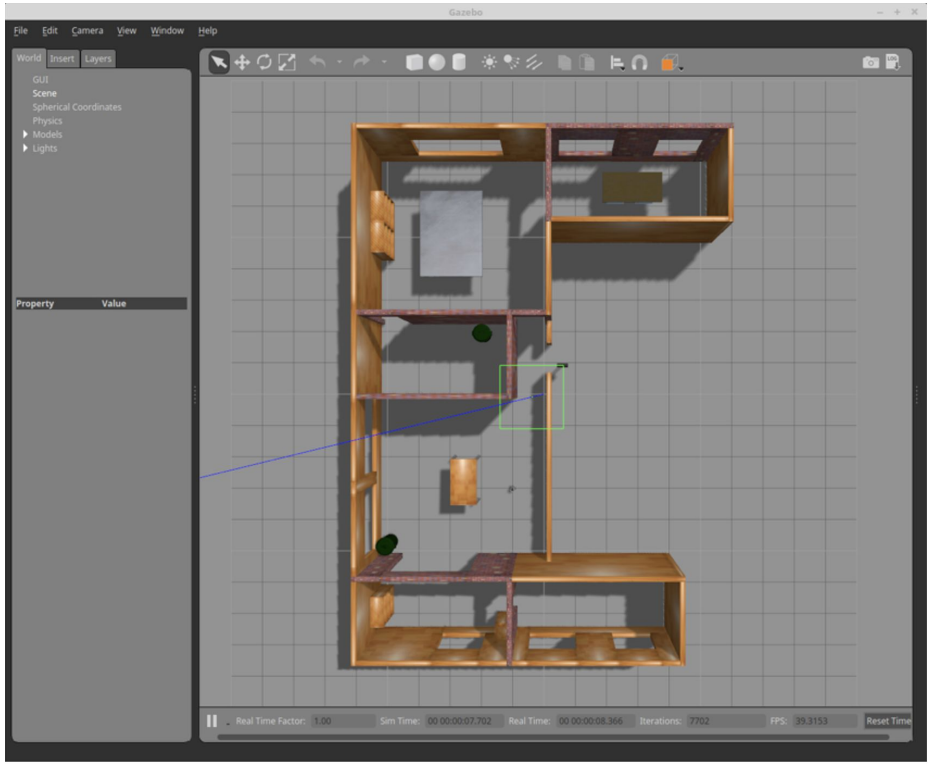
Activity: Gazebo - World

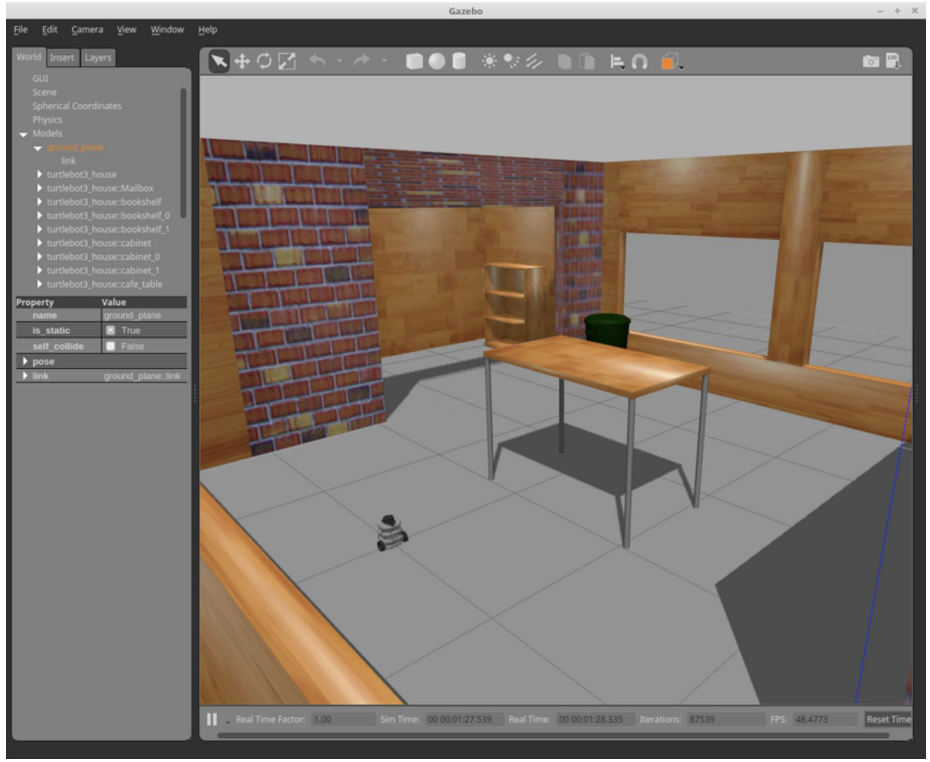
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



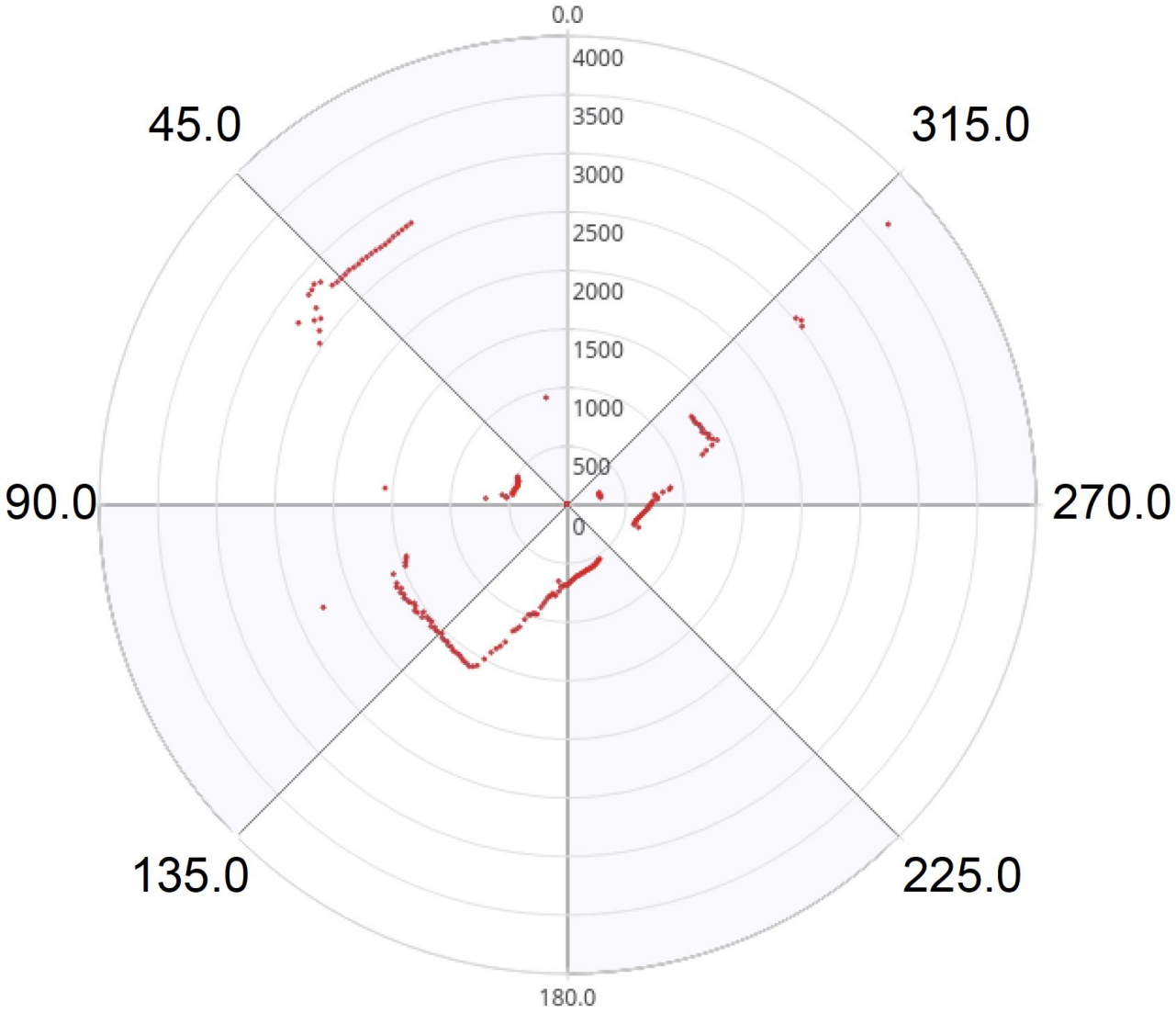
Activity: Gazebo - House

$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch





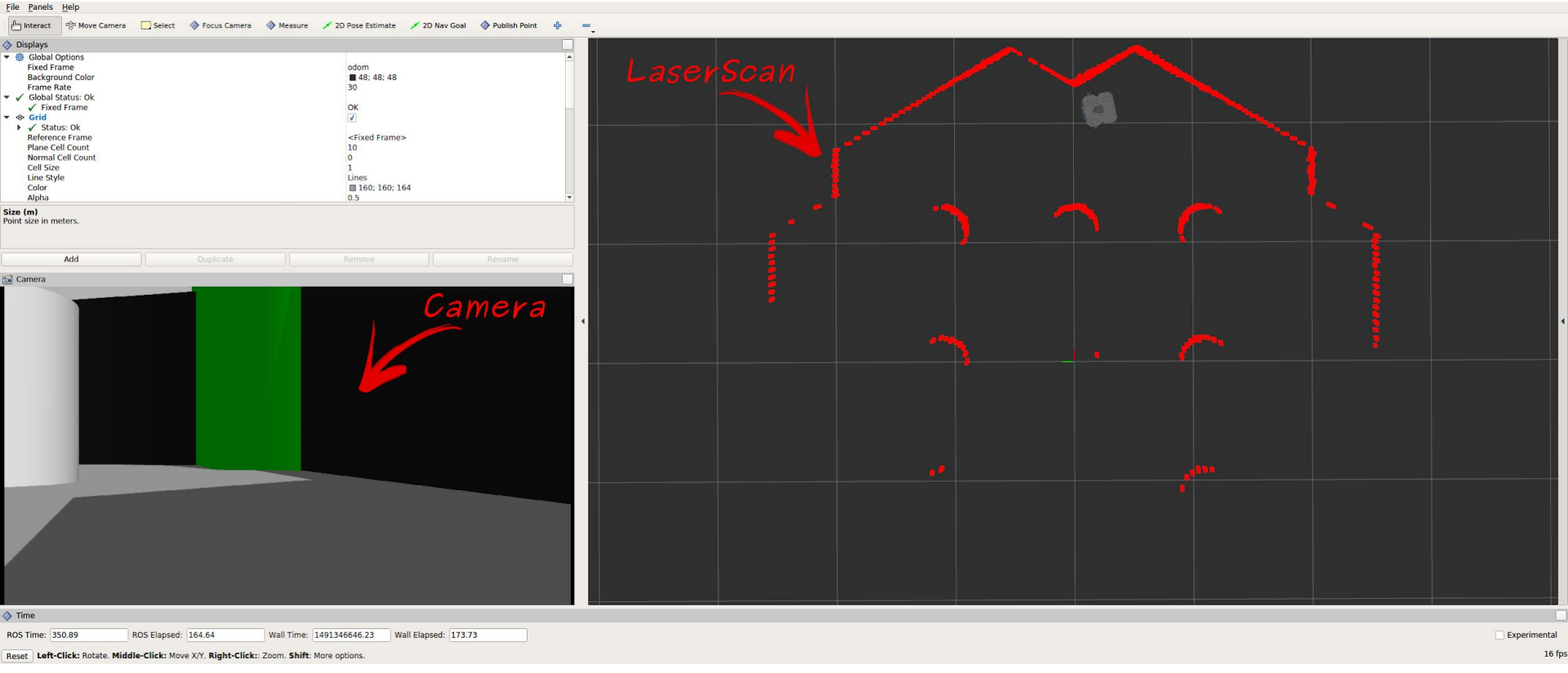
Laser Scan Data from LIDAR



Activity: LiDAR Scan Data in RViz

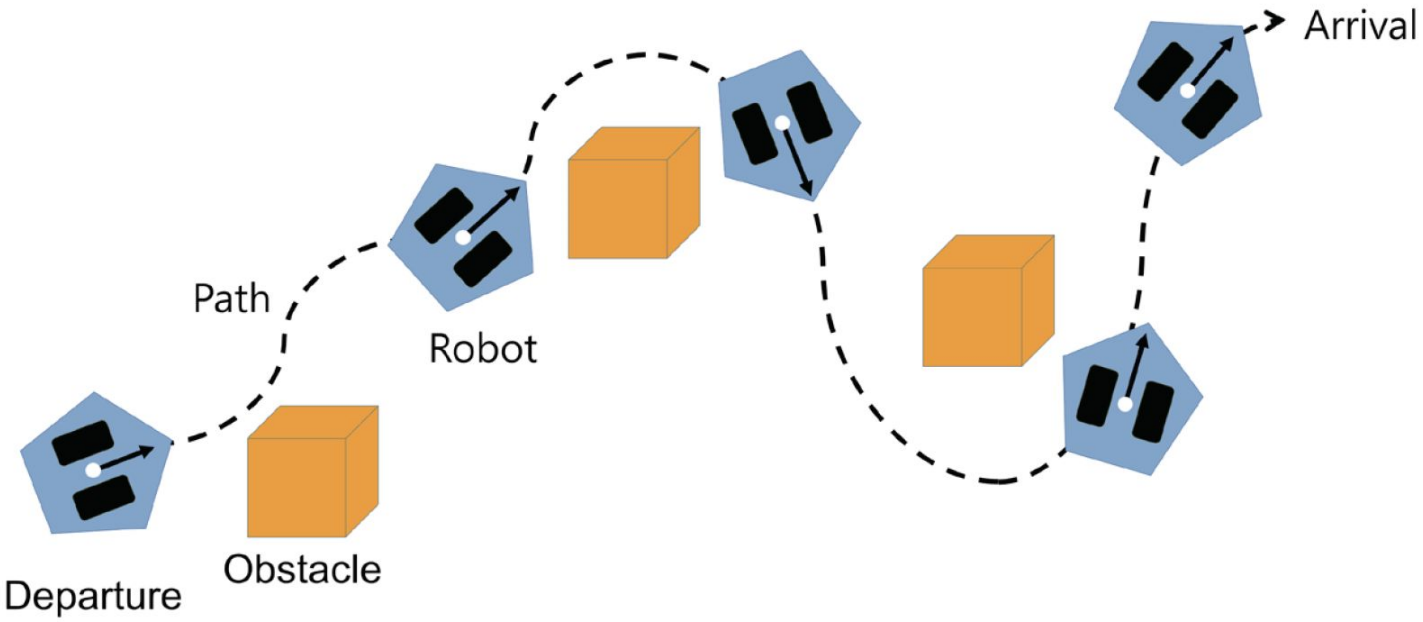
RViz can complement Gazebo to show the sensor data eg LIDAR and camera

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



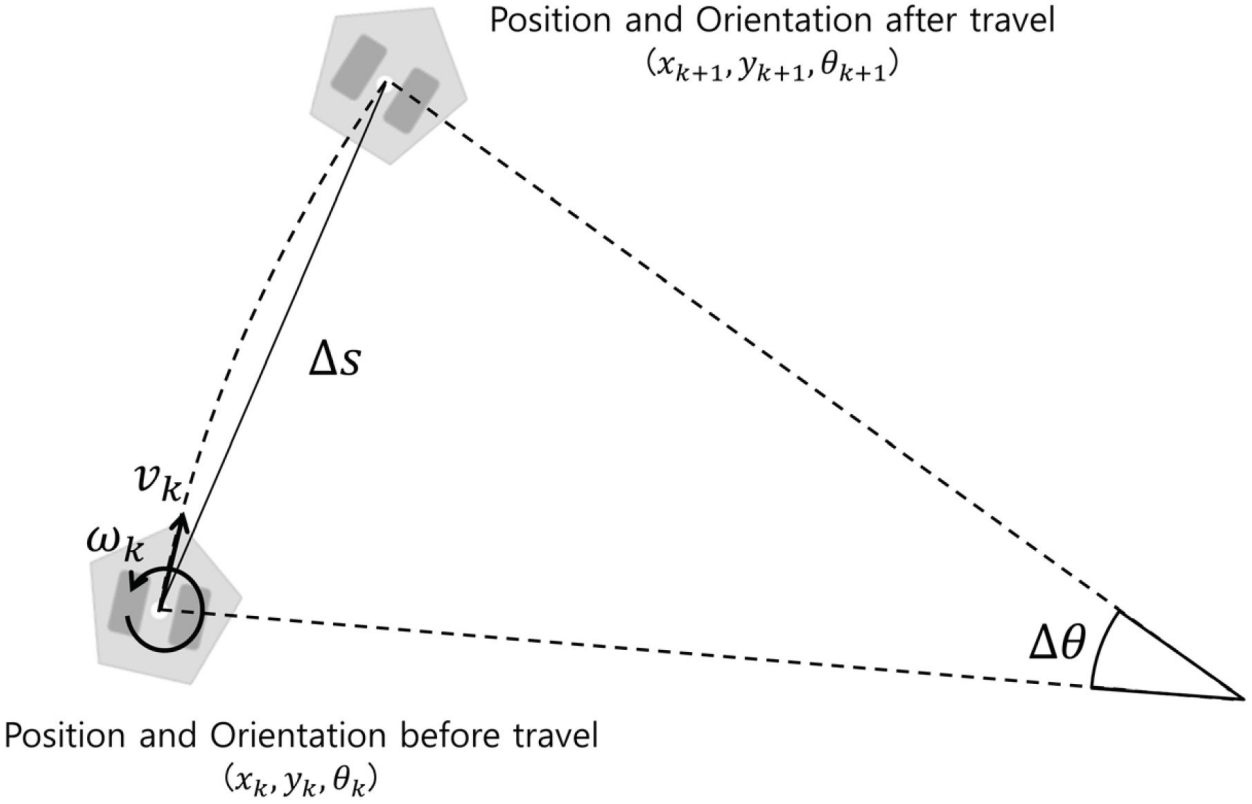
Robot Navigation

- Navigation is the movement of the robot to a defined destination, which is not as easy as it sounds.
- But it is important to know where the robot itself is and to have a map of the given environment. It is also important to find the optimized route among the various routing options, and to avoid obstacles such as walls and furniture.
- What do we need to implement navigation in robots? It may vary depending on the navigation algorithm, and the followings may be required as basic features - Map, Pose of Robot, Sensing, Path Calculation and Driving



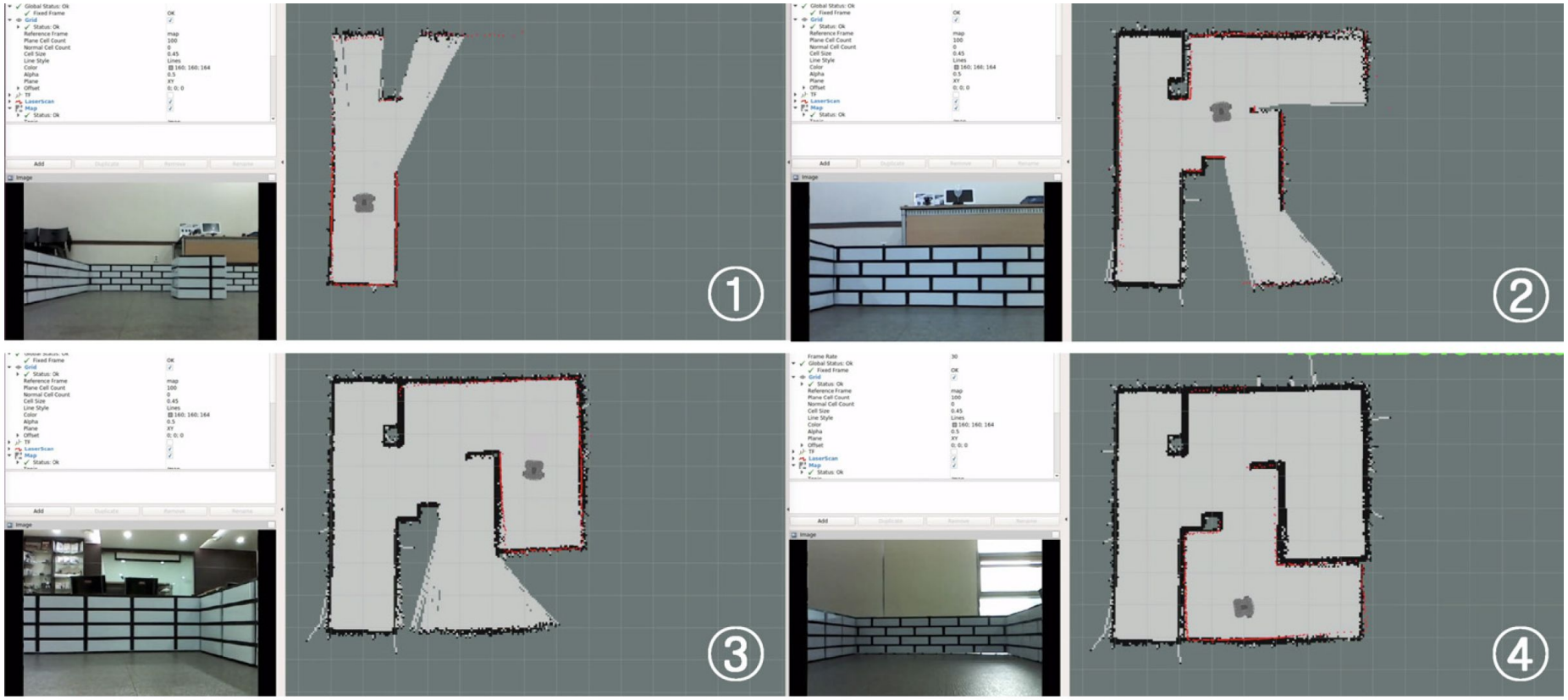
Pose of a Robot

- ROS defines pose as the combination of the robot's position(x, y, z) and orientation (x, y, z, w).
- The orientation is described by x, y, z, and w in quaternion form, whereas position is described by three vectors, such as x, y, and z.



SLAM

- SLAM (Simultaneous Localization And Mapping) means to explore and map the unknown environment while estimating the pose of the robot itself by using the mounted sensors on the robot.
- This is the key technology for navigation such as autonomous driving.



SLAM and Pose Estimation Algorithms

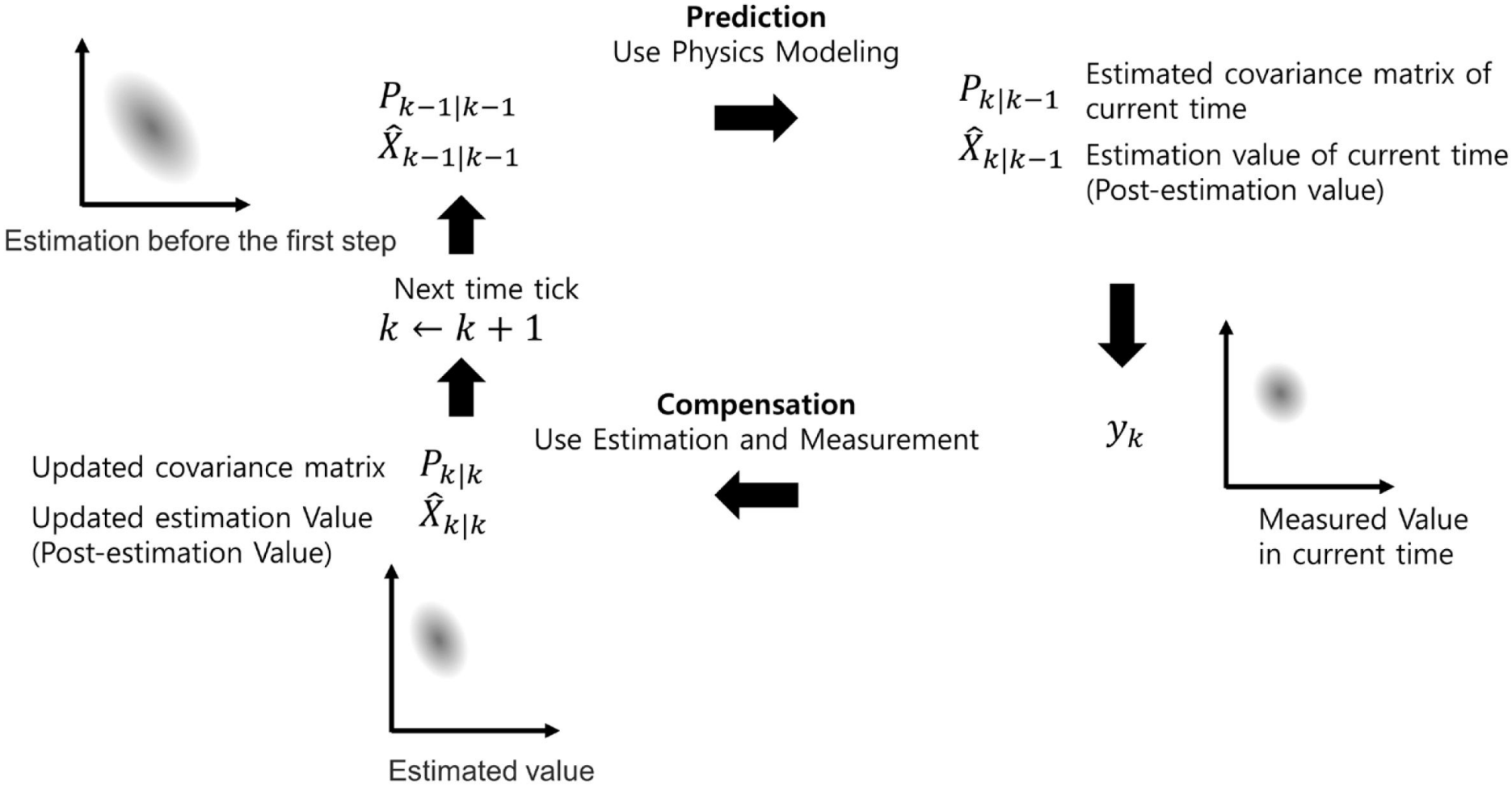
- Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.
- While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments.
- Popular approximate solution methods include
  - Extended Kalman filter
  - Particle filter
  - AMCL
  - Gmapping (ROS package available)
  - Cartographer (ROS package available)

Kalman Filter

- The Kalman filter which was used in NASA's Apollo project was developed by Dr. Rudolf E. Kalman, who has since then become famous for the algorithm.
- His filter was a recursive filter that tracks the state of an object in a linear system with noise.
- The filter is based on the Bayes probability which assumes the model and uses this model to predict the current state from the previous state.
- Then, an error between the predicted value of the previous step and the actual measured present value obtained by measuring instrument is used to perform an update step of estimating more accurate state value.
- The filter repeats above process and increases the accuracy.

Kalman Filter

However, the Kalman filter only applies to linear systems. Most of our robots and sensors are nonlinear systems, and the EKF (Extended Kalman Filter) modified from Kalman filter are widely used.

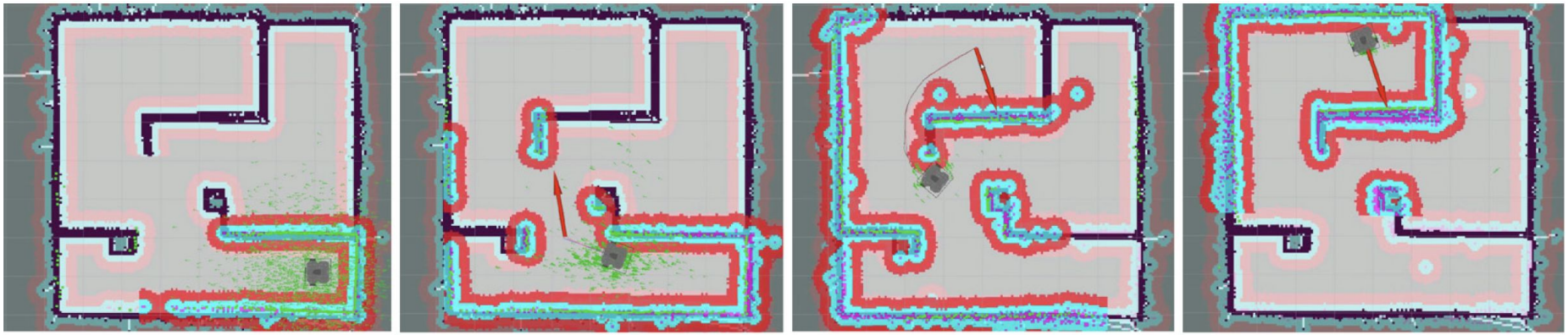


Particle Filter

- Particle Filter is the most popular algorithm in object tracking.
- A particle filter gained its name because the estimated value generated by the probability distribution in the system is represented as particles.
- This is also called the Sequential Monte Carlo (SMC) method or the Monte Carlo method.
- In the particle filter method, the uncertain pose is described by a bunch of particles called samples.
- We move the particles to a new estimated position and orientation based on the robot's motion model and probabilities, and measure the weight of each particle according to the actual measurement value, and gradually reduce the noise to estimate a precise pose.
- In the case of a mobile robot, each particle is represented as a particle = pose (x, y, i), weight, and each particle is an arbitrary small particle representing the estimated position and orientation of the robot expressed by x, y, and i of the robot and the weight of each particle.

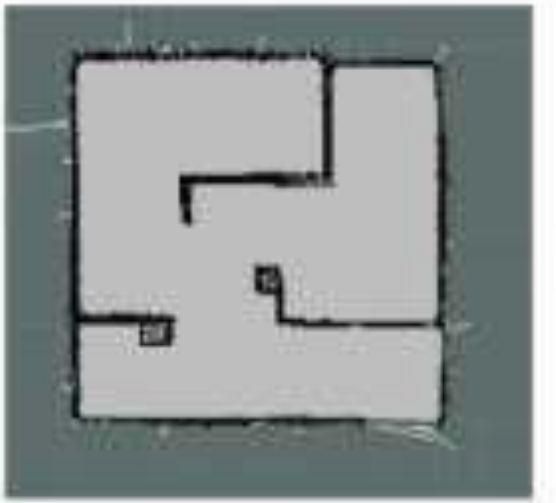
Adaptive Monte Carlo Localization

- The Monte Carlo localization (MCL) pose estimation algorithm is widely used in the field of pose estimation.
- AMCL (Adaptive Monte Carlo Localization) can be regarded as an improved version of Monte Carlo pose estimation, which improves real-time performance by reducing the execution time with less number of samples in the Monte Carlo pose estimation algorithm.



(t1) (t2) (t3) (t4)

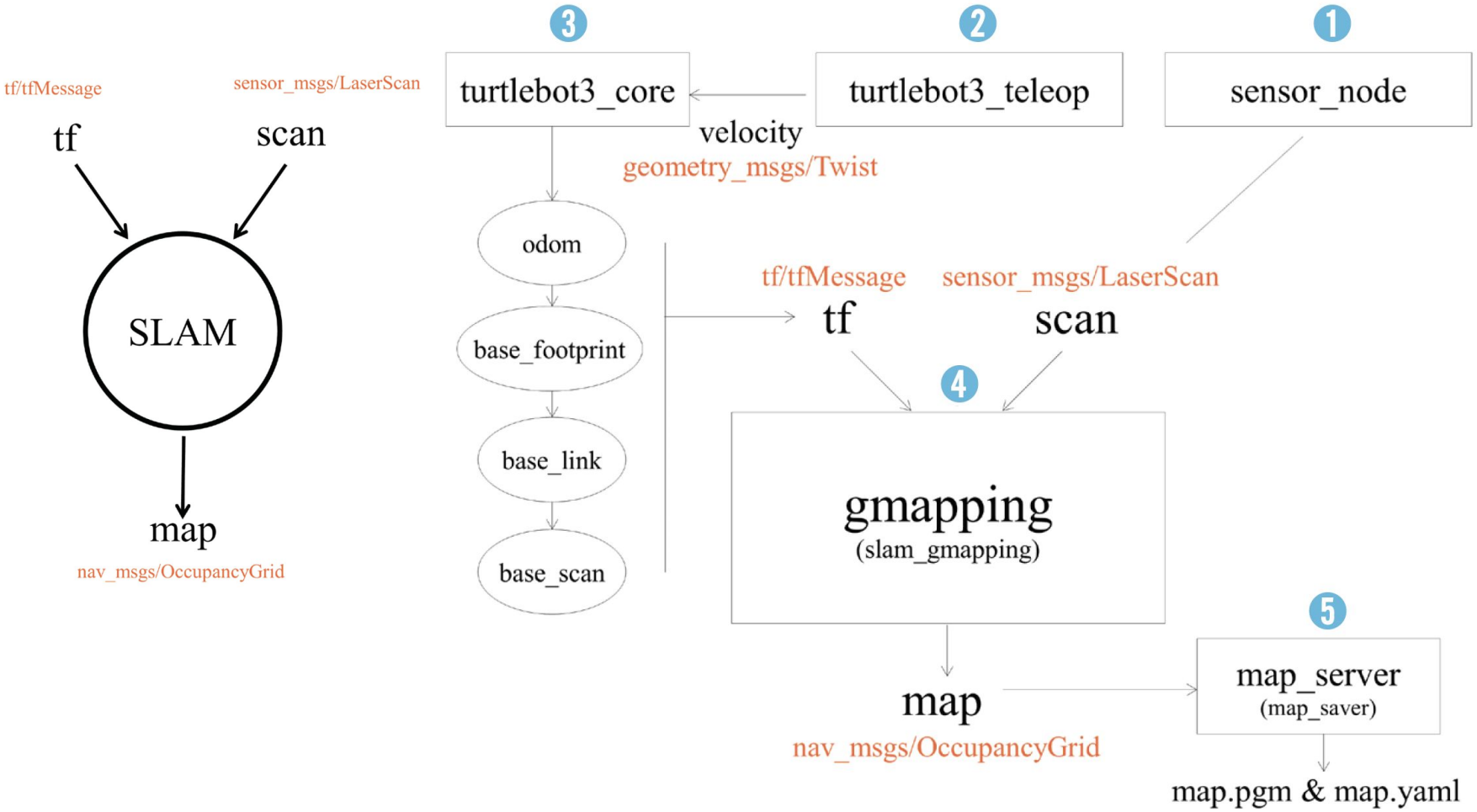
SLAM Demo



SLAM Demo

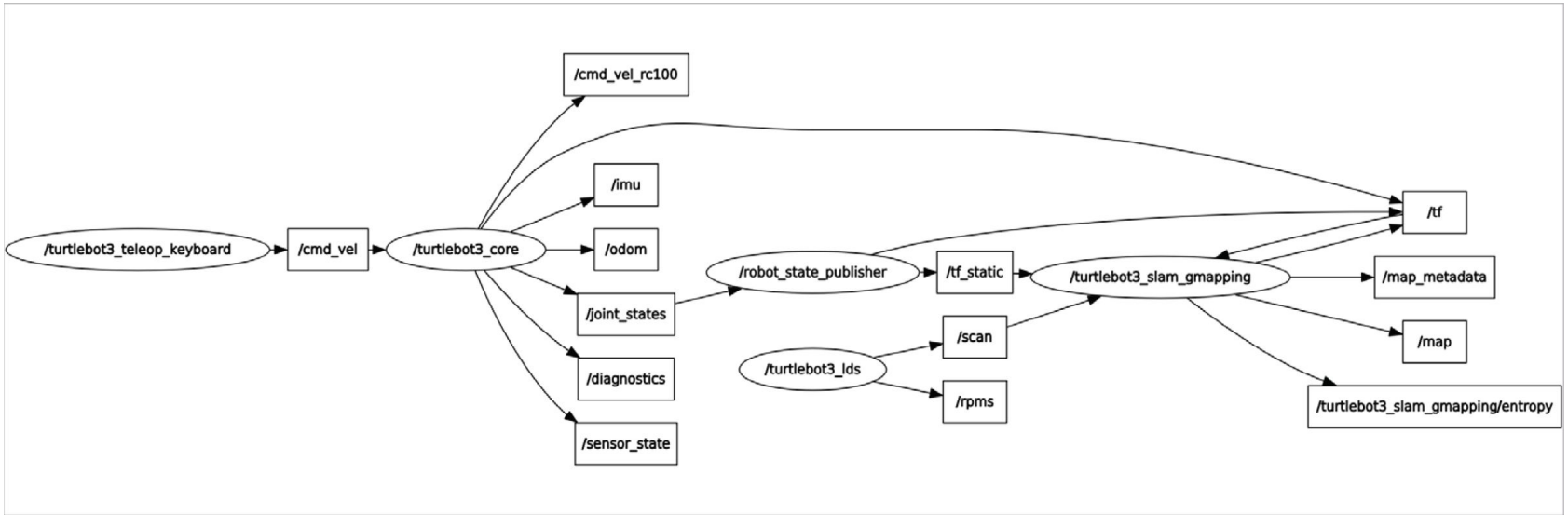
Turtlebot3 SLAM Process

- To create the map with SLAM, the 'turtlebot3_slam' package was created in addition to the 'turtlebot3_core' node.
- This package does not have a source file, but packages needed for SLAM is launched when executing the launch file

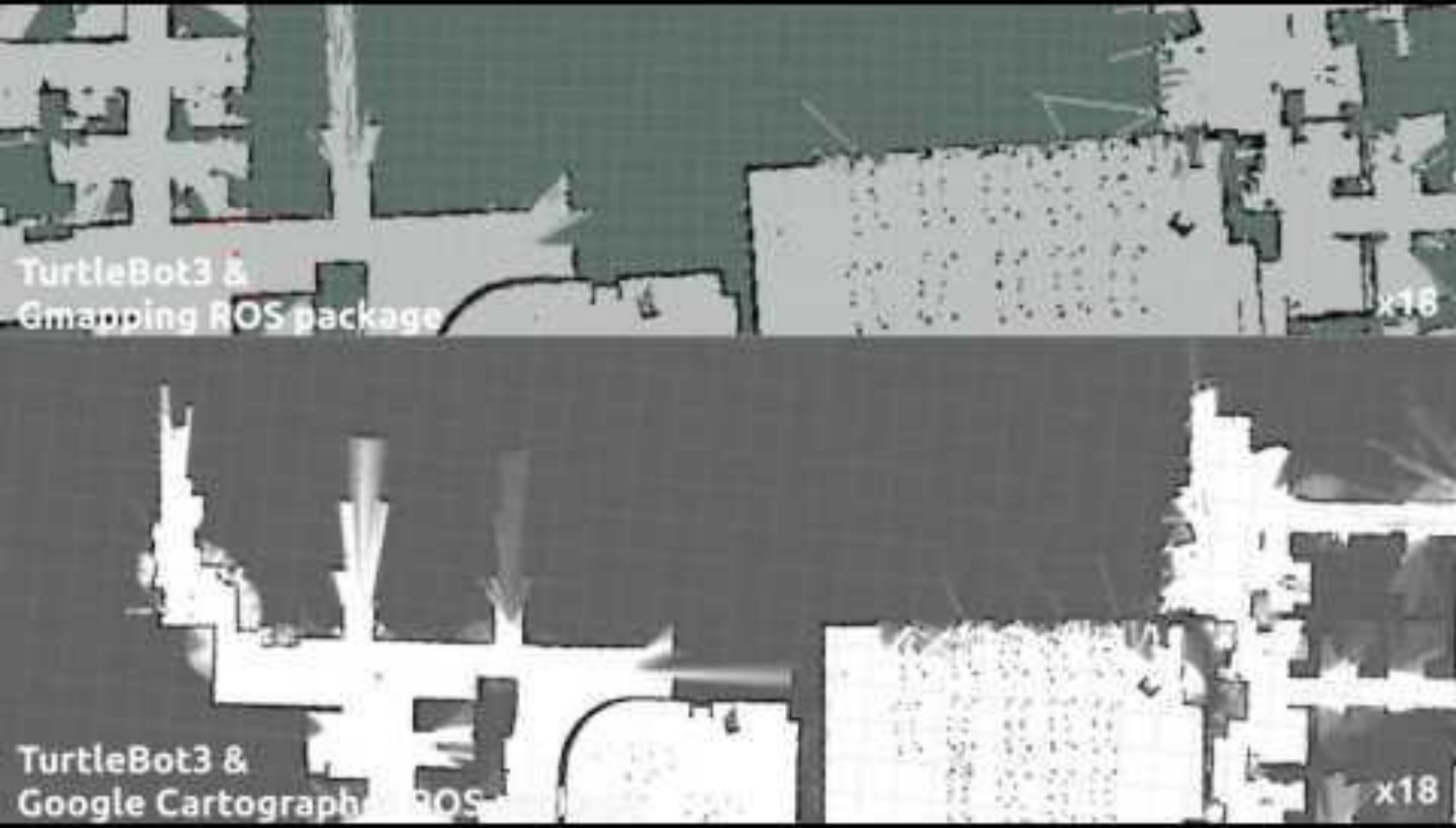


Nodes and Topics Required for SLAM

- sensor_node - this node runs the LDS sensor and sends the ‘scan’ information, which is necessary for SLAM, to the ‘slam_gmapping’ node.
- turtlebot3_teleop - this node is a node that can receive the keyboard input and control the robot.
- turtlebot3_core - this node receives the translation and rotation speed command and moves the robot.
- turtlebot3_slam_gmapping - this node creates a map based on the scan information from the distance measuring sensor and the tf information, which is the pose value of the sensor.
- map_saver - this node in the ‘map_server’ package creates a ‘map.pgm’ file and a ‘map.yaml’ file for the map.

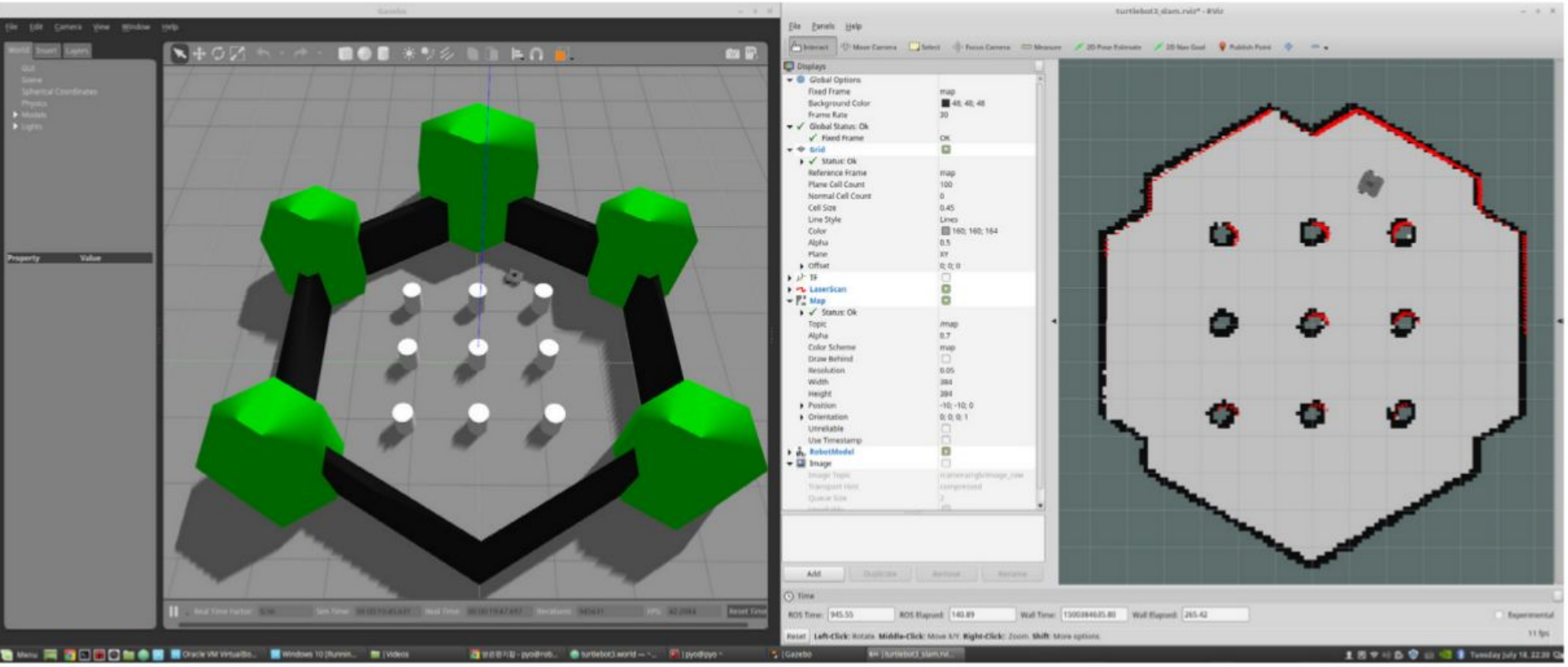


SLAM with GMapping



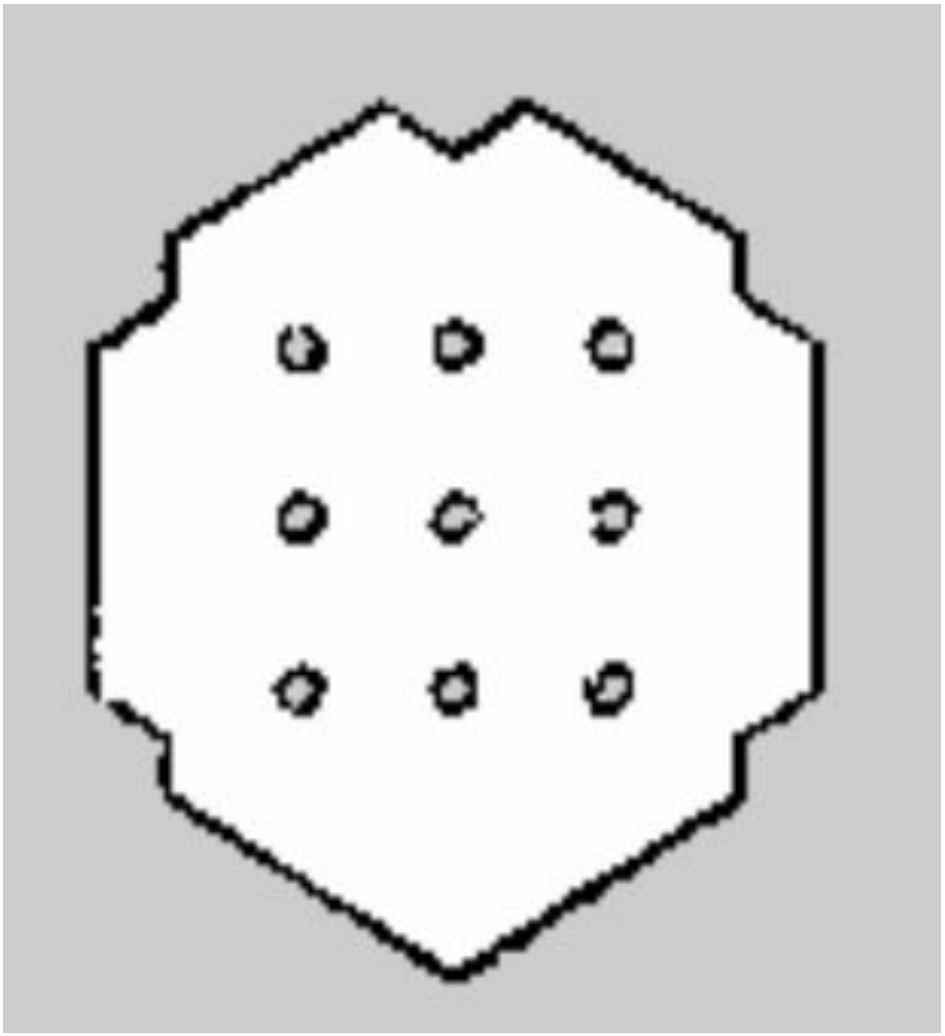
Activity: Virtual SLAM with Gmapping

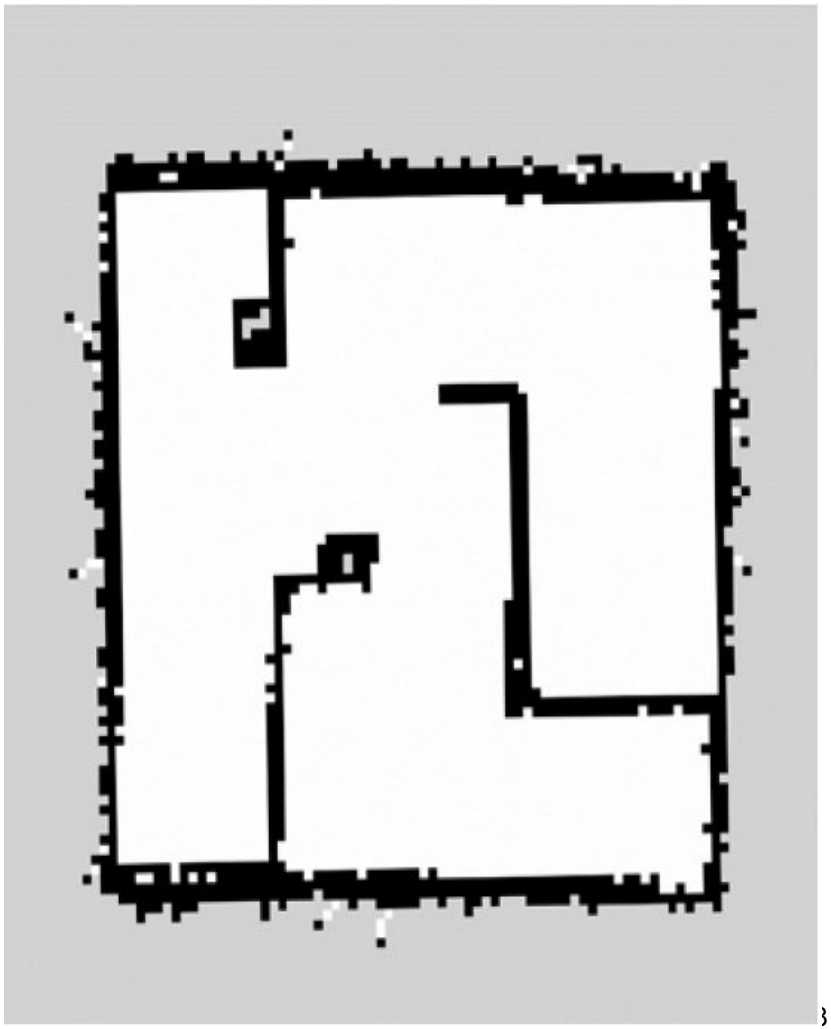
(terminal 1) roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2) roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
(terminal 3) roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
(terminal 4) rosrun map_server map_saver -f ~/my_map



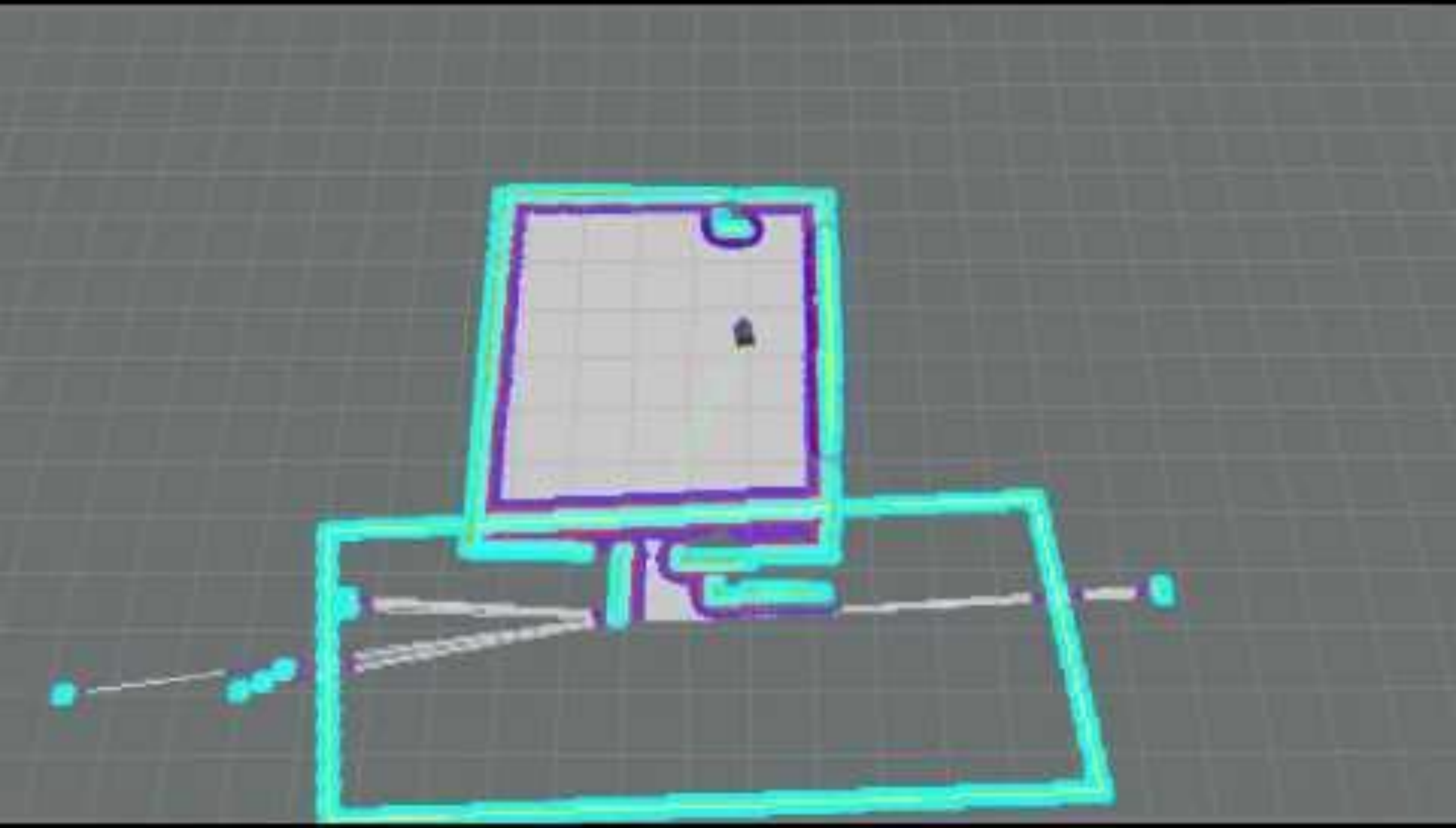
Map

- The map is saved to HOME folder
- It consists of two files: pgm and yaml





SLAM with Frontier Exploration



Activity: Frontier Exploration SLAM

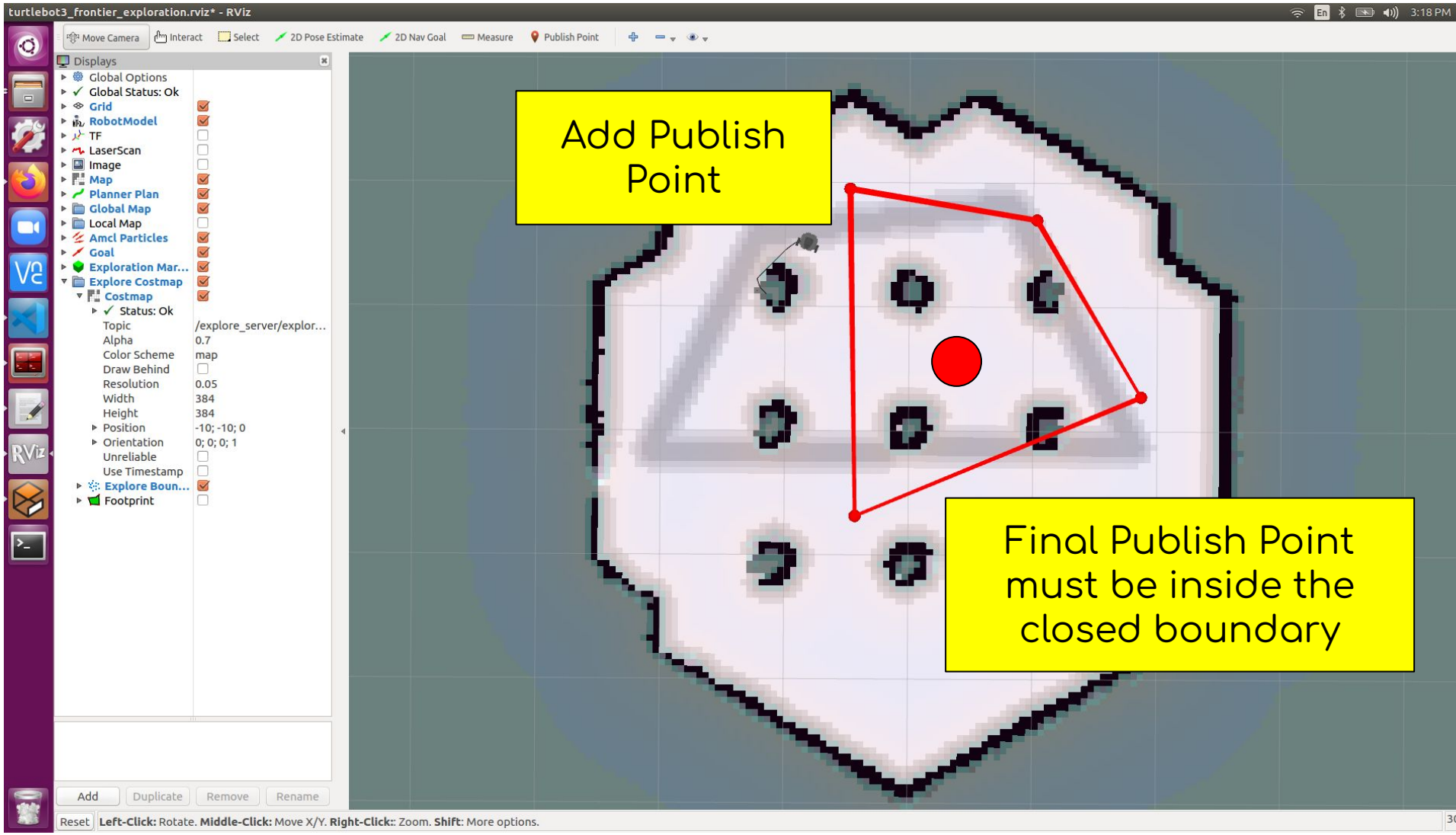
(terminal 1) roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2) roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=frontier_exploration
(terminal 3) rosrun map_server map_saver -f ~/my_map4

- Click on "Publish Point" button and click the points (you have to click the button for each point) on the map (illustrated on next slide) to indicate what is the area of exploration you want the robot to explore
- After closing the boundary box, click on "Publish Point" button and indicate anywhere within the boundary box (preferably in front of the robot) for the robot to start exploring at
- If there is a problem to launch frontier exploration, pls install as follows:

$ sudo apt install ros-kinetic-frontier-exploration ros-kinetic-navigation-stage

*** Deprecated since Melodic. ***

Activity: Frontier Exploration SLAM



Robot Navigation

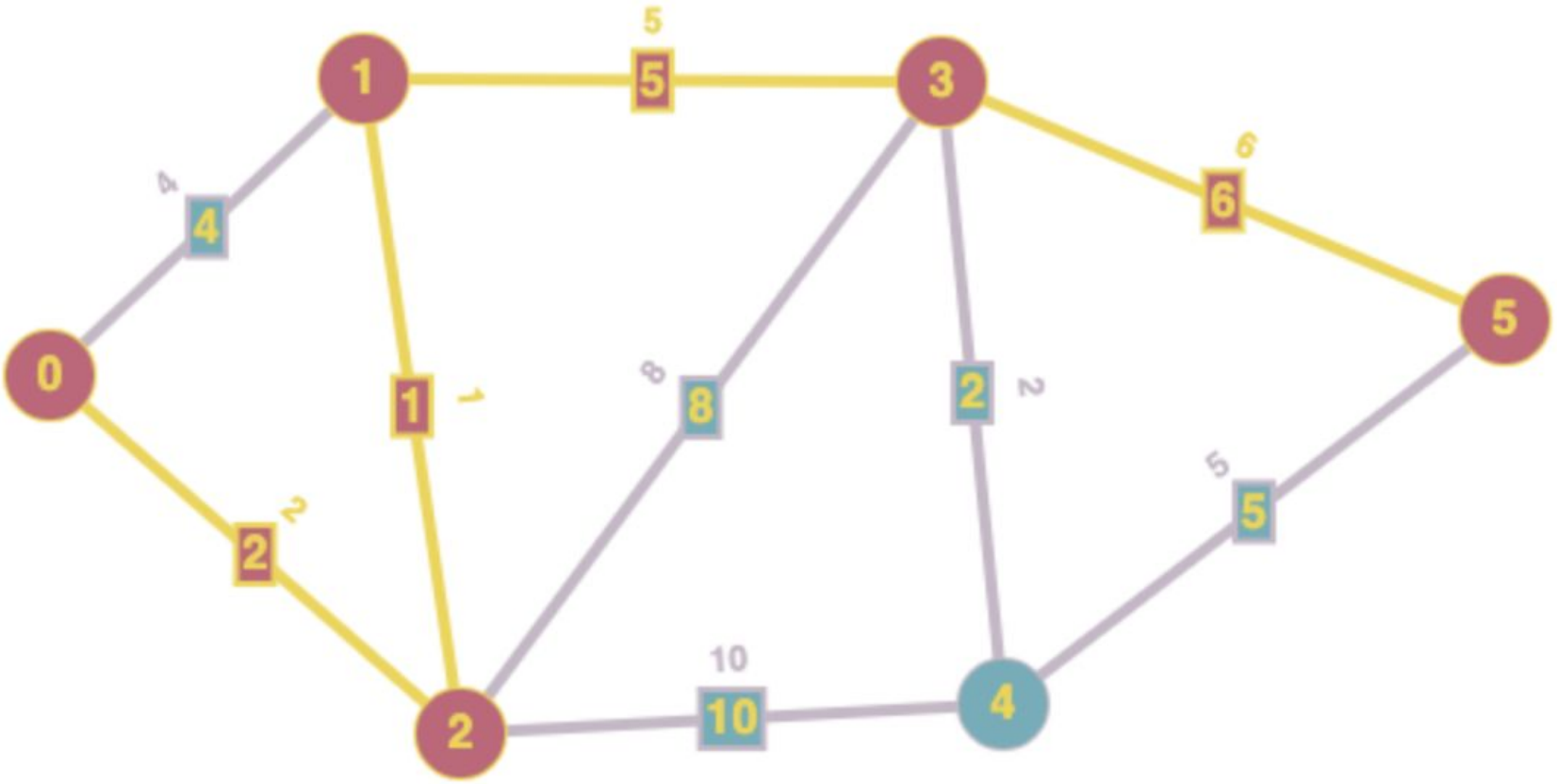
- Navigation is to move the robot from one location to the specified destination in a given environment.
- For this purpose, a map that contains geometry information of furniture, objects, and walls of the given environment is required. As described in the previous SLAM section, the map was created with the distance information obtained by the sensor and the pose information of the robot itself.
- The navigation enables a robot to move from the current pose to the designated goal pose on the map by using the map, robot's encoder, inertial sensor, and distance sensor.

Path Finding Algorithms

- There are many algorithms that perform path finding such as
  - Dijkstra's algorithm - Instead of exploring all possible paths equally, it favors lower cost path
  - A modification of Dijkstra's algorithm that is optimized for a single destination.
  - Potential Field
  - Particle Filter
  - RRT (Rapidly-exploring Random

Activity: Path Finding

- Use the following online tool to draw a graph as follows https://graphonline.ru/en/
- Determine the shortest path from 0 to 5 using Dijkstra's algorithm

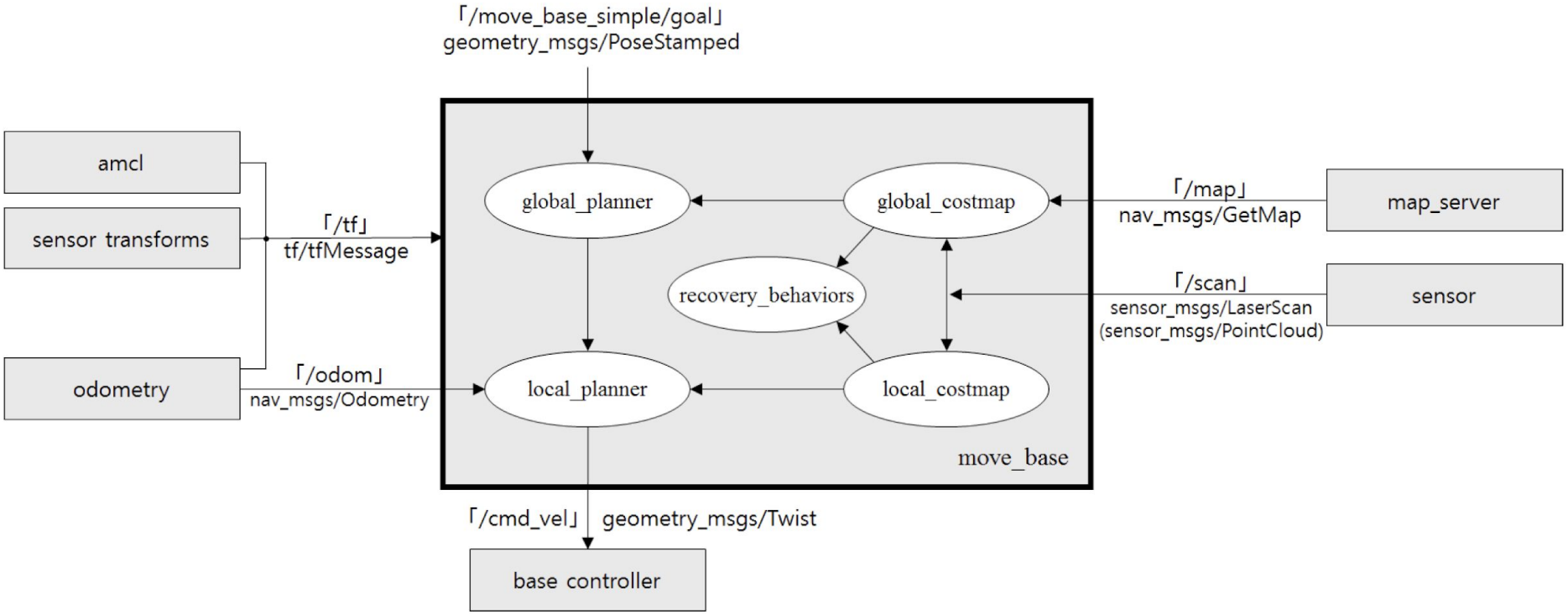


Navigation Demo

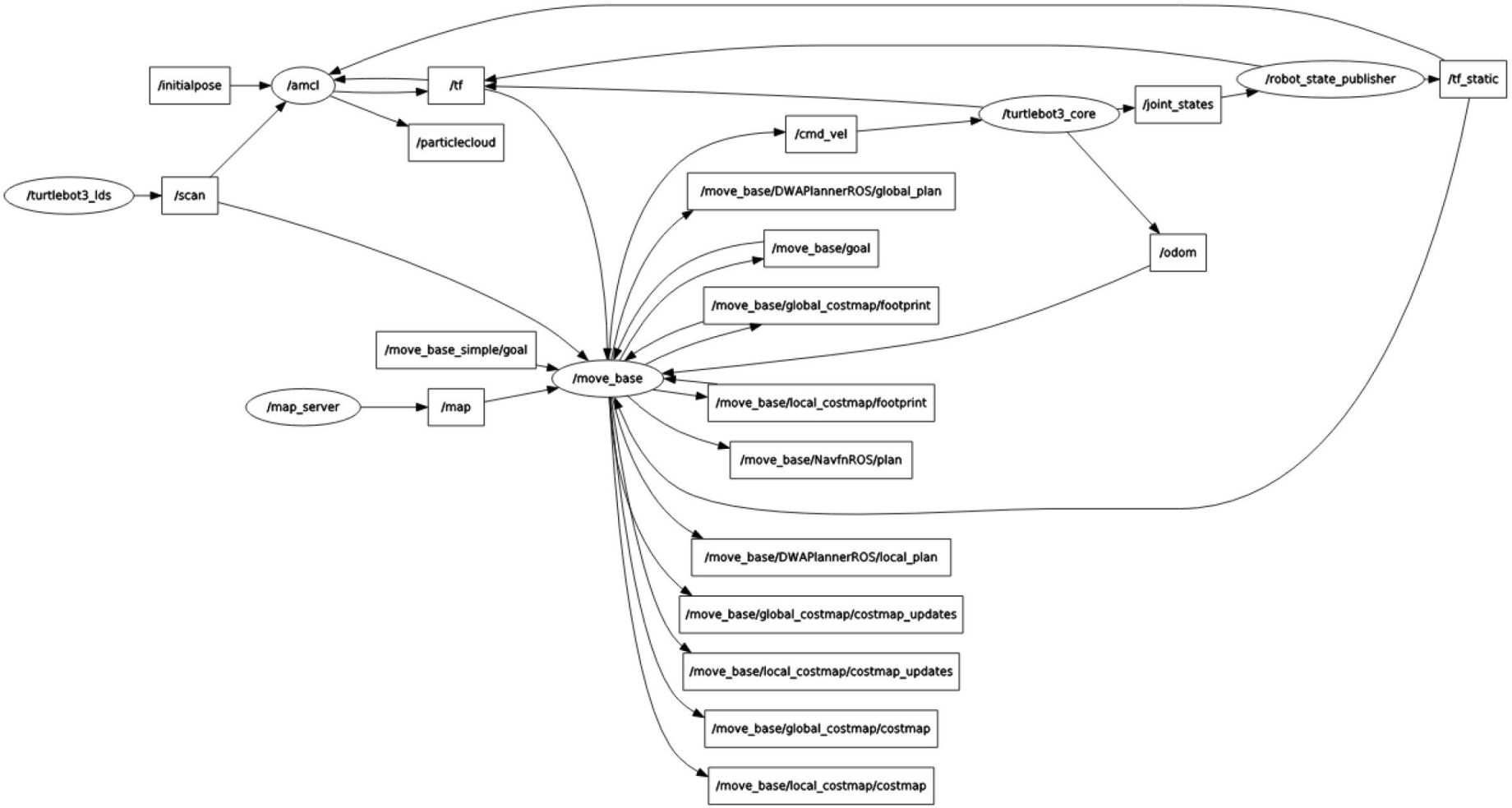
Navigation Demo

Robot Navigation with ROS

- The figure illustrates the relationship between the essential nodes and topics to run the ROS navigation package.
- The robot's odometry information is used in local path planning by receiving information such as the current speed of the robot to generates a local path or to avoid obstacles.
- Since the pose of the robot sensor changes depending on the hardware configuration of the robot, ROS uses relative coordinate transformation (TF).

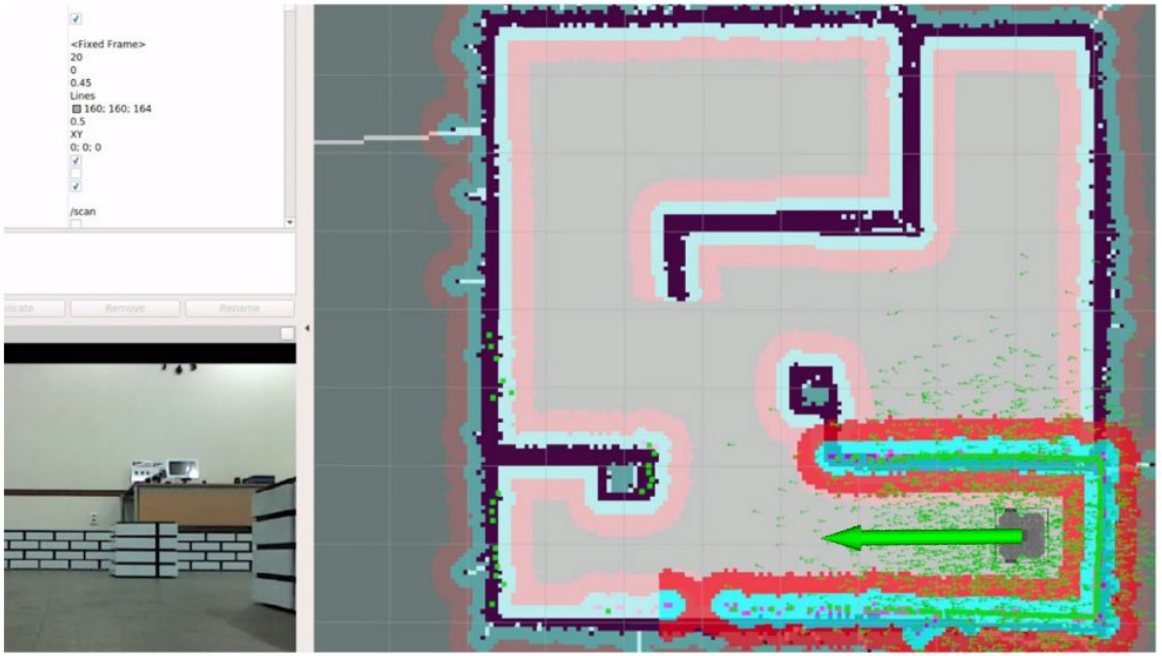


Node and Topic of Turtlebot3 Navigation



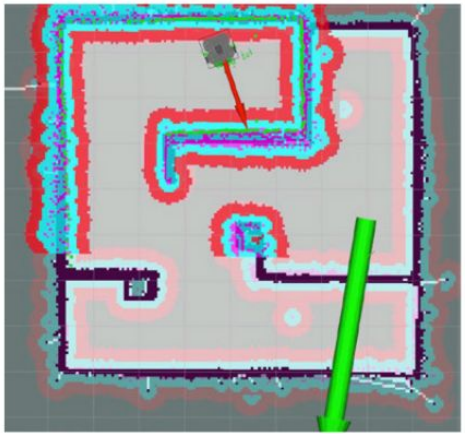
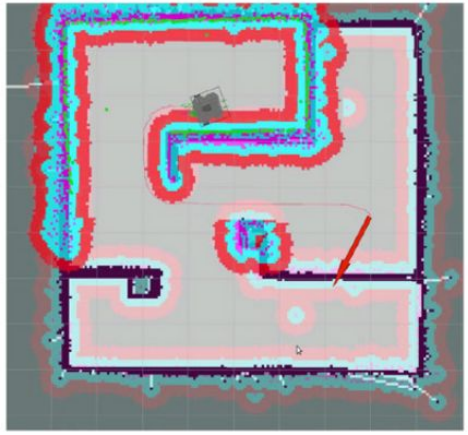
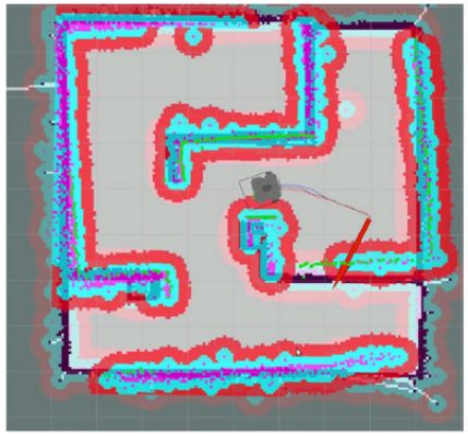
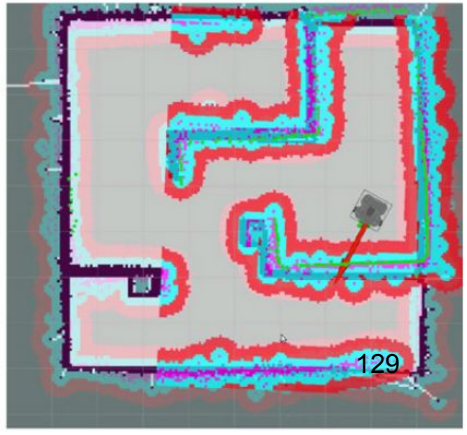
Estimate Initial Pose

- First, the initial pose estimation of the robot should be performed.
- When you press 2D Pose Estimate in the menu of RViz, a very large green arrow appears. Move it to the pose where the actual robot is located in the given map, and while holding down the left mouse button, drag the green arrow to the direction where the robot's front is facing, follow the instruction below.
  - Click the 2D Pose Estimate button.
  - Click on the approximate point in the map where the TurtleBot3 is located and drag the cursor to indicate the direction where TurtleBot3 faces.



Send Navigation Goal

- When everything is ready, let's try the move command from the navigation GUI. If you press 2D Nav Goal in the menu of RViz, a very large green arrow appears.
- This green arrow is a marker that can specify the destination of the robot. The root of the arrow is the x and y position of the robot, and the orientation pointed by the arrow is the theta direction of the robot.
- Click this arrow at the position where the robot will move, and drag it to set the orientation like the instruction below.
  - Click the 2D Nav Goal button.
  - Click on a specific point in the map to set a goal position and drag the cursor to the direction where TurtleBot should be facing at the end.
- The robot will create a path to avoid obstacles to its destination based on the map. Then, the robot moves along the path. At this time, even if an obstacle is suddenly detected, the robot moves to the target point avoiding the obstacle.

Costmap

- In navigation, costmap calculates obstacle area, possible collision area, and a robot movable area based on the aforementioned four factors.

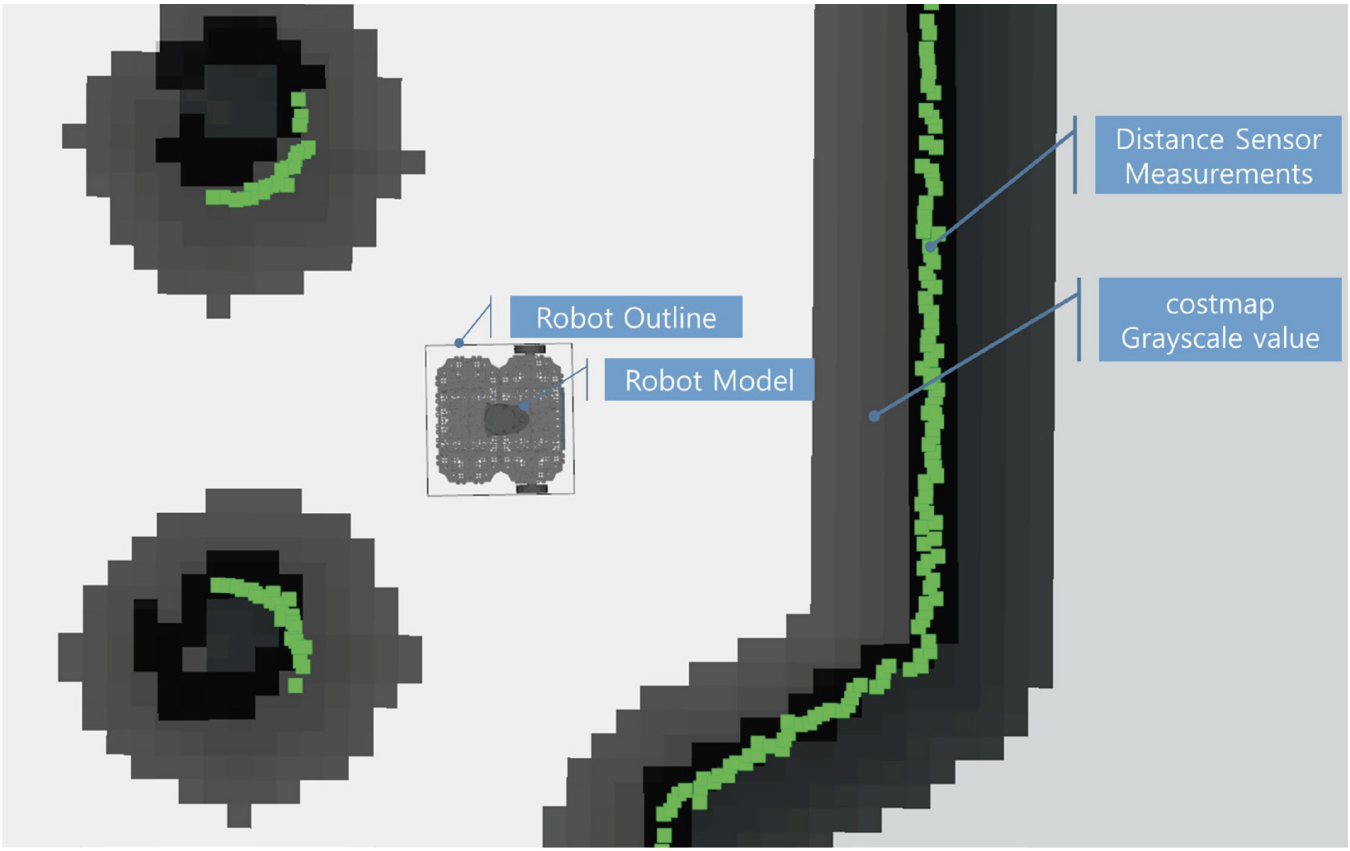
- Depending on the type of navigation, costmap can be divided into two. One is the 'global_costmap', which sets up a path plan for navigating in the global area of the fixed map. The other is 'local_costmap' which is used for path planning and obstacle avoidance in the limited area around the robot.

- Although their purposes are different, both costmaps are represented in the same way.

- The costmap is expressed as a value between '0' and '255'.
  - 000: Free area where robot can move freely
  - 001~127: Areas of low collision probability
  - 128~252: Areas of high collision probability
  - 253~254: Collision area
  - 255: Occupied area where robot can not move

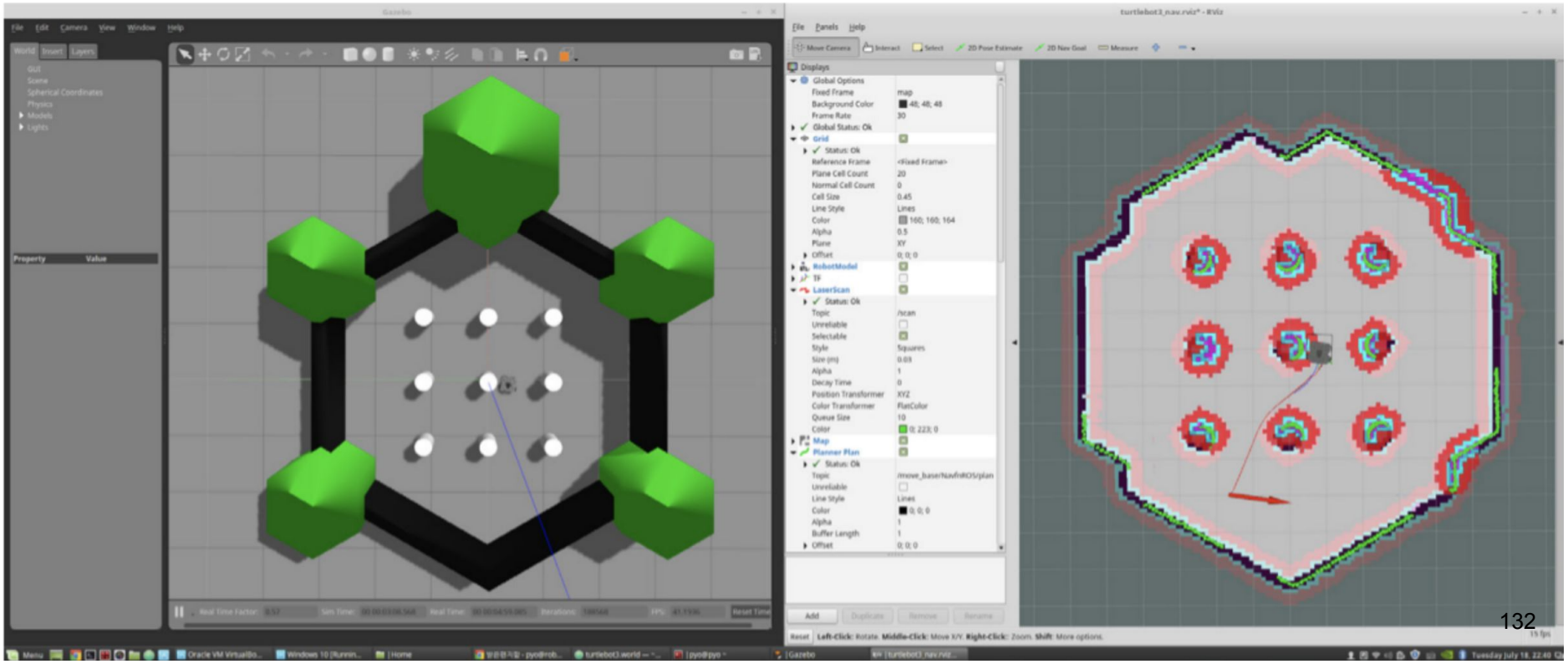
Costmap Representation

- There is a robot model in the middle and the rectangular box around it corresponds to the outer surface of the robot. When this outline line contacts the wall, the robot will bump into the wall as well.
- Green represents the obstacle with the distance sensor value obtained from the laser senso.
- As the gray scale costmap gets darker, it is more likely to be collision area



Activity: Virtual ROS Navigation

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/my_map.yaml



Initial Pose Estimate Code Explanation

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import PoseWithCovarianceStamped
from tf.transformations import quaternion_from_euler
from nav_msgs.msg import Odometry

def callback(msg):
        print msg.pose.pose

        global px, py, pz, ox, oy, oz, ow
        px = msg.pose.pose.position.x
        py = msg.pose.pose.position.y
        pz = msg.pose.pose.position.z

        ox = msg.pose.pose.orientation.x
        oy = msg.pose.pose.orientation.y
        oz = msg.pose.pose.orientation.z
        ow = msg.pose.pose.orientation.w
rospy.init_node('init_pos')
odom_sub = rospy.Subscriber("/odom", Odometry, callback)
pub = rospy.Publisher('/initialpose', PoseWithCovarianceStamped, queue_size = 10)
rospy.sleep(3)
checkpoint = PoseWithCovarianceStamped()
checkpoint.pose.pose.position.x = px
checkpoint.pose.pose.position.y = py
checkpoint.pose.pose.position.z = pz
[x,y,z,w]=quaternion_from_euler(0.0,0.0,0.0)
checkpoint.pose.pose.orientation.x = ox
checkpoint.pose.pose.orientation.y = oy
checkpoint.pose.pose.orientation.z = oz
checkpoint.pose.pose.orientation.w = ow
print checkpoint
pub.publish(checkpoint)
```

To estimate the initial pose, subscribe the pose data from odom topic and publish the location of the TB3

Activity: Initial Pose Estimate

- Instead of manual estimate the initial pose, you can use odom data to estimate the initial pose as follows:

(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
~~$ roslaunch autonomous initial_pose2.launch map_file :=$HOME/my_map.yaml~~
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/my_map.yaml
(terminal 3)
$ rosrun my_robotics initial_pose.py

- Observe in the RViz application that the TB3's pose estimate will be automatically executed

Move Turtlebot Code Explanation

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

def talker():
    rospy.init_node('vel_publisher')
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    move = Twist()
    rate = rospy.Rate(1)
    while not rospy.is_shutdown():
        move.linear.x = 0.8
        move.angular.z = 0.8
        pub.publish(move)
        rate.sleep()

if __name__== '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

To move the robot, create a Publisher node

Activity: Move TB3 from Python Code

(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
(terminal 3)
$ rosrun my_robotics trajectory.py

Get Scan Data Code Explanation

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan

def callback(msg):
        print('s1 [0]')
        print msg.ranges[0]
        print('s2 [90]')
        print msg.ranges[90]
        print('s3 [180]')
        print msg.ranges[180]
        print('s4 [270]')
        print msg.ranges[270]
        print('s5 [359]')
        print msg.ranges[359]

rospy.init_node('laser_data')
sub = rospy.Subscriber('scan', LaserScan, callback)
rospy.spin()
```

To collect scan data, create a Subscriber node

Activity: Get Laser Scan Data

(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
(terminal 3)
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
(terminal 4)
$ rosrun my_robotics laser_data.py

Avoid Obstacle Code Explanation

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
def callback(msg):
        print('s1 [0]')
        print msg.ranges[0]

        if msg.ranges[0] > 0.5:
                move.linear.x = 0.3
                move.angular.z = 0.0
        else:
                move.linear.x = 0.0
                move.angular.z = 0.0
        pub.publish(move)

rospy.init_node('obstacle_avoidance')
sub = rospy.Subscriber('/scan', LaserScan, callback)
pub = rospy.Publisher('/cmd_vel', Twist)
move = Twist()
rospy.spin()
```

To avoid obstacle, create a Subscriber node to collect scan data, and a Publisher node to move the robot

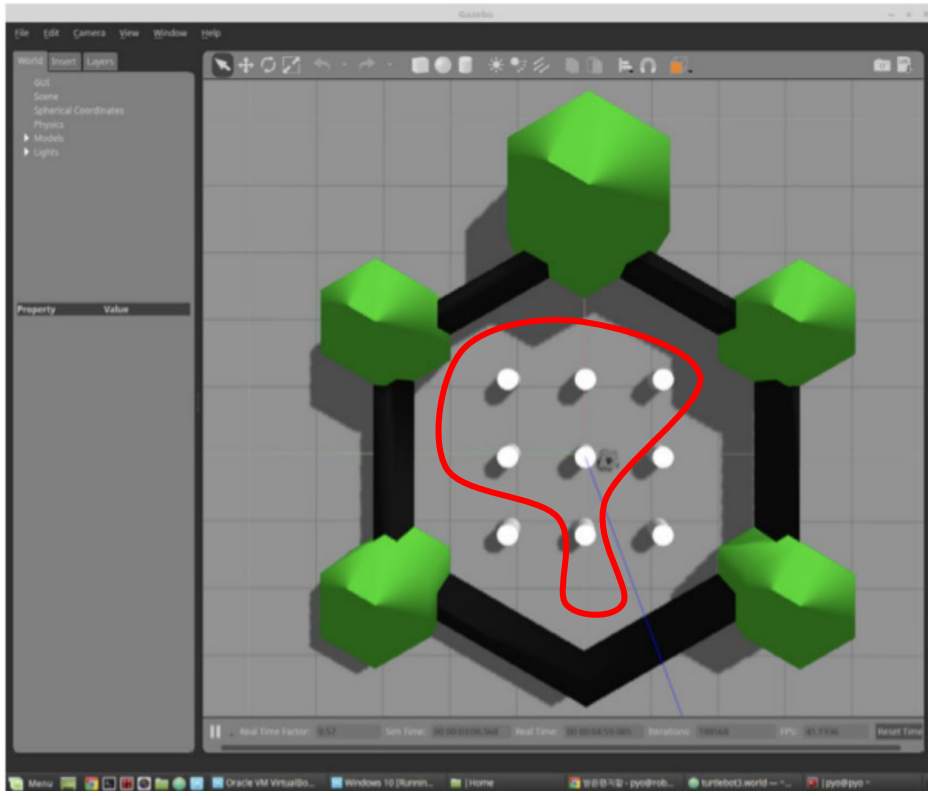
Activity: Avoid Obstacle

(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
(terminal 3)
$ rosrun my_robotics avoid_obstacle.py

Activity: Path Planning

- You can get the turtlebot to move in a sequence of waypoints as follows:

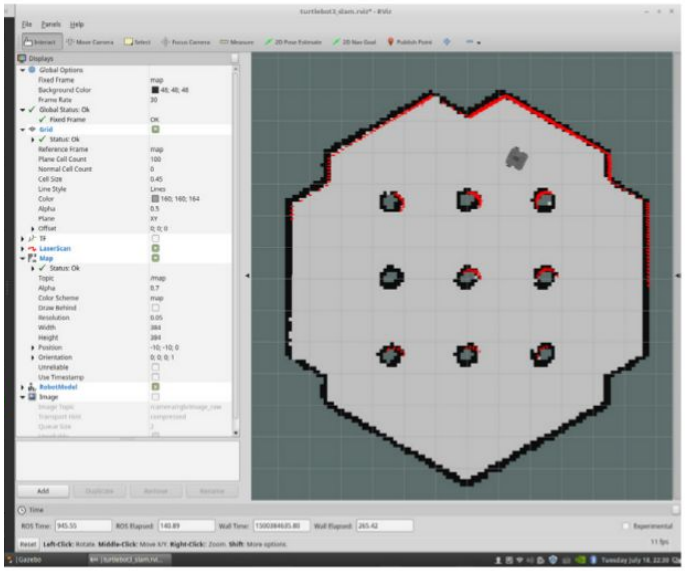
(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/my_map.yaml
(terminal 3)
$ rosrun my_robotics initial_pose.py
$ rosrun my_robotics path_planning.



Activity: Autonomous SLAM

- You can also let the turtlebot to move in a sequence of waypoints

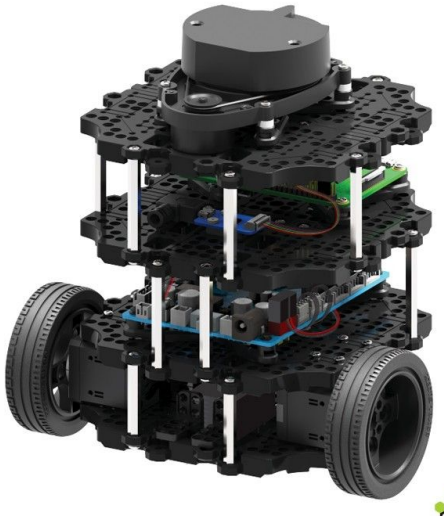
(terminal 1)
roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
(terminal 3)
$ rosrun my_robotics autonnomous_exploring.py
(terminal 4) rosrun map_server map_saver -f ~/my_map2



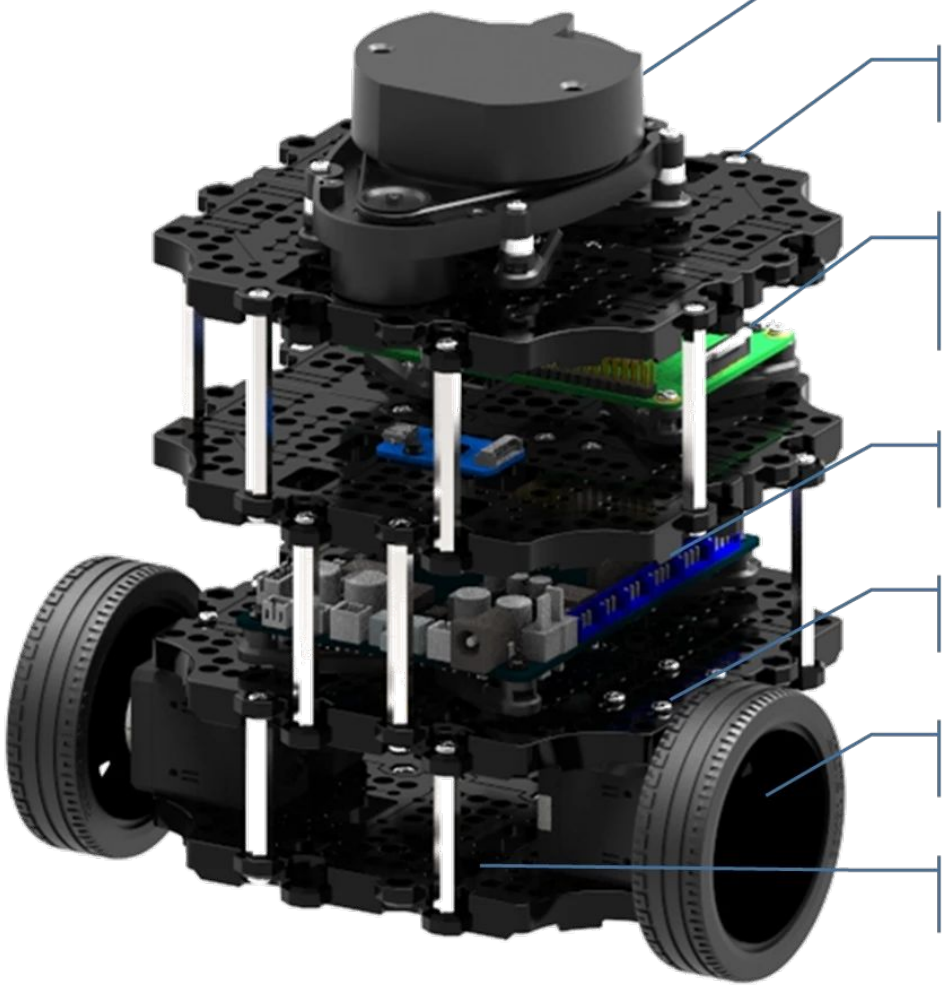
Topic 4
Robotics System Integration

TurtleBot 3 Burger

- TurtleBot3 Burger is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping.
- The TurtleBot3's core technology is SLAM, Navigation and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around your room.



Turtlebot 3 Burger

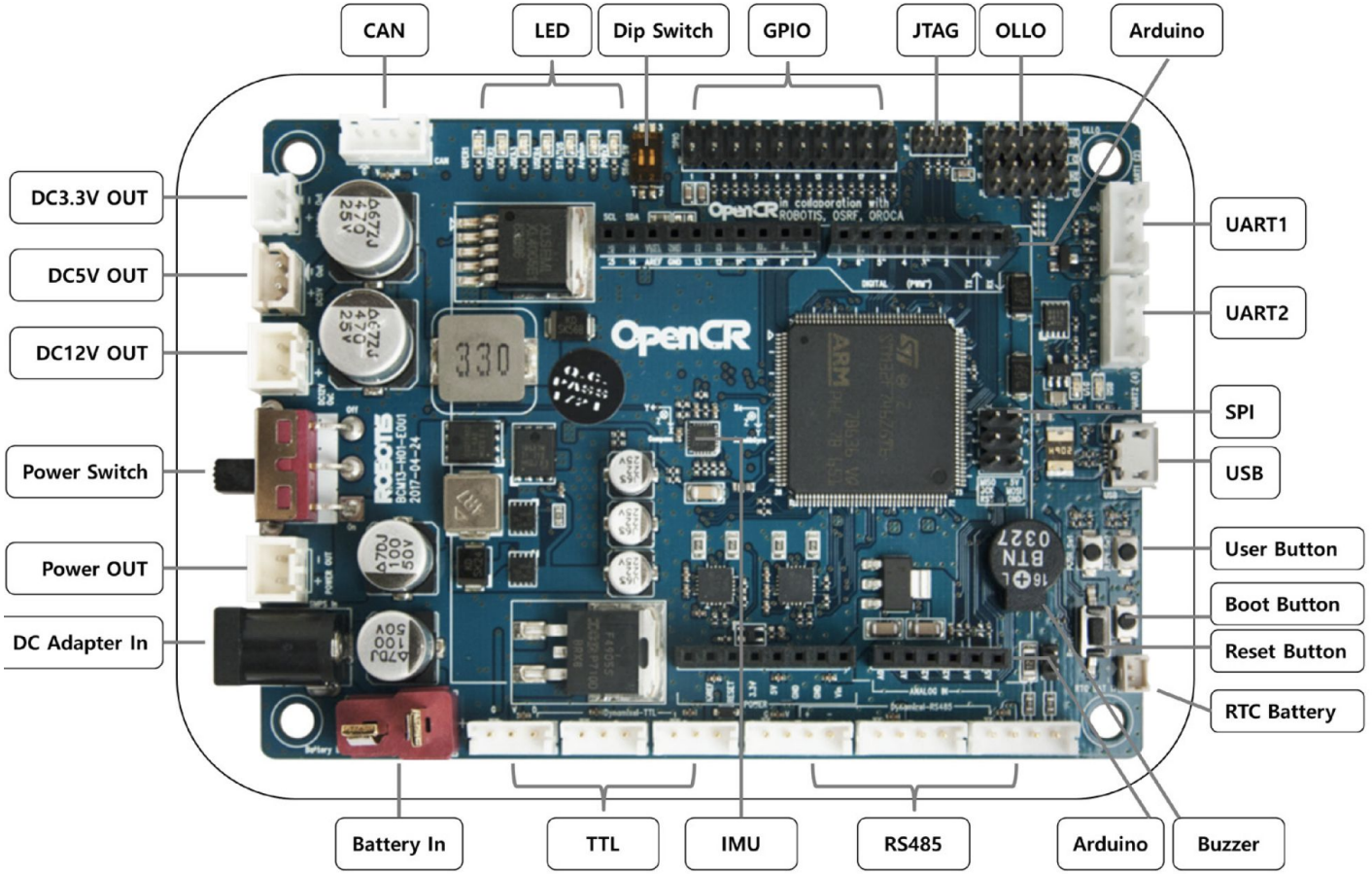


- 360° LiDAR for SLAM & Navigation
- Scalable Structure
- Single Board Computer (Raspberry Pi)
- OpenCR (ARM Cortex-M7)
- DYNAMIXEL x 2 for Wheels
- Sprocket Wheels for Tire and Caterpillar
- Li-Po Battery 11.1V 1,800mAh

Turtlebot 3 Burger Accessories

OpenCR Module

OpenCR (Open-source Control Module for ROS)3 is an embedded board that supports ROS and is used as the main controller of TurtleBot3.

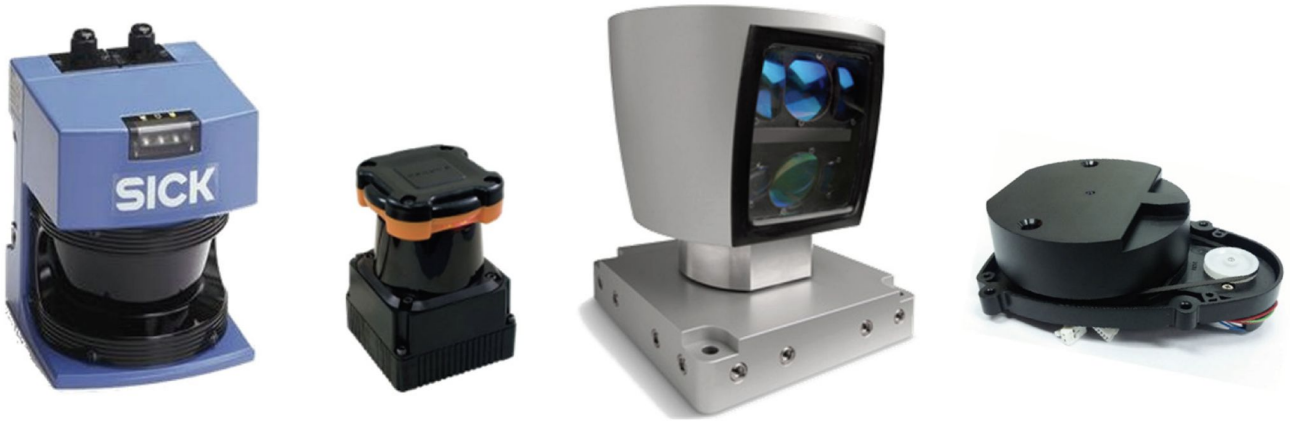


Characteristics of OpenCR Module

- High Performance
  - ST's STM32F746 chip used in OpenCR is a high-performance microcontroller running at up to 216MHz with the Cortex-M7 core at the top of ARM microcontrollers.
- Arduino Support
  - OpenCR is easy to use by using the Arduino IDE7. The OpenCR provides Arduino
- Various Interfaces
  - OpenCR supports both TTL and RS485 communication, which are default interfaces for Dynamixel from ROBOTIS. In addition, the board supports UART, SPI, I2C, CAN communication interface as well as additional GPIO pins
- IMU Sensor
  - OpenCR includes MPU925010 chip, which is integrated triple-axis gyroscope, triple-axis accelerometer, and triple-axis magnetometer sensor in one chip, therefore, various applications using IMU sensor can be used without adding a sens

Laser Distance Sensor

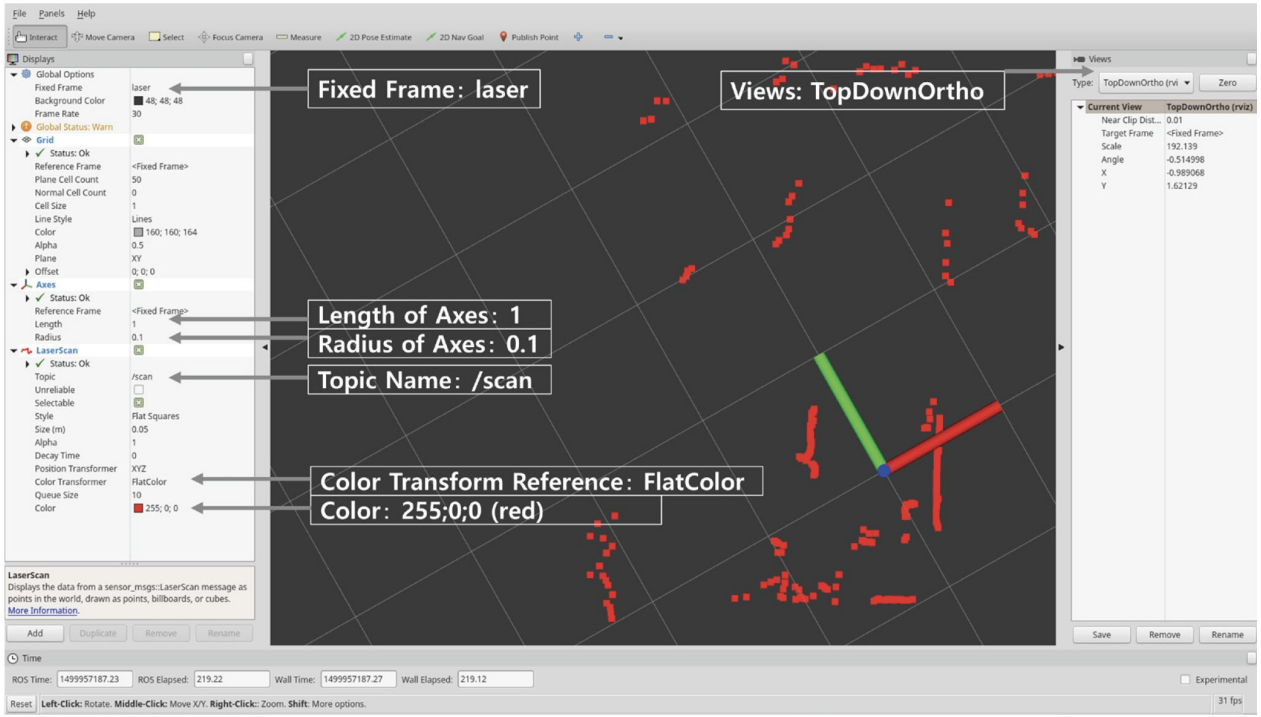
- Laser Distance Sensors (LDS) are referred to various names such as Light Detection And Ranging (LiDAR), Laser Range Finder (LRF) and Laser Scanner. LDS is a sensor used to measure the distance to an object using a laser as its source.
- The LDS sensor has the advantage of high performance, high speed, real time data acquisition, so it has a wide range of applications in relation to distance measurement.
- This is a sensor widely used in the field of robots for recognition of objects and people, and distance sensor based SLAM (distance-based sensor), and also widely used in unmanned vehicles due to its real time data acquisition.
- There are also rplidar which supports RPLIDAR and 'hls_lfcd_lds_driver' which supports LDS of TurtleBot 3.



Activity: LDS Test

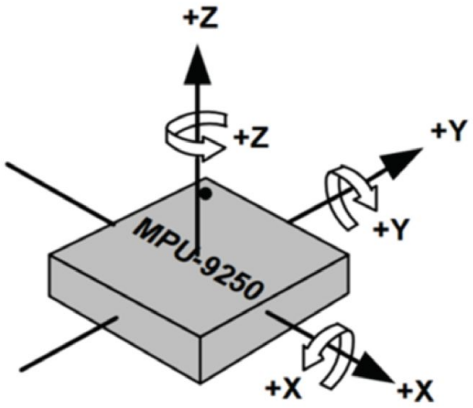
- We are going to test using LDS (HLS-LFCD2) from HLDS (Hitachi-LG Data Storage), so you must install 'hls_lfcd_lds_driver' package
- Execute the following commands to the scan data

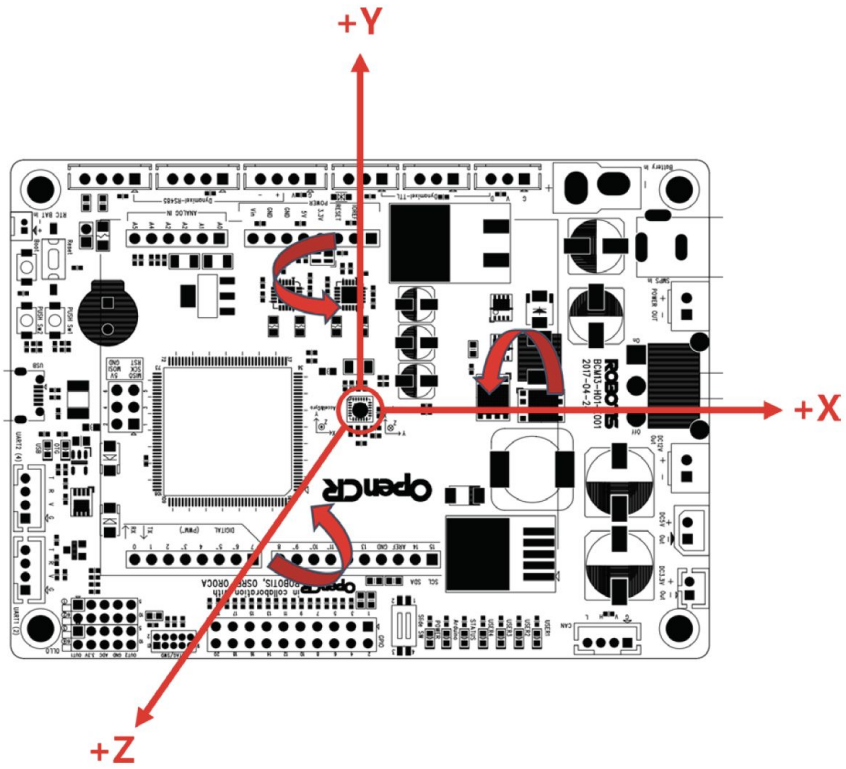
  sudo apt-get install ros-kinetic-hls-lfcd-lds-driver
  roslaunch hls_lfcd_lds_driver hlds_laser.launch
  rostopic echo /scan
  rviz



IMU Sensor

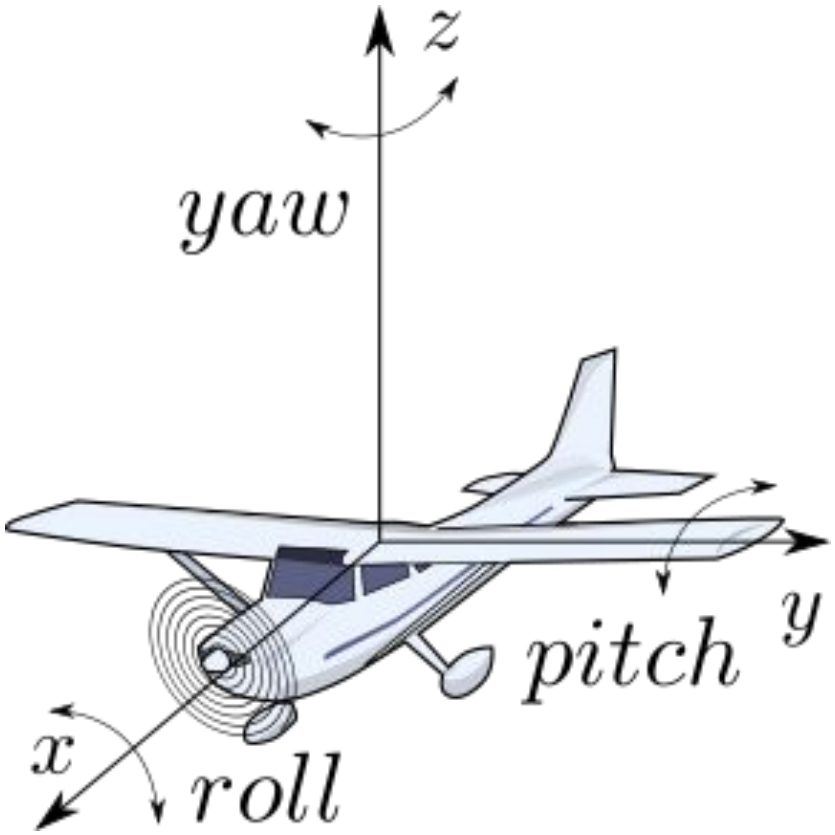
- MPU9250 sensor from InvenSense is located at the center of the OpenCR board for accurate measurement.
- The MPU9250 has built-in gyroscope, accelerometer, and magnetometer sensors on one chip.

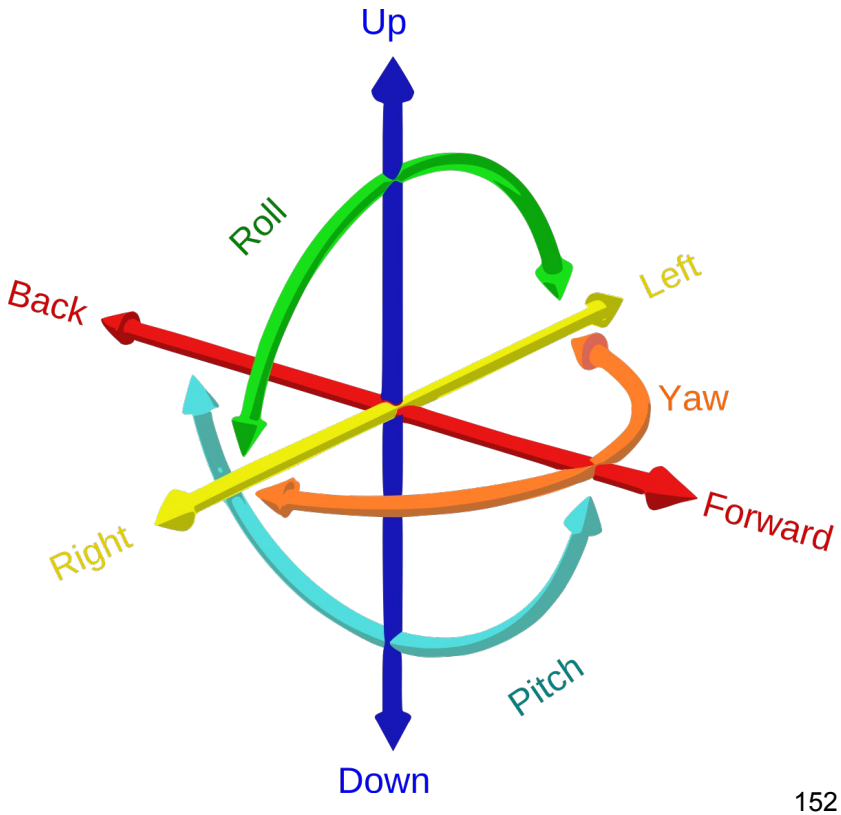




Inertial Measurement Units (IMU)

- There are totally six sequences for any physical rotation: R-xyz, R-yxz, R-zxy, R-zyx, R-xzy, R-yzx
- Different rotation sequence may result in the same posture, or saying that the same posture have different pitch and roll, that is why we have to define a default sequence: Yaw-Pitch-Roll(ψ-θ-ϕ)



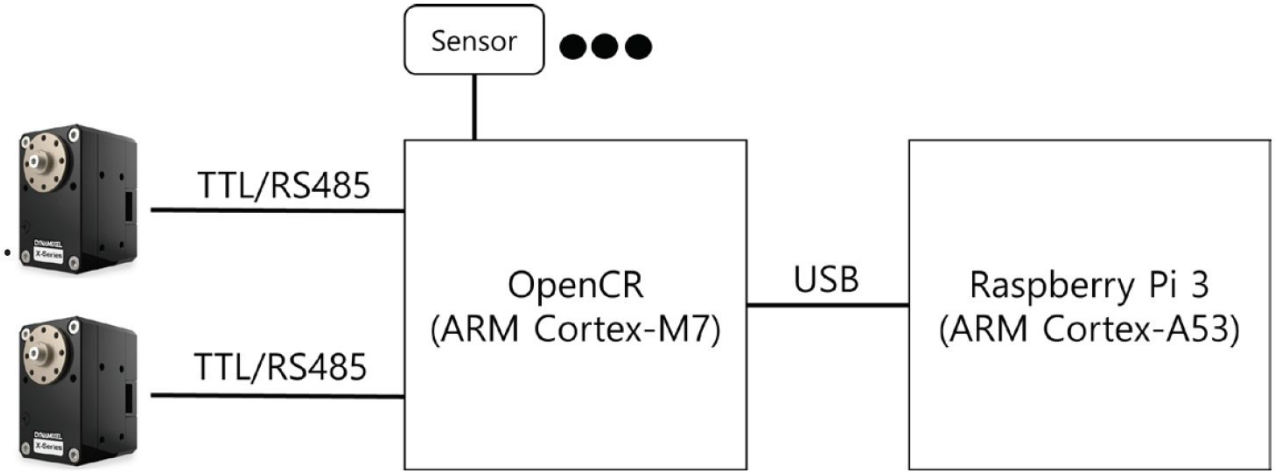


Dynamixel Actuator

- The Dynamixel is an integrated module that is composed of a reduction gear, a controller, a motor and a communication circuit.
- Dynamixel series offers feedbacks for position, speed, temperature, load, voltage and current data by using a daisy-chain method that enables simple wire connection between devices.

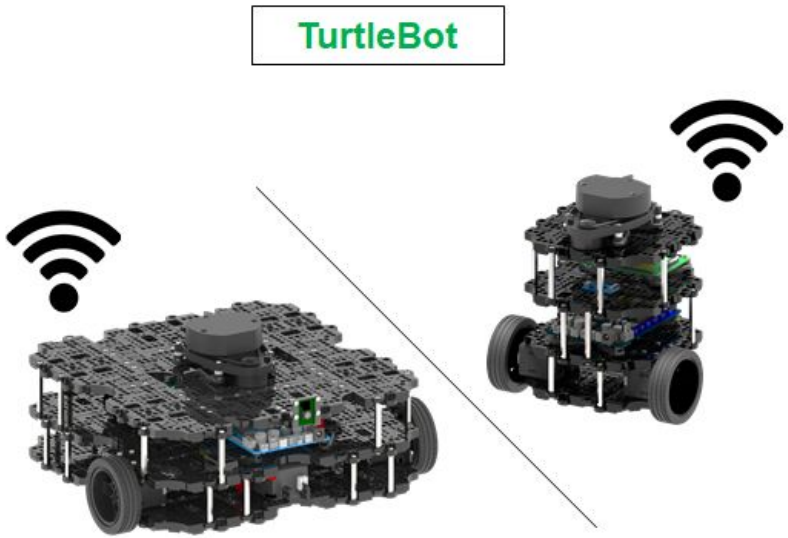
Turtlebot3 Embedded System

- Operating systems such as Linux do not guarantee real-time operation, and microcontrollers suitable for real-time control are required to control actuators and sensors.
- In case of TurtleBot3 Burger and Waffle Pi, a ARM Cortex-M7 series microcontroller is used for its actuator and sensor control, and the Raspberry Pi 3 board, which uses for Linux and ROS, is connected via USB



Setup Remote PC

- Install Ubuntu on Remote PC
- Install ROS 1 on Remote PC
- Install Dependent ROS 1 Packages
- Network Configuration



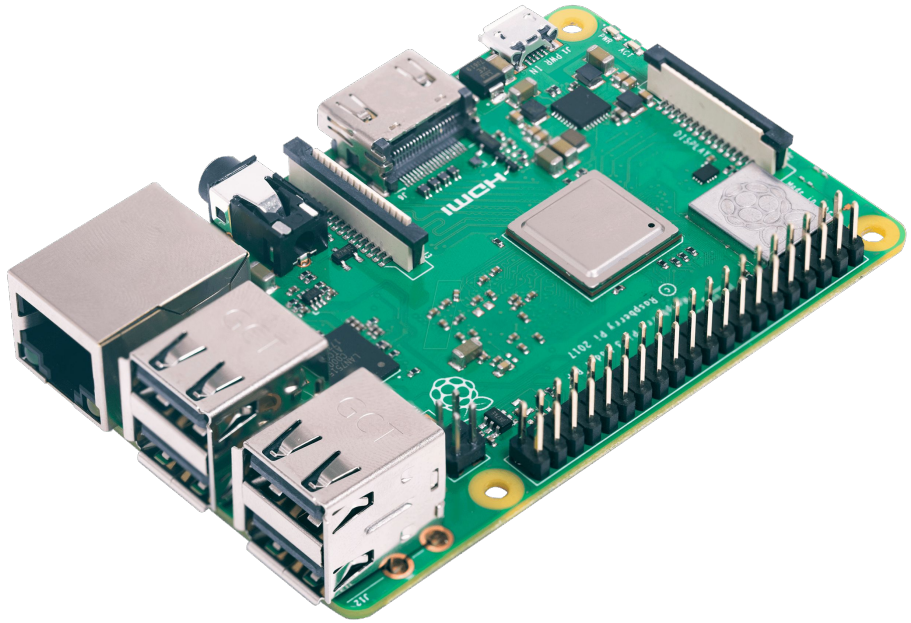
ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_TURTLEBOT

ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_REMOTE_PC

* Example when ROS Master is running on the Remote PC

Setup Turtlebot3 SBC

- Turtlebot uses Raspberry Pi 3 its Single Board Controller (SBC)
- To set up the Turtlebot3 SBC
  - Install Raspbian or Linux (Ubuntu MATE)
  - Install ROS on TurtleBot PC
  - Install Dependent Packages on TurtleBot PC
  - USB Settings
  - Network Configuration



Configure Network

- $ ifconfig
- $ nano ~/.bashrc
- $ source ~/.bashrc

```
enp3s0    Link encap:Ethernet  HWaddr d8:cb:8a:f3:d3:00
          inet addr:192.168.0.165  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::b5ed:414a:b396:f212/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:137578 errors:0 dropped:0 overruns:0 frame:0
          TX packets:69670 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:122799864 (122.7 MB)  TX bytes:10093863 (10.0 MB)
          Interrupt:19

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:9381 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9381 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1811987 (1.8 MB)  TX bytes:1811987 (1.8 MB)

wlp2s0    Link encap:Ethernet  HWaddr ac:2b:6e:6d:08:ee
          inet addr:192.168.0.200  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::7a:7d5c:9ca8:bd9c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:627 errors:0 dropped:0 overruns:0 frame:0
          TX packets:802 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:142095 (142.0 KB)  TX bytes:115501 (115.5 KB)
```

```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

if [ -x /usr/bin/mint-fortune ]; then
    /usr/bin/mint-fortune
fi

alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.200

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell ^_ Go To Line
```

Bringup Turtlebot3

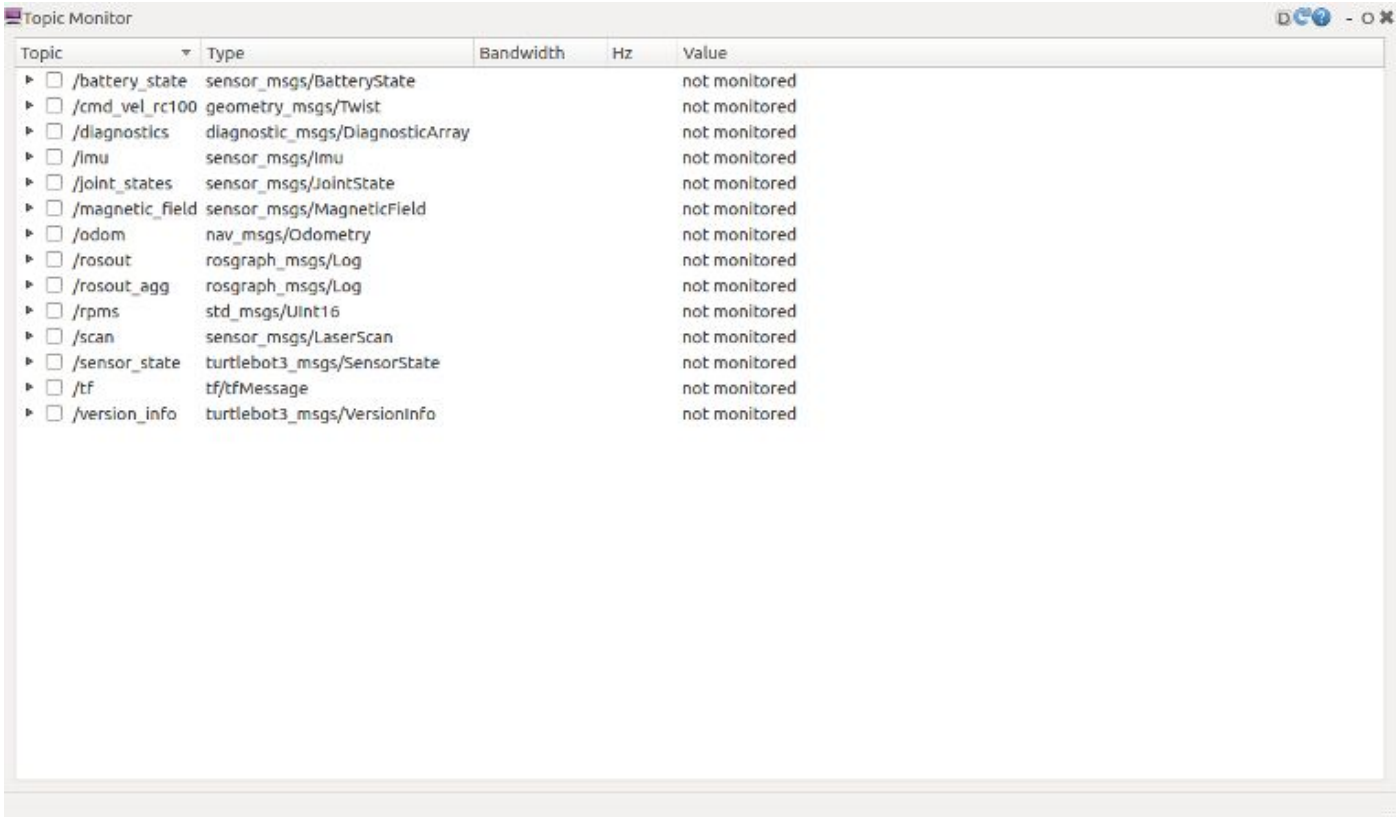
[terminal 1] $ roscore

[terminal 2]
$ ssh pi@192.168.3.121
$ password: turtlebot
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

Topic Monitoring

- You can troubleshoot the TurtleBot3 by monitoring the various topics such as odometry data, laser scan data, battery state
- To monitor the topics, you can use rqt provided by ROS. The rqt is a Qt-based framework for GUI development for ROS.
- To monitor the camera images, you can use rqt_image_view provided by ROS

[terminal 4] $ rqt



Teleoperation

- The TurtleBot3 can be teleoperated by various devices. It is tested with several wireless devices such as PS3, XBOX 360, ROBOTIS RC100 and etc.
- [terminal 5]
  roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

Obstacle Avoidance

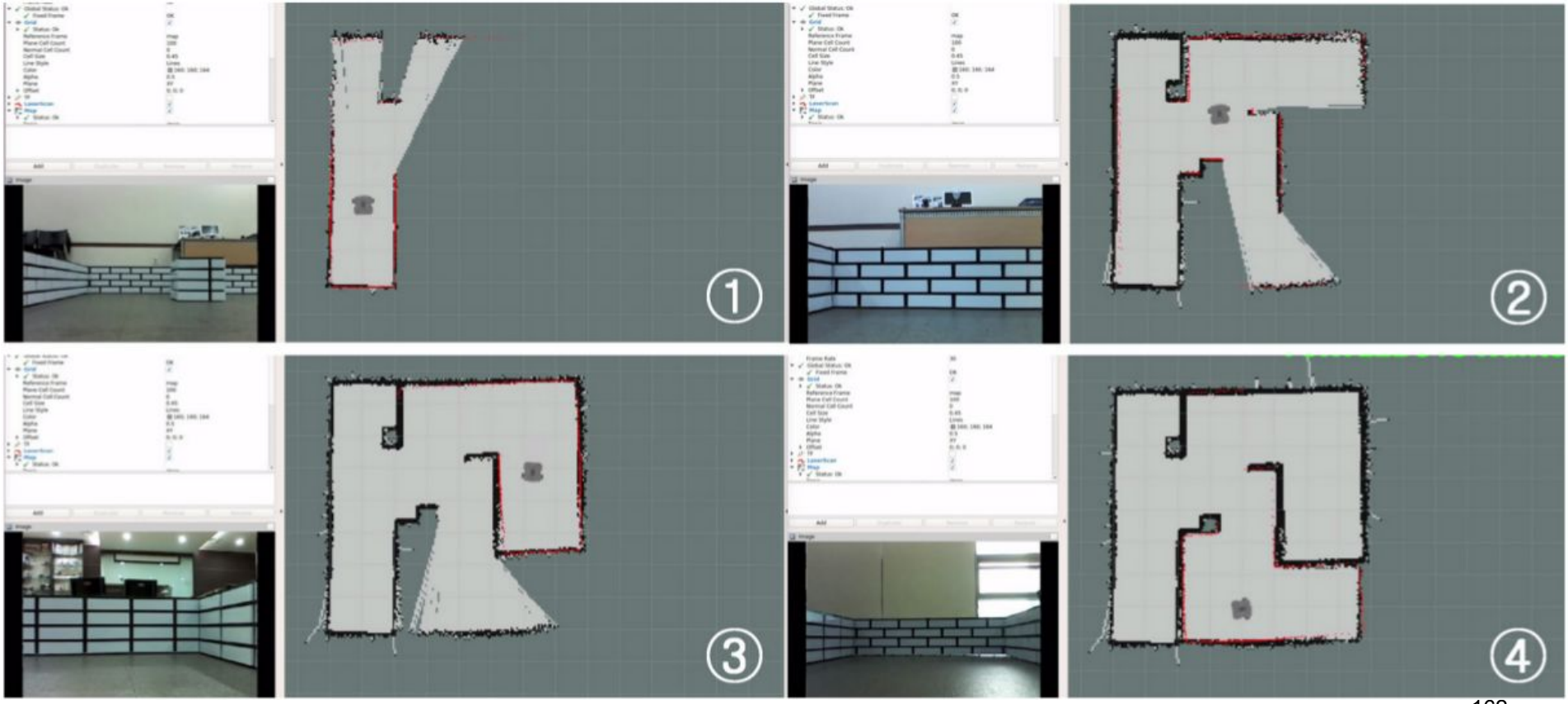
- The TurtleBot3 can be moved or stopped by LDS data. When the TurtleBot3 moves, it stops when it detects an obstacle ahead.

$ roslaunch turtlebot3_example turtlebot3_obstacle.launch

SLAM

- The SLAM (Simultaneous Localization and Mapping) is a technique to draw a map by estimating current location in an arbitrary space.

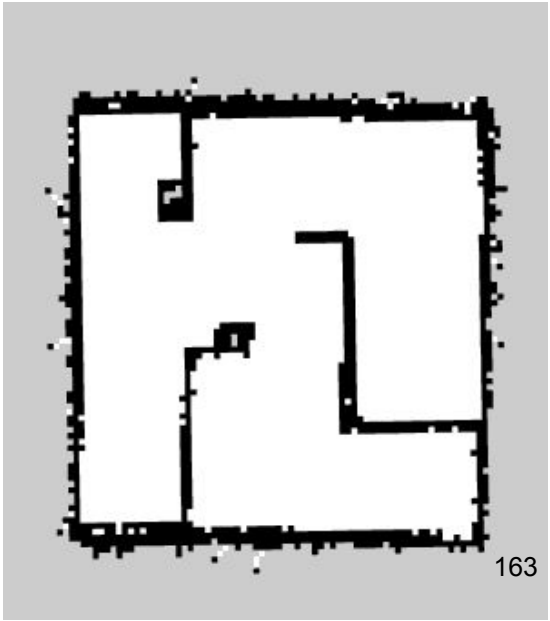
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



Save Map

- Now that you have all the work done, let's run the map_saver node to create a map file.
- The map is drawn based on the robot's odometry, tf information, and scan information of the sensor when the robot moves.
- These data can be seen in the RViz from the previous example video. The created map is saved in the directory in which map_saver is running.

[Remote PC]
$ rosrun map_server map_saver -f ~/map



Run Navigation Node

$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml

4th Industrial Revolution

- We are at the beginning of 4th industrial revolution - AI+IoT+5G+Blockchain+Robots+Cloud Computing

1st Industrial Revolution:

- Mechanization, steam power, weaving looms
- Large-scale transportation with steam-powered vessels and railroads
- Replacing human and animal power with machines

2nd Industrial Revolution:

- Electricity, assembly line, mass production
- Internal combustion engines, automobiles
- Radio and television

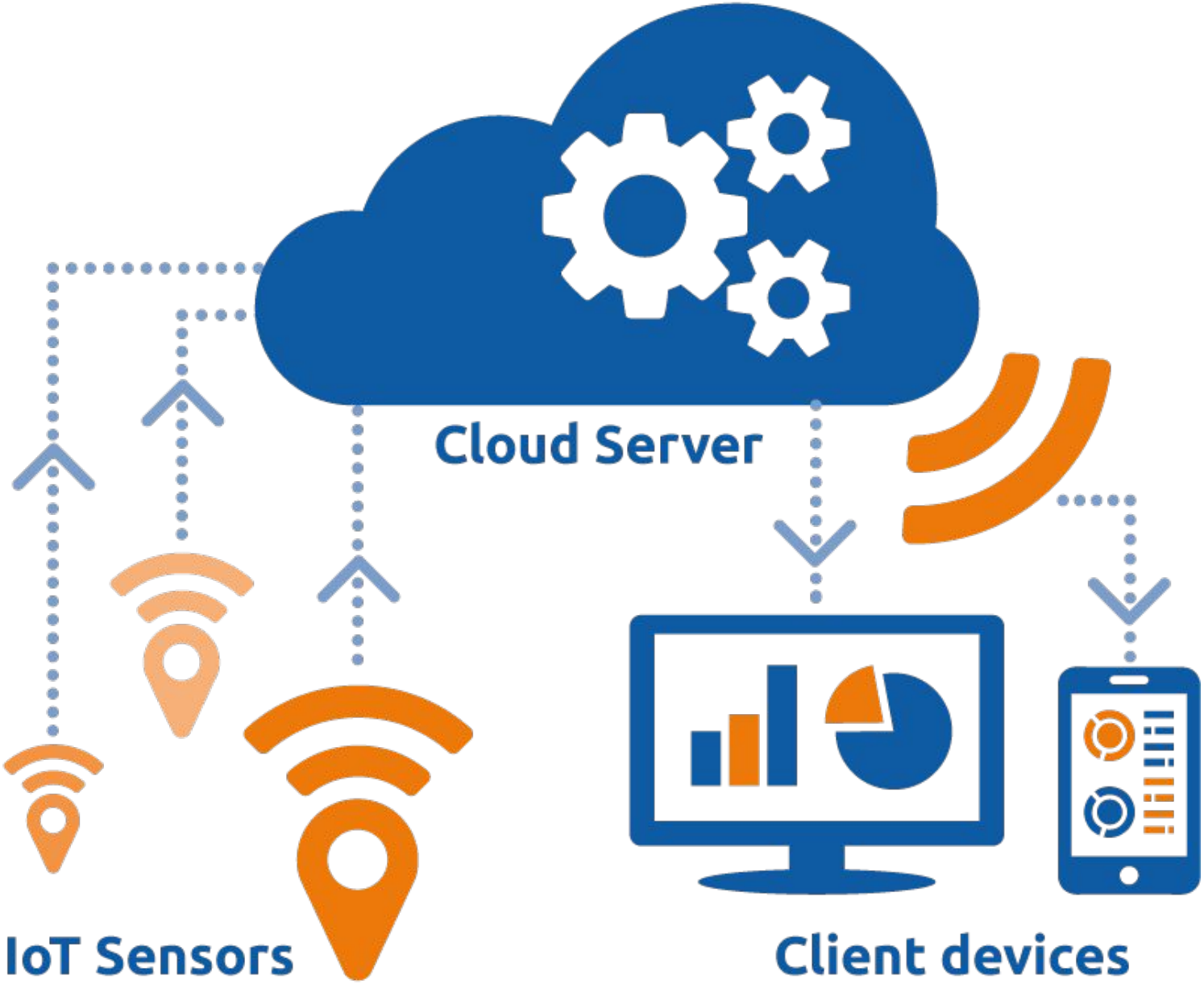
3rd Industrial Revolution:

- Electronics, computers
- Automation
- Information technology

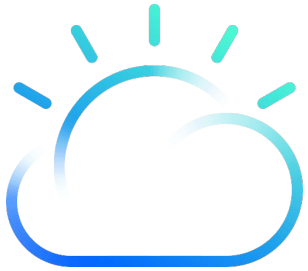
4th Industrial Revolution:

- Cyber-physical systems
- Internet of Things
- Sensor Networks
- Advanced Robotics
- Big Data
- Machine Learning
- Cloud Computing
- Driverless cars
- 3D/4D printing-based manufacturing
- Blockchain transaction architecture

Cloud IoT Computing



Cloud Platforms

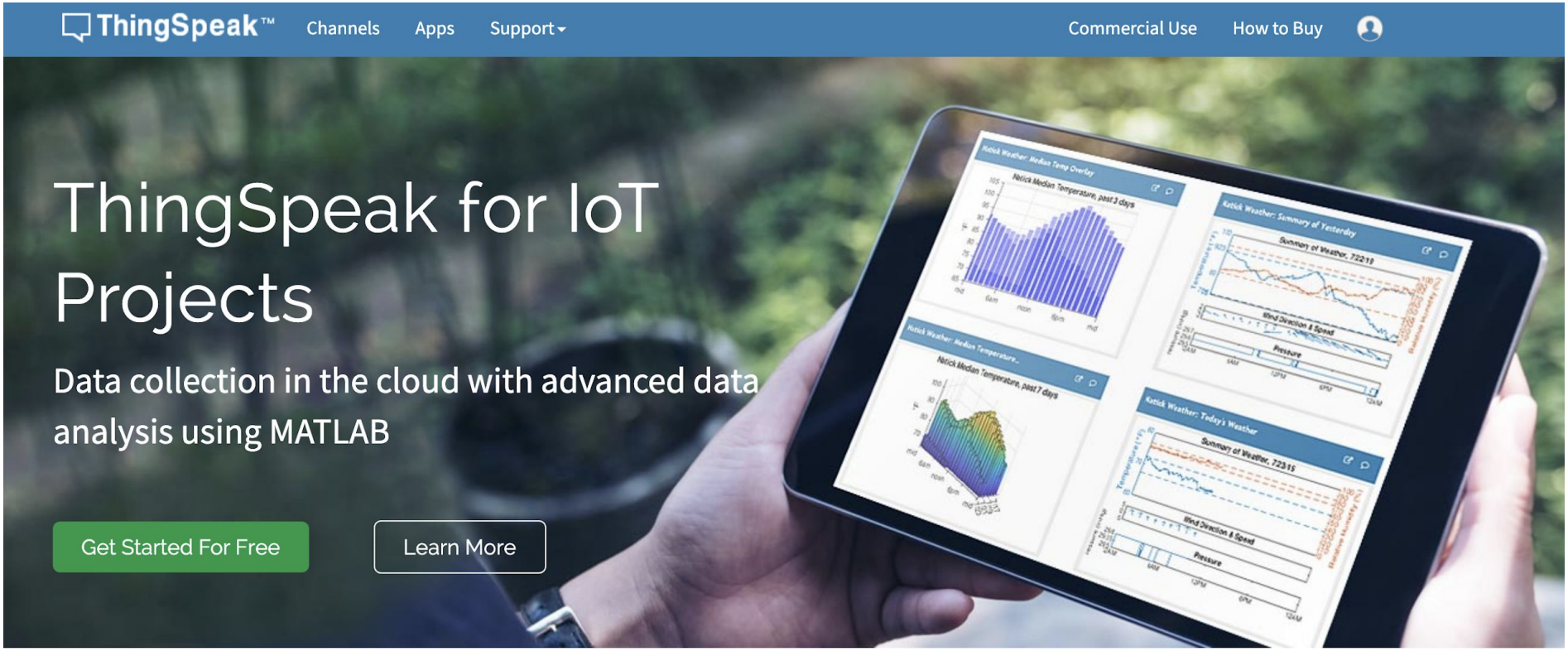


Google Cloud

Alibaba Cloud

thingworx®

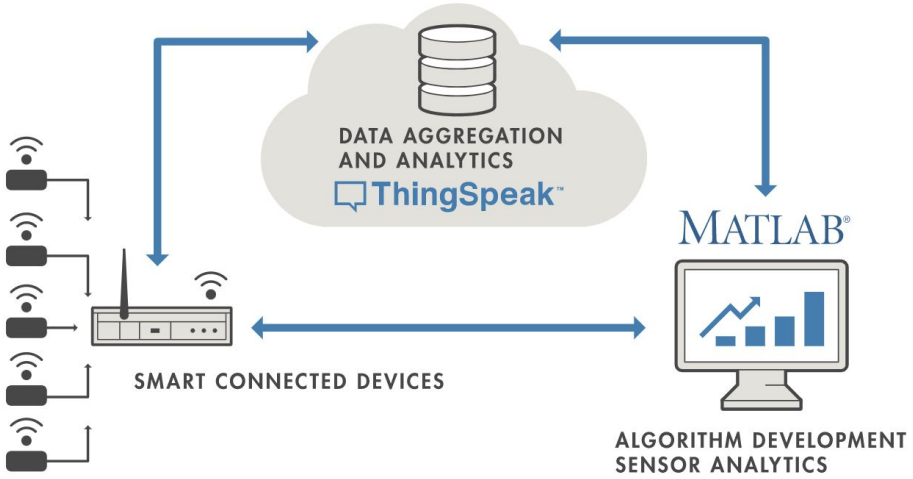
ThingSpeak



- ThingSpeak™ is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams.
- Once you send data to ThingSpeak from your devices, you can create instant visualizations of live data without having to write any code
- With MATLAB® analytics inside ThingSpeak, you can write and execute MATLAB code to perform more advanced preprocessing, visualizations, and analyses

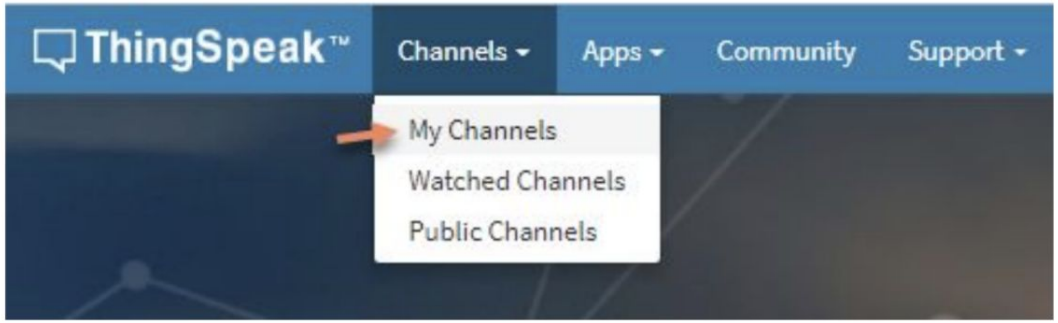
ThingSpeak Features

- Collect data in private channels
- Share data with public channels
- RESTful and MQTT APIs
- MATLAB® analytics and visualizations
- Event scheduling
- Alerts
- App integrations:
  - MATLAB®
  - Arduino®
  - ESP8266 Wifi Module
  - Raspberry Pi™
  - LoRaWAN®



Setup ThingSpeak Channel

- Create account on https://thingspeak.com/
- Click Channels > MyChannels.
- On the Channels page, click New Channel.
- Check the boxes next to Fields 1–7. Enter these channel setting values:
  - Name: Turtlebot Pose
  - Field 1: x Position
  - Field 2: y Position
  - Field 3: z Position
  - Field 4: x Orientation
  - Field 5: y Orientation
  - Field 6: z Orientation
  - Field 7: w Orientation



- Click Save Channel at the bottom of the settings.
- Take note of the Channel ID and the Write API Key; you may want to copy and paste these details into the notepad first

Activity: Install Python MQTT Package

- Install the Paho client library by typing:

$ ~~sudo~~ pip install paho-mqtt

If you have problem to install pip, install pip as follows:
$ curl "https://bootstrap.pypa.io/pip/2.7/get-pip.py" -o "get-pip.py"
$ python get-pip.py

--- Noetic Only ---
$ sudo apt install python3-pip
$ path_planning.py

Modify the Python Code

- Update the Channel ID and API key in the iot.py python code

```
channelID = "XXXXX"

apiKey = "XXXXXXXXXXXXXXXXXXXXXXX"
```

Activity: Run the ROS IoT Simulation

- Launch Gazebo in TurtleBot3 World

$ roslaunch turtlebot3_gazebo turtlebot3_world.launch

- Run node that send required data to your ThingSpeak Channel

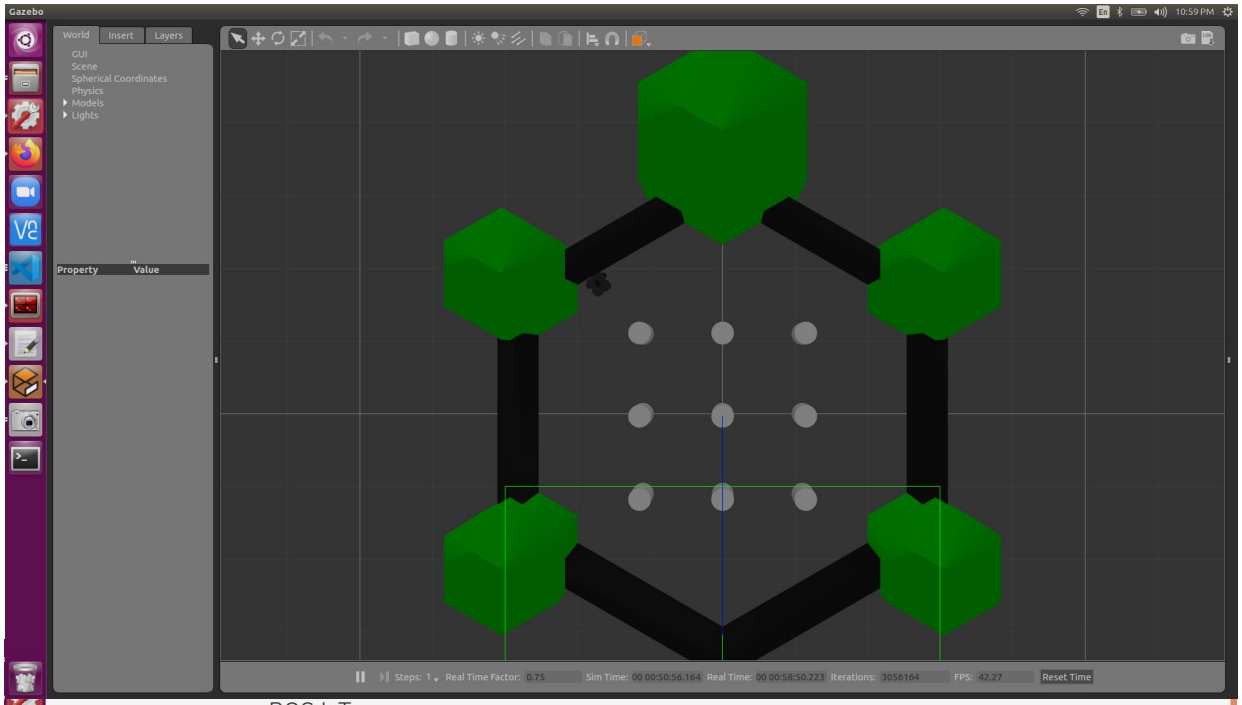
$ rosrun my_robotics iot.py

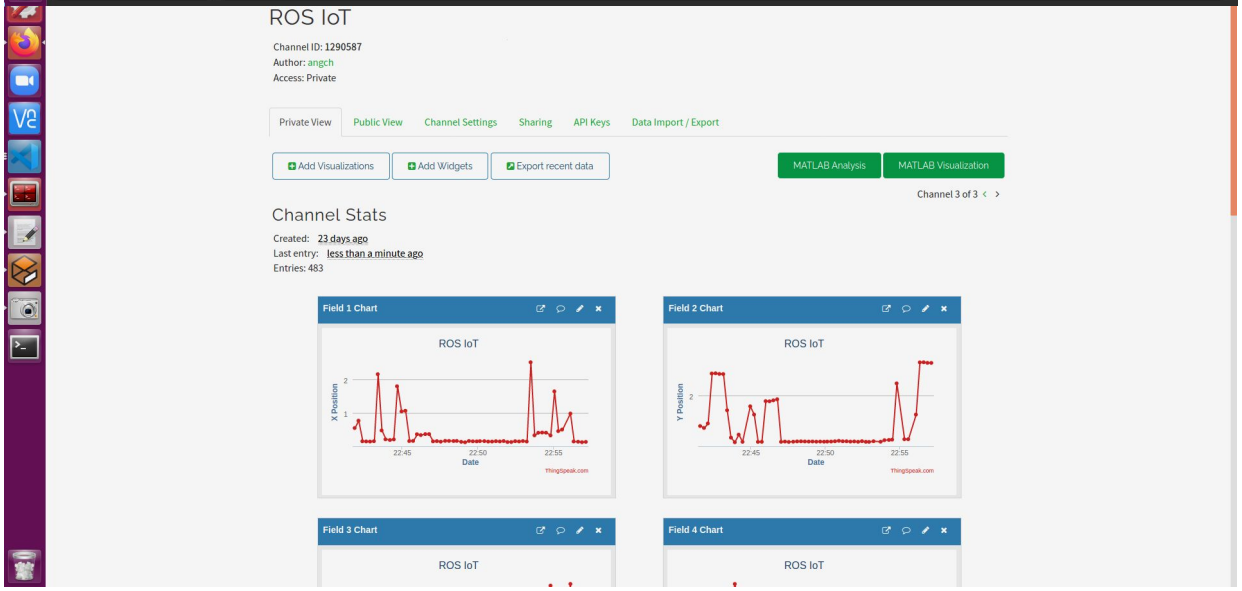
- Check if your ThingSpeak Channel receives the data from your running node

- Move the TB3 around the map using the Teleop key node

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

ROS IoT on ThingSpeak





Vulnerabilities of IoT Devices

- IoT devices are extremely varied in nature such as sensors to collect information from the environment or actuators for moving or controlling a system or environment. These IoT devices are often powered by batteries.

- IoT devices are often resource-constrained. Most IoT devices use a microcontroller rather than a full-fledged microprocessor, and run at a few MHz rather than GHz.

- IoT devices may have physical constraints and accessibility constraints imposed by the operational environment, e.g. pacemakers within the human body. A

- Cost is a critical constraint for IoT devices.

IoT Security vs IT Security

While IoT security faces many similar cyber threats as conventional IT security, there are cyber security problems that pertaining specifically to IoT. For example:

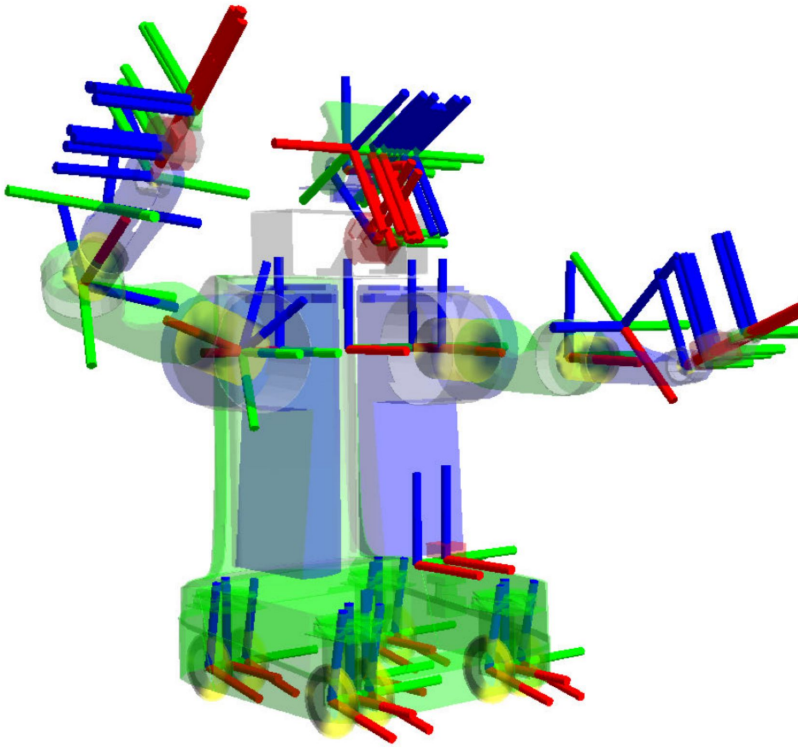
- IoT connections cannot generally rely on TLS/SSL for encrypted and authenticated communications because many IoT devices do not have the resources to handle session establishment, communication overheads, or encryption.

- IoT devices may run without supervision and for extended periods of time, possibly in hostile environments – making them particularly susceptible to hacking. Many might have zero or limited user interfacing; thus, patching and updating may not be convenient and malfunctioning or rogue devices may not be immediately detectable.

- The fact that IoT is closely integrated with the physical world can increase the impact of cyber-attacks. While IT cyber-attacks have resulted in data leakage and financial losses, IoT cyber-attacks have the potential to cause direct physical harm

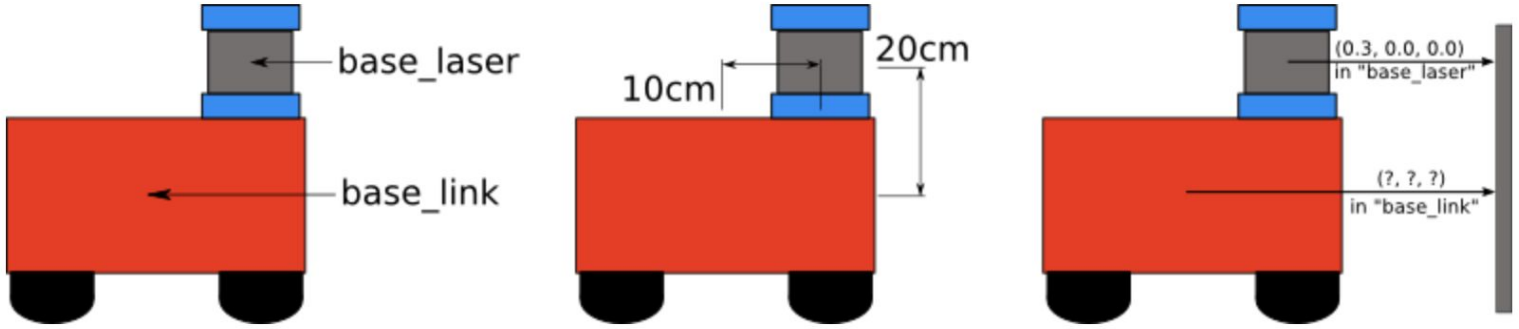
Transformation System (TF)

- A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc.
- tf (transformation system) keeps track of all these frames over time, and allows you to ask questions like:
  - Where was the head frame relative to the world frame, 5 seconds ago?
  - What is the pose of the object in my gripper relative to my base?
  - What is the current pose of the base frame in the map frame
- tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

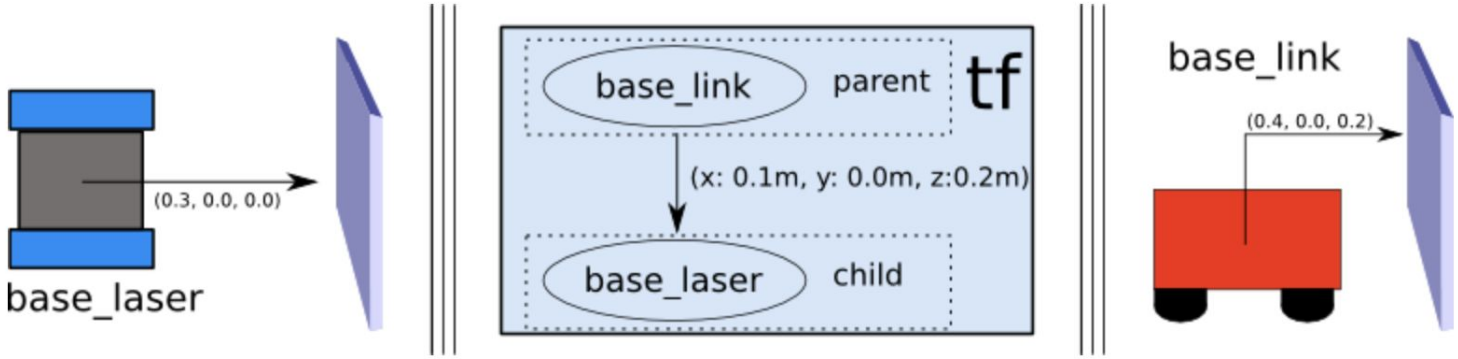


A Simple Robot Example Using TF

- Consider the example of a simple robot (base_link) that has a mobile base (base_laser) with a single laser mounted on top of it.

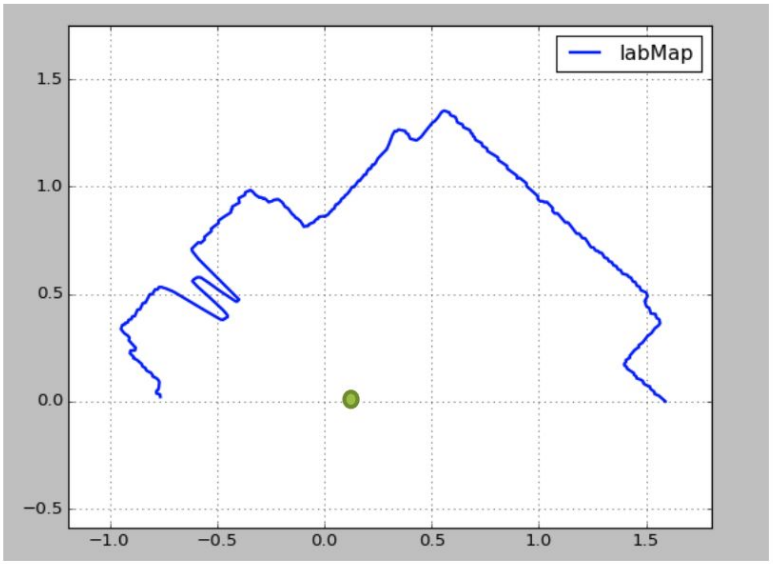


- TF defines and stores the relationship between the "base_link" and "base_laser" frames, and add them to a transform tree.
- To create a transform tree, we create two nodes, one for the "base_link" coordinate frame and one for the "base_laser" coordinate frame.
- To create the edge between them, we need to decide which node will be the parent and which will be the child

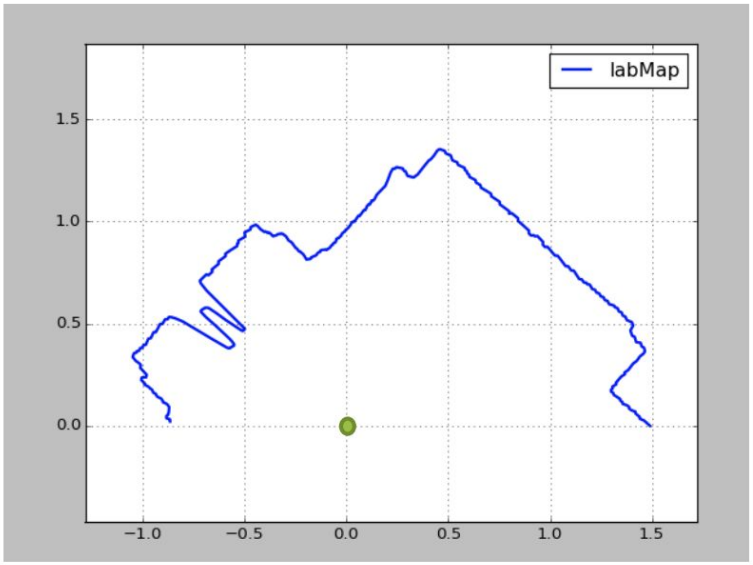


A Simple Robot Example Using TF

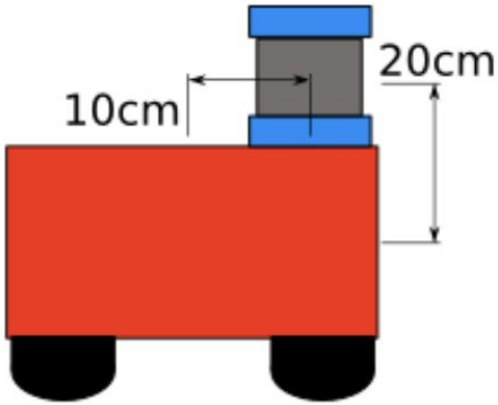
- TF allows one to view the LIDAR maps based on the base_lin or base_laser coordinate frames

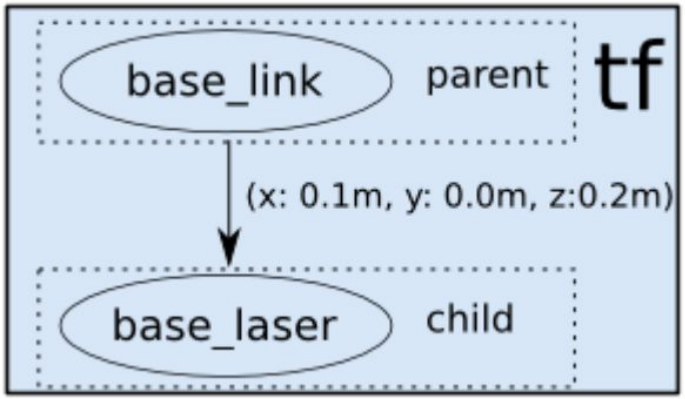


LIDAR reading in base_link coordinate frame



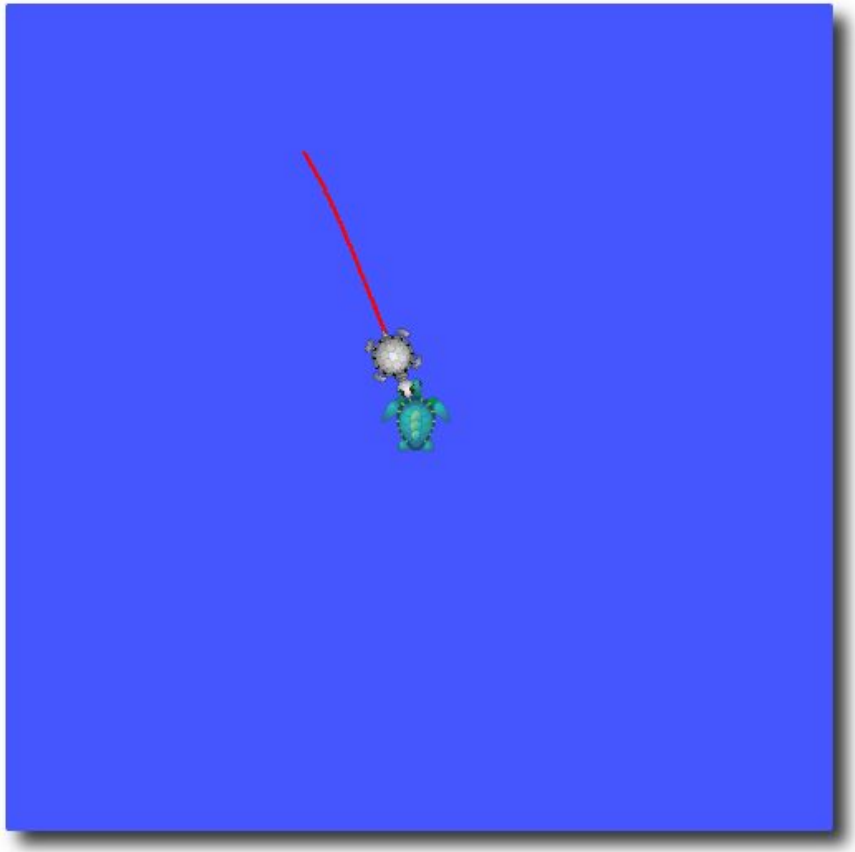
LIDAR reading in base_laser coordinate frame

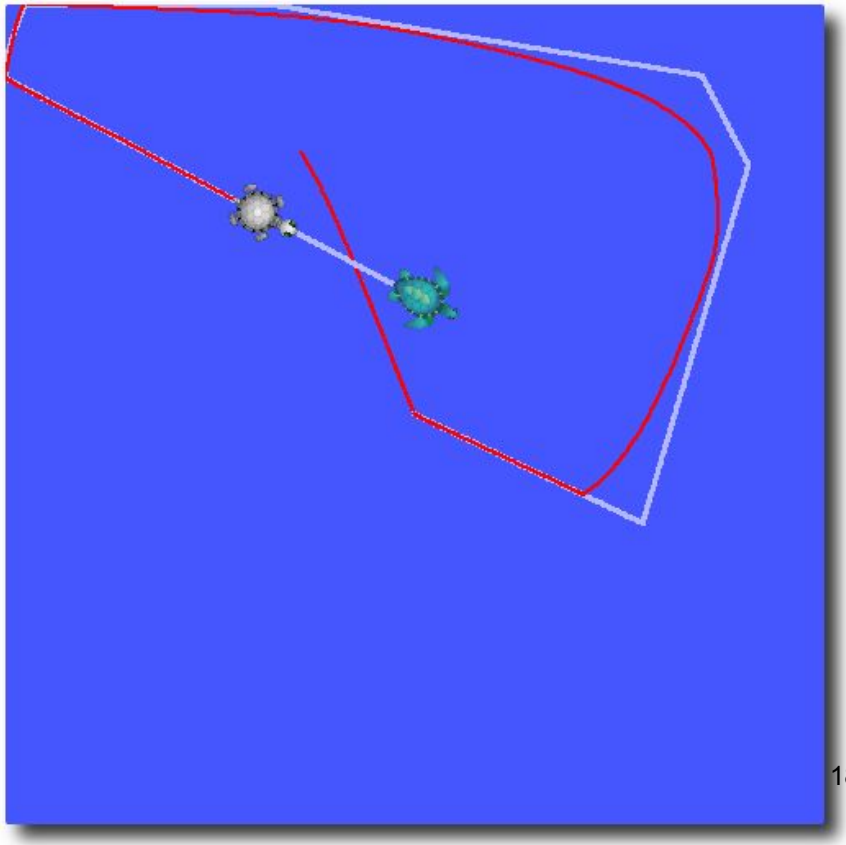




Activity: TF Demo - 1

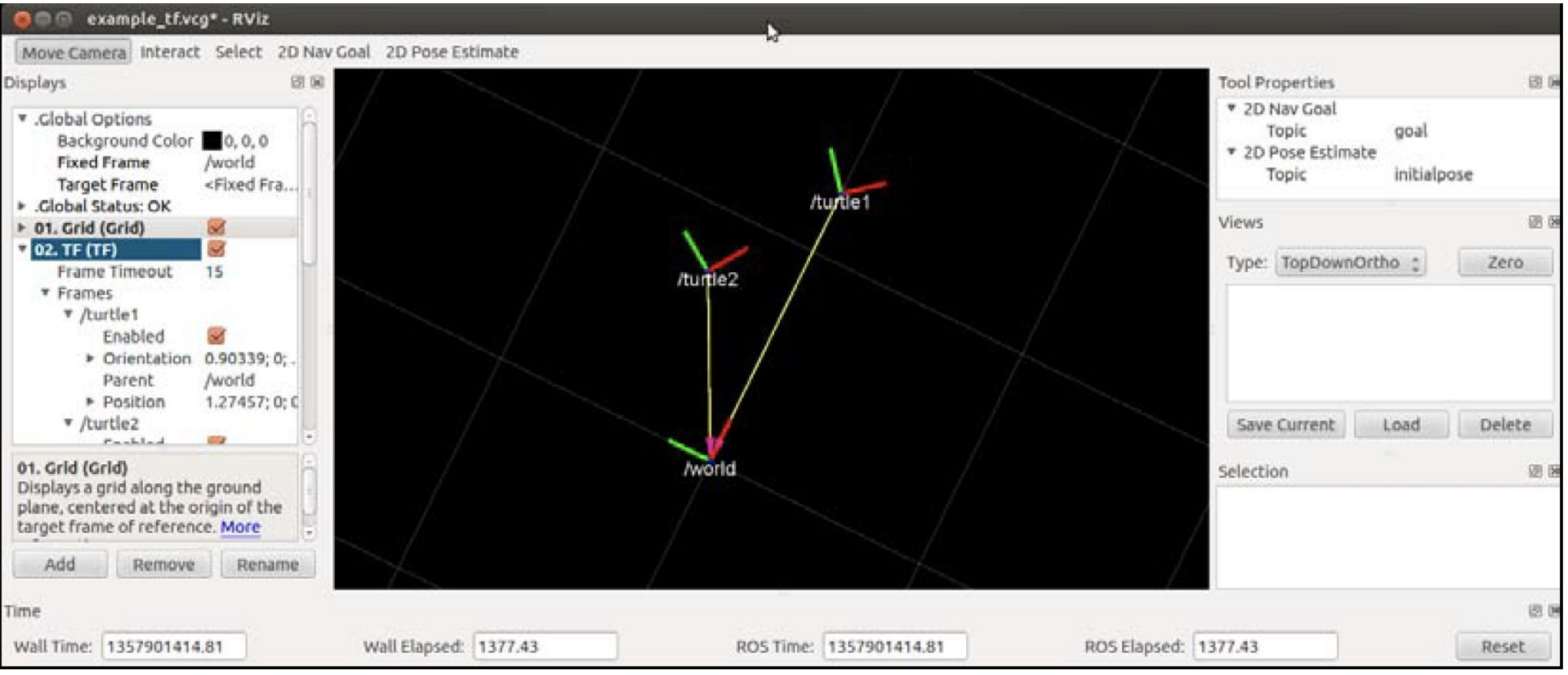
- sudo apt-get install ros-kinetic-turtle-tf2 ros-kinetic-tf2-tools ros-kinetic-tf
- (terminal 1) roslaunch turtle_tf2 turtle_tf2_demo.launch
- (terminal 2) ~~rosrun turtlesim turtle_teleop_key~~
- (terminal 3) rviz
- Turn on TF
- Observe one turtle tracks the other turtle





Activity: TF Demo - 2

- Set the fixed frame to /world
- Add the TF plugin to the left area. You will see that we have the /turtle1 and /turtle2 frames, both as children of the /world frame.
- Set the view of the world to TopDownOrtho



What is URDF?

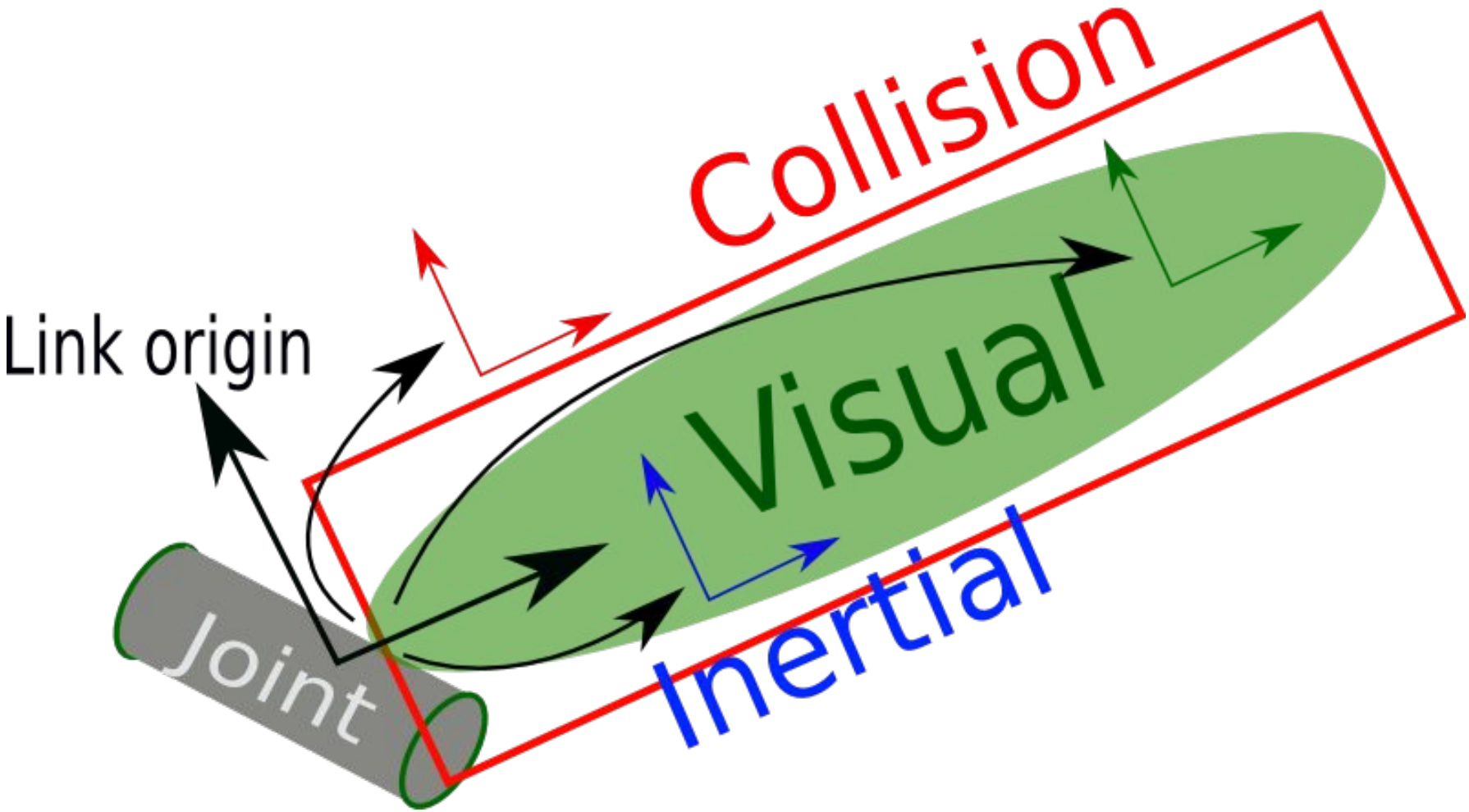
- The Unified Robot Description Format (URDF) is a XML specification to describe a robot.
- Note that the URDF specification cannot describe all robots. The main limitation at this point is that only tree structures can be represented, ruling out all parallel robots. Also, the specification assumes the robot consists of rigid links connected by joints; flexible elements are not supported.
- The URDF specification covers:
  - Kinematic and dynamic description of the robot
  - Visual representation of the robot
  - Collision model of the robot

<robot> Element

- The root element in a robot description file must be a robot, with all other elements must be encapsulated within.

```
<robot name="pr2">
  <!-- pr2 robot links and joints and more -->
</robot>
```

<link> Element

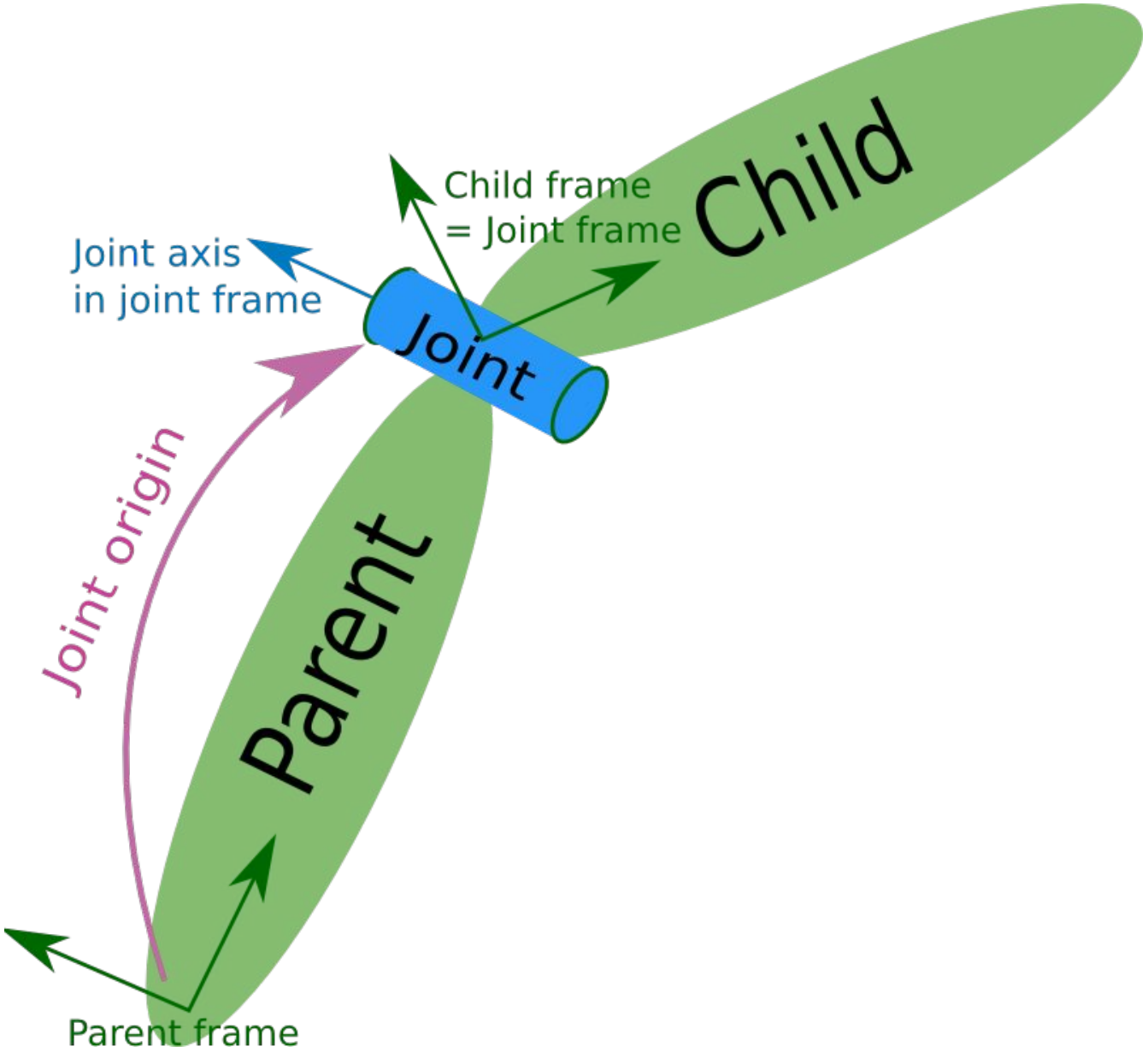


<link> Element

- The link element define the geometry (cylinder, box, , sphere, or mesh), the material (color, texture) and the origin.
- It also describes a rigid body with an inertia, visual features, and collision properties.

```
<link name="base_link">
    <visual>
        <geometry>
            <box size="0.2 .3 .1"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0.05"/>
        <material name="white">
            <color rgba="1 1 1 1"/>
        </material>
    </visual>
</link>
```

<joint> Element



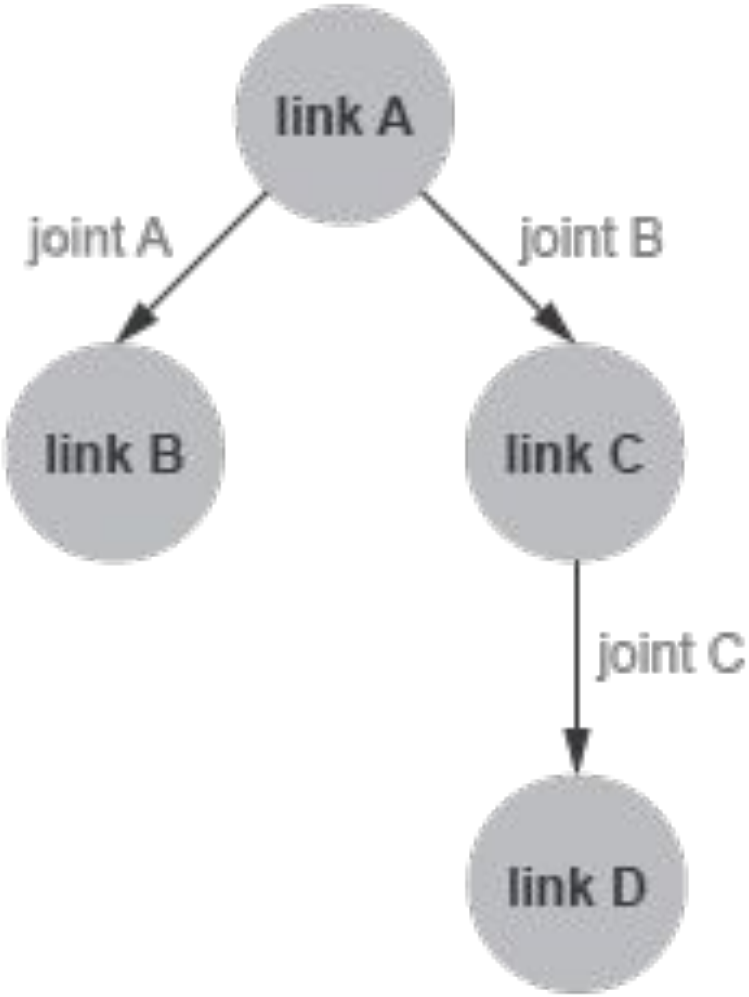
<joint> Element

- The joint element define the type of joint (fixed, revolute, continuous, floating, or planar).
- It also describes the kinematics and dynamics of the joint and also specifies the safety limits of the joint.

```
<joint name="base_to_wheel1" type="fixed">
    <parent link="base_link"/>
    <child link="wheel_1"/>
    <origin xyz="0 0 0"/>
</joint>
```

URDF Structure

```
<robot>
    <joint name = "joint A>
        <parent link = "link A" />
        <child link = "link B" />
    </joint>
    <joint name = "joint B>
        <parent link = "link A" />
        <child link = "link C" />
    </joint>
    <joint name = "joint C>
        <parent link = "link C" />
        <child link = "link D" />
    </joint>
</robot>
```



A Simple Robot URDF

```
<?xml version="1.0"?>
 <robot name="multipleshapes">
     <link name="base_link">
       <visual>
         <geometry>
           <cylinder length="0.6" radius="0.2"/>
         </geometry>
       </visual>
     </link>
    <link name="right_leg">
      <visual>
        <geometry>
          <box size="0.6 0.1 0.2"/>
        </geometry>
      </visual>
    </link>
    <joint name="base_to_right_leg" type="fixed">
      <parent link="base_link"/>
      <child link="right_leg"/>
    </joint>
 </robot>
```

Adding Physics to a URDF Model

- To add physics to a URDF model, you need to add collision and inertia elements. These are required for the gazebo simulation.
- The collision element is a direct subelement of the link object, at the same level as the visual tag
- The inertia element consists of mass (in kg) and 3x3 rotational inertia matrix defined by:

  ixx ixy ixz
  ixy iyy iyz
  ixz iyz izz

Example of Collision and Inertia

```
<link name="base_link">
      <visual>
        <geometry>
          <cylinder length="0.6" radius="0.2"/>
        </geometry>
        <material name="blue">
            <color rgba="0 0 .8 1"/>
        </material>
       </visual>
      <collision>
        <geometry>
          <cylinder length="0.6" radius="0.2"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="10"/>
        <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4" iyz="0.0" izz="0.2"/>
      </inertial>
</link>
```

Display on RViz

- To view the URDF in RViz, we can create a launch file with the following code
- Change the red text to your package name

```
<launch>
  <arg name="gui" default="true" />
  <param name="robot_description" command="$(find xacro)/xacro $(arg
model)" />
  <node if="$(arg gui)" name="joint_state_publisher"
pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />
  <node unless="$(arg gui)" name="joint_state_publisher"
pkg="joint_state_publisher" type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
trobotics_arv)/rviz/urdf.rviz" required="true" />
</launch>
```

Install Joint State Publisher Package

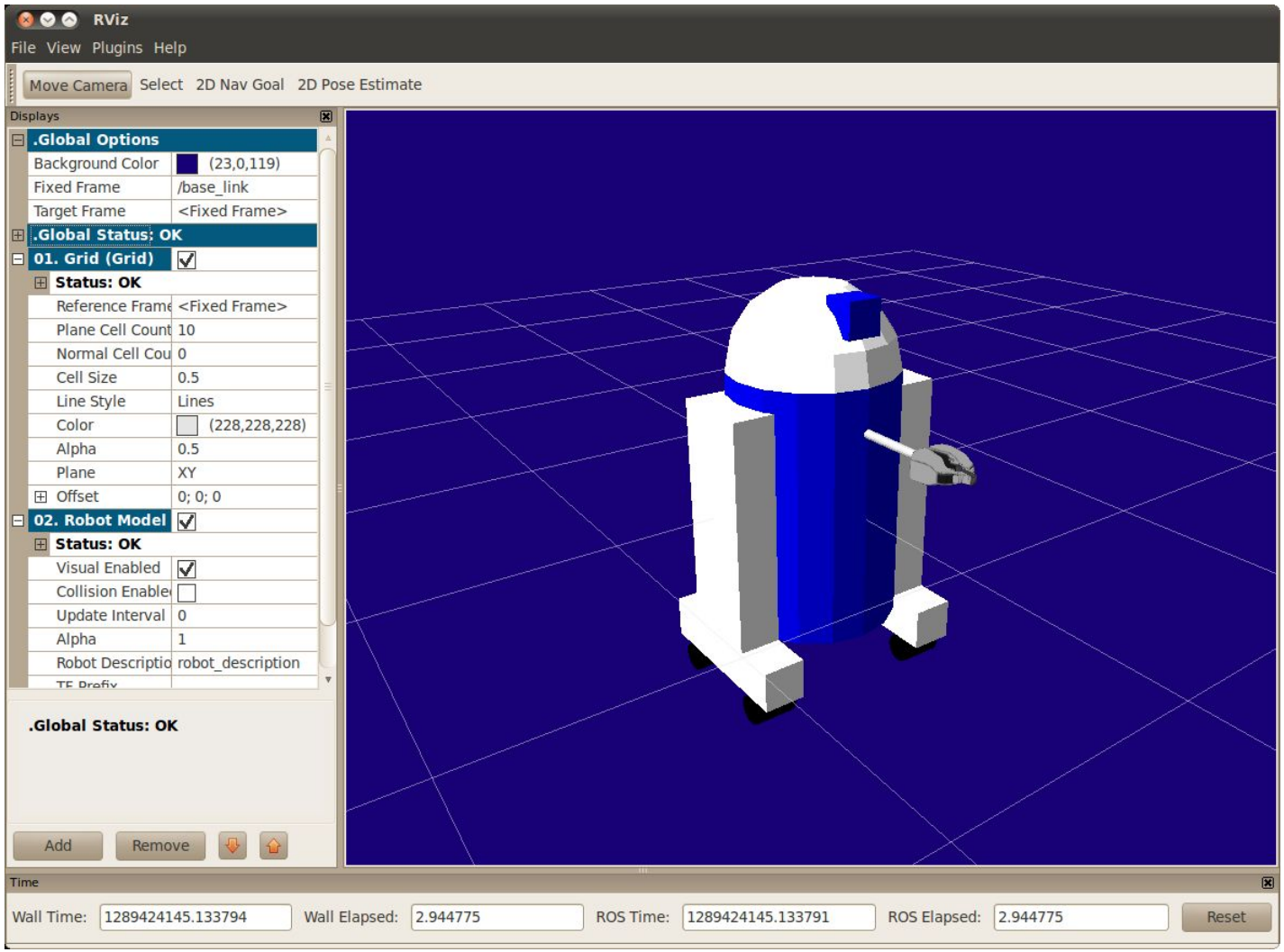
- If you have problem to view the RViz robot correct, install the following packages

```
sudo apt install ros-kinetic-joint-state-publisher-gui
```

Activity: Visualize URDF in RViz

- Add the display.launch to the launch folder in your package
- Launch the visual.urdf as follows:

$ roslaunch <your package name> display.launch model:='visual.urdf'



Xacro

- Xacro helps to reduce the overall size of the URDF and make it easier to read and maintain.
- It also allows us to create modules and reutilize them to create repeated structures such as several arms or legs.
- To convert a xacro to urdf file, you can run the following command line:

```
xacro --inorder model.xacro > model.urdf
```

Xacro - Constant

- Xacro can define constants using xacro:property and make it easier to maintain.
- The value of the contents of the ${} construct are then used to replace the ${}. This means you can combine it with other text in the attribute.

```
<link name="base_link">
  <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue"/>
  </visual>
  <collision>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
  </collision>
</link>
```

```
<xacro:property name="width" value="0.2" />
<xacro:property name="len" value="0.6" />
<link name="base_link">
  <visual>
      <geometry>
        <cylinder radius="${width}" length="${len}"/>
      </geometry>
      <material name="blue"/>
  </visual>
  <collision>
      <geometry>
        cylinder radius="${width}" length="${len}"/>
      </geometry>
  </collision>
</link>
```

Xacro - Math

- You can build up arbitrarily complex expressions in the ${} construct using the four basic operations (+,-,*,/), the unary minus, and parenthesis.
- Examples:

```
<cylinder radius="${wheeldiam/2}" length="0.1"/>
<origin xyz="${reflect*(width+.02)} 0 0.25" />
```

Xacro - Macro

- Xacro can generate a macro using xacro:micro

```
<xacro:macro name="default_origin">
    <origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:macro>
<xacro:default_origin />
```

- This code will generate the following.

```
<origin rpy="0 0 0" xyz="0 0 0"/>
```

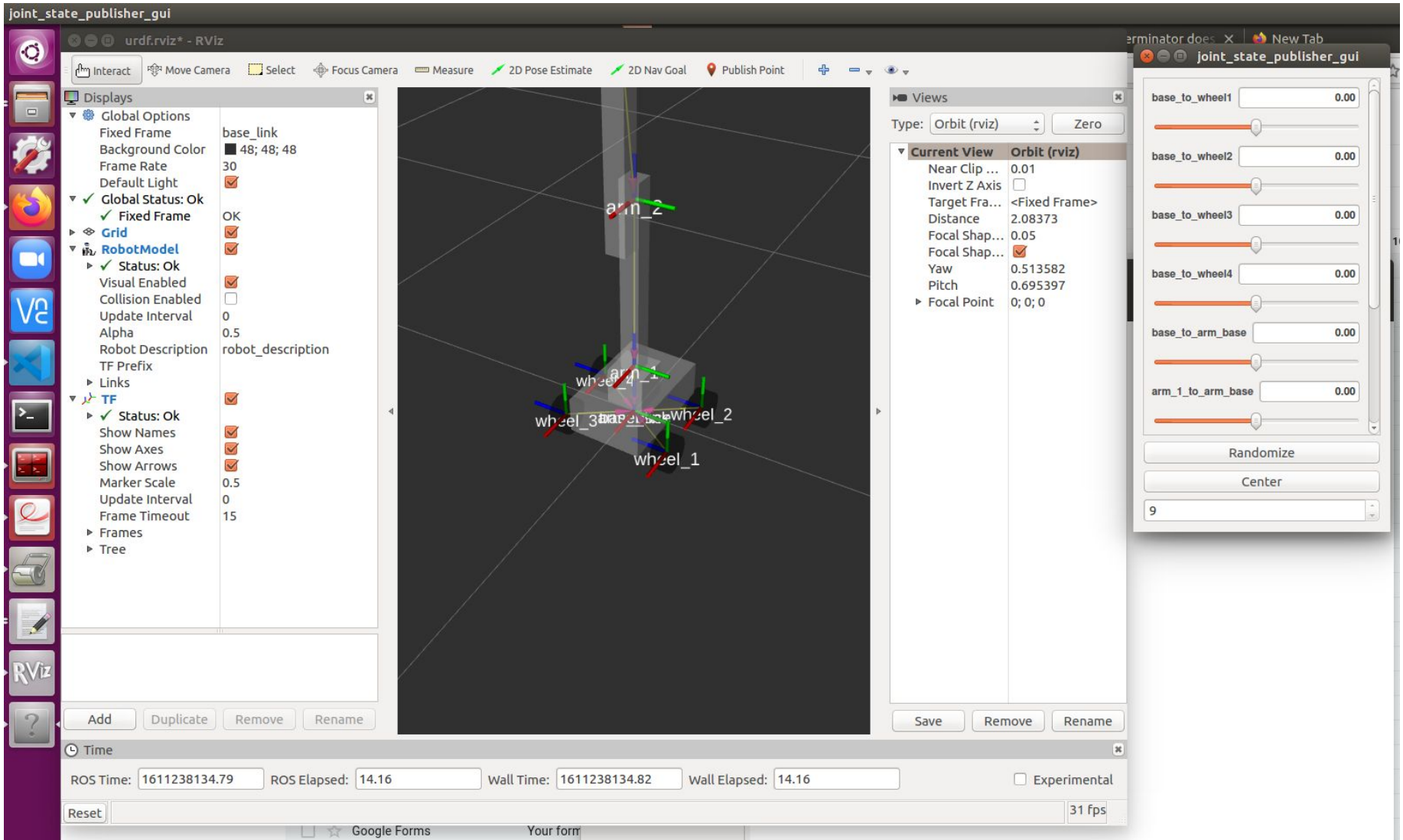
Xacro - Parameterized Macro

- You can also parameterize macros so that they don't generate the same exact text every time

```
<xacro:macro name="default_inertial" params="mass">
    <inertial>
        <mass value="${mass}" />
        <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
        izz="1.0" />
    </inertial>
 </xacro:macro>
 <xacro:default_inertial mass="10"/>
```

Activity: Xacro

$ xacro --inorder robot1.xacro > robot1.urdf
$ roslaunch <your package name> display.launch model:='robot1.urdf'
$ move the sliders to see how the robot arm move



Summary Q&A

Final Assessment

Written Assessment (Q&A)

This is the Written Assessment (Q&A)

Duration: 60 mins

1. The assessor will pass the questions in hardcopy to you. There are 10 questions. You need to answer all the questions.
2. This is an open book exam that must be completed individually.

.

Written Assessment (PP)

This is a Practical Performance Assessment

Duration: 90 mins

1. The assessor will pass a practical problem in hardcopy to you.
2. You need to solve the problem using ROS.
3. This is an open book exam that must be completed individually.

Feedback

https://goo.gl/R2eumq

TRAQOM Survey

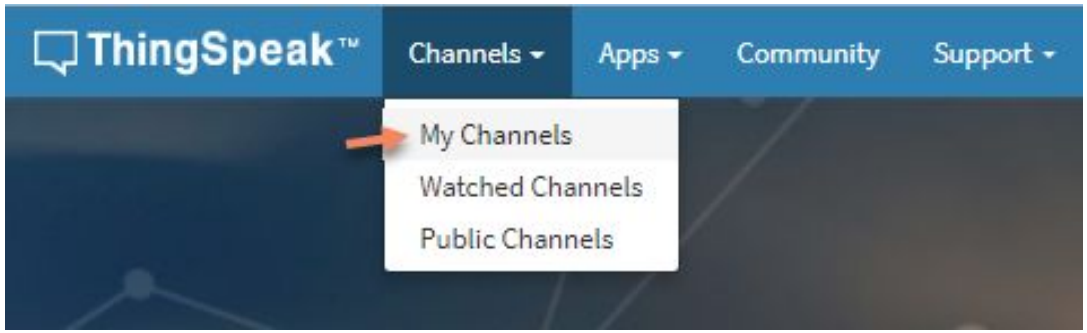
- You will receive a TRAQOM link after the class.
- Please submit TRAQOM feedback and survey after the class.

Thank You!

Man Guo Chang
gc.man.sg@gmail.com

Setup ThingSpeak Channel

- Create account on https://thingspeak.com/
- Click Channels > MyChannels.
- On the Channels page, click New Channel.
- Check the boxes next to Fields 1–7. Enter these channel setting values:
  - Name: Turtlebot Pose
  - Field 1: x Position
  - Field 2: y Position
  - Field 3: z Position
  - Field 4: x Orientation
  - Field 5: y Orientation
  - Field 6: z Orientation
  - Field 7: w Orientation



- Click Save Channel at the bottom of the settings.
- Take note of the Channel ID and the Write API Key; you may want to copy and paste these details into the notepad first