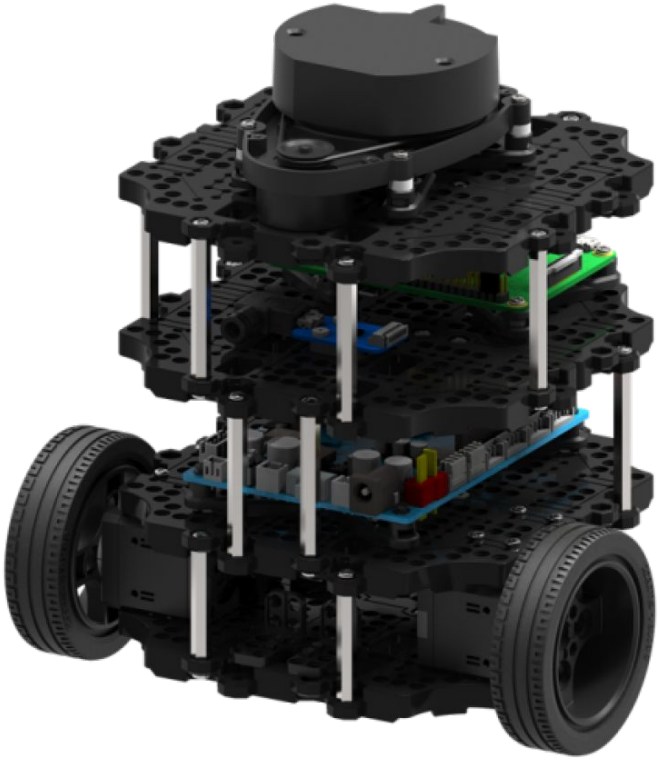
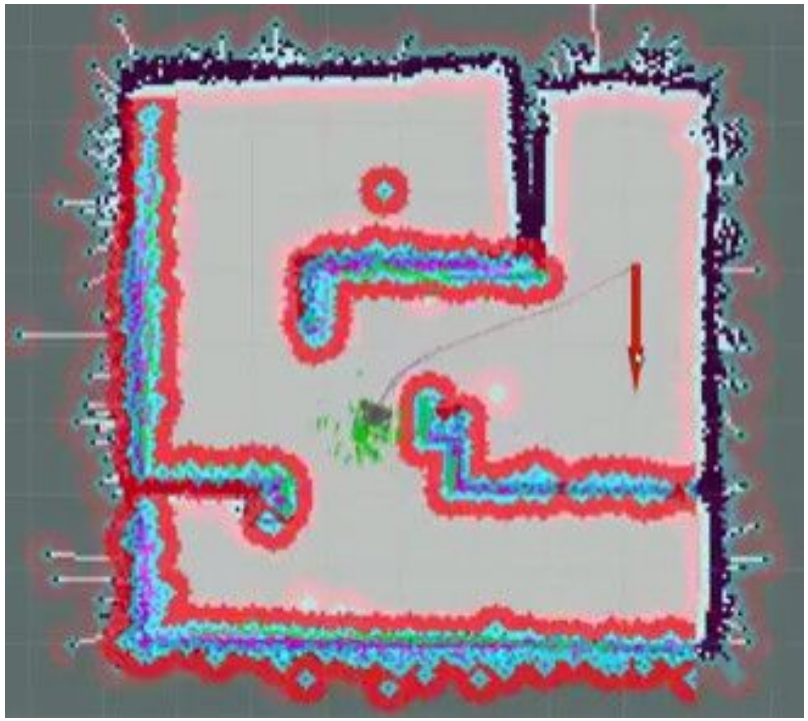
Full ROS Navigation Course v15





Trainer: Man Guo Chang

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, Robot Operating System, 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 ROS?
- Why do you want to learn ROS Navigation?
- 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
- Participant actively. Feel free to ask questions on the chat whenever.
- Facilitators can use breakout rooms for private sessions.

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 Python or C++ programming
- Basic Linux
- Basic ROS

Agenda

Topic 1 LiDAR (Light Detection and Ranging)

- Introduction to LiDAR
- Overview of Turtlebot3 Burger
- Turtlebot3 Burger Simulation on Gazebo and RViz
- Obstacle Avoidance Simulation

Topic 2 SLAM (Simultaneous Localization And Mapping)

- Introduction to SLAM
- Virtual Robot SLAM Simulation on Gazebo
- Introduction to Path Finding
- Virtual Robot Navigation on Gazebo

Topic 3 Physical Robot Navigation

- Bring Up Physical Turtlebot3 Burger with Remote Control
- Teleoperation of Turtlebot3 Burger
- Obstacle Avoidance on Turtlebot3 Burger
- Perform SLAM and Navigation on Turtlebot3 Burger

Agenda

Topic 4 ROS Navigation with Python

- Move Turtlebot with Python
- Collect Laser Scan Data
- Obstacle Avoidance with Python
- Initial Pose Estimation
- Autonomous SLAM

Topic 5 ROS Navigation IoT

- Introduction to IoT
- Setup ThingSpeak
- IoT with ROS

Topic 6 ROS Navigation with Vision

- Line Following
- Gesture Controlled Navigation

Download Course Material

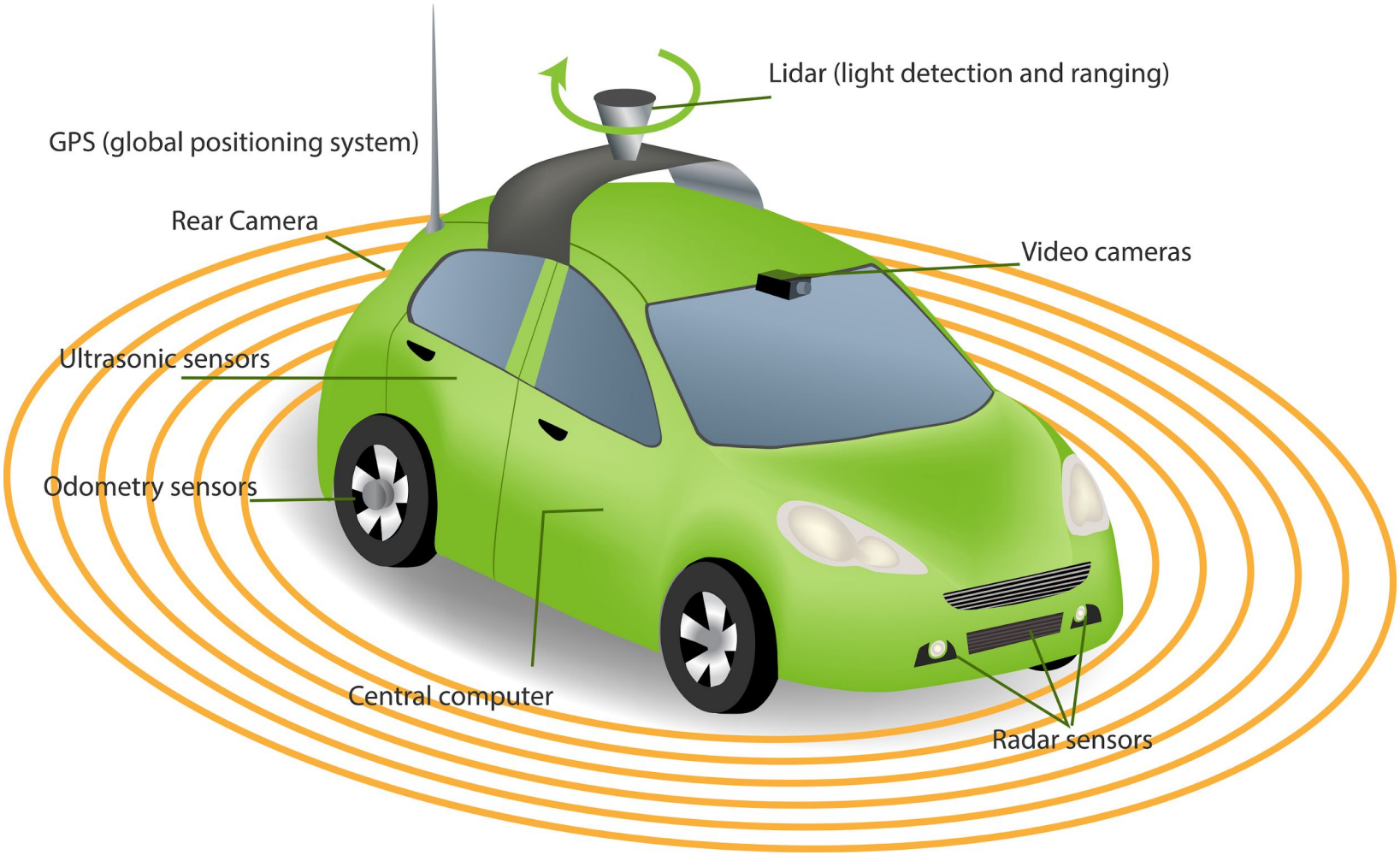
- Your can download the course material from Google Classroom https://classroom.google.com
- Enter the class code below to join the class on the top right.
- Goto Classwork >> Course Material
- If you cannot access the google classroom, please inform the trainer or staff.
-

q7qdfzo

Topic 1
LiDAR (Light Detection and Ranging)

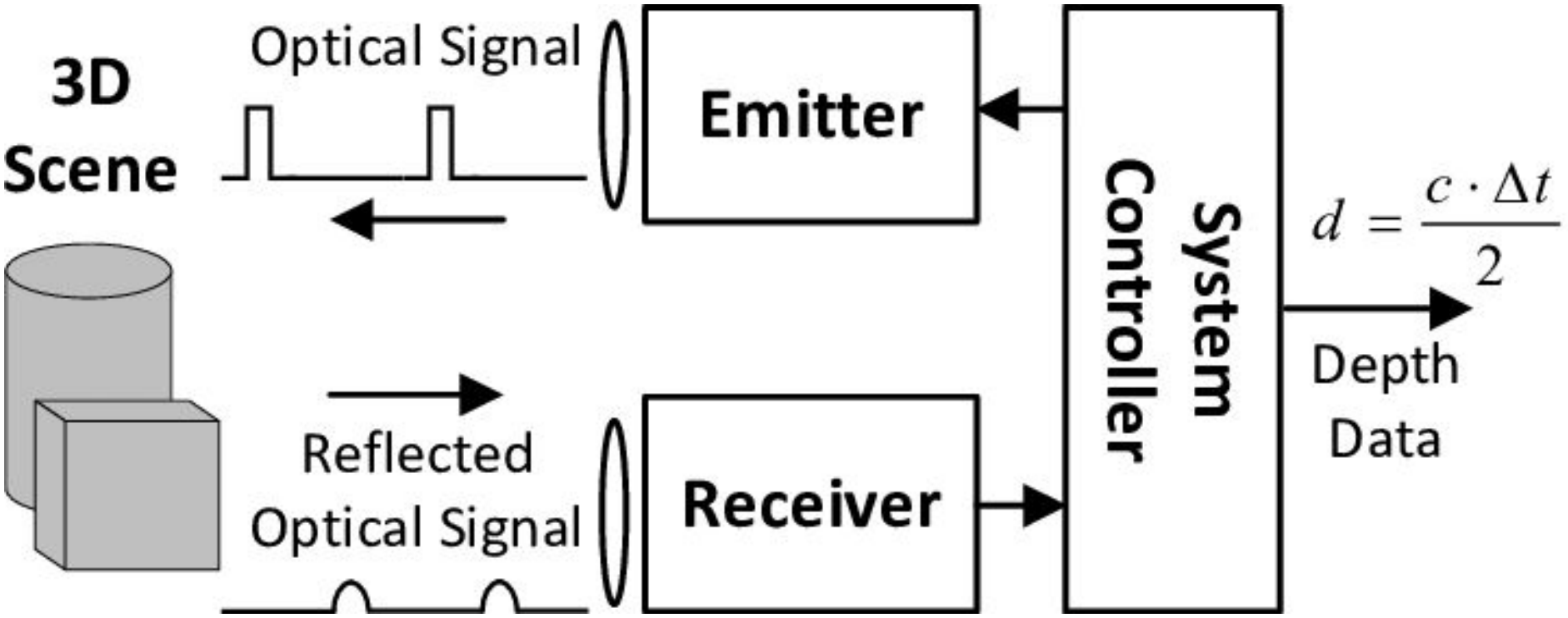
What is LiDAR

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



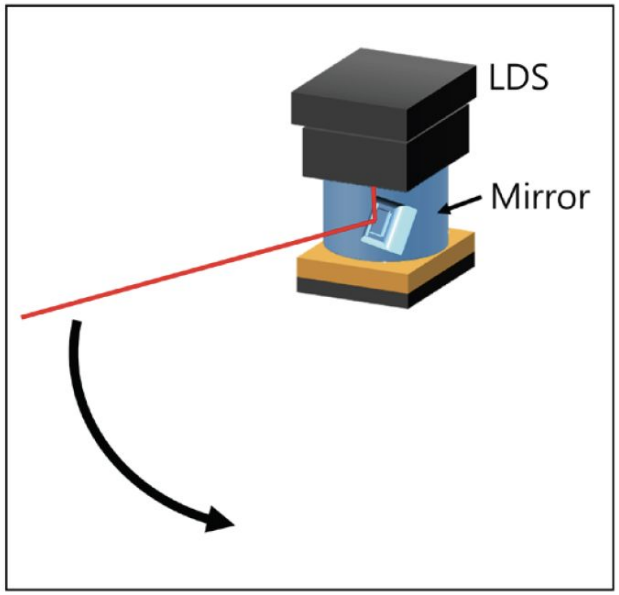
Principle of LIDAR -1

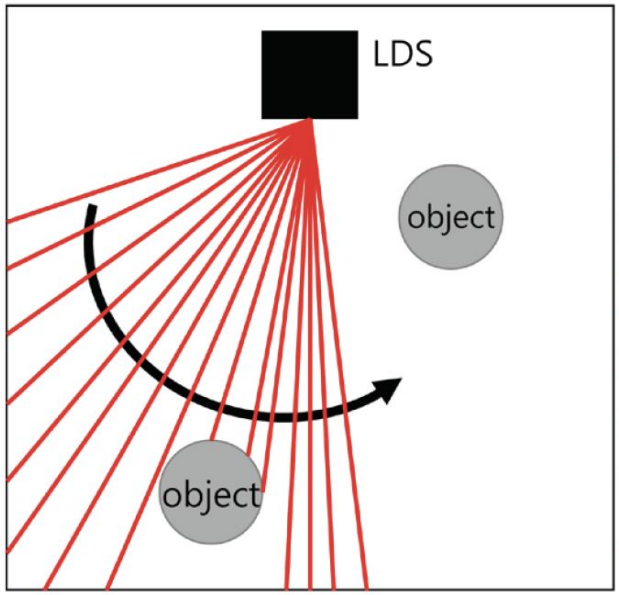
LiDAR (Light Detection And Ranging) sensors work on the same principle as RADAR, firing a wavelength at an object and timing the delay in its return to the source to measure the distance between the two points.

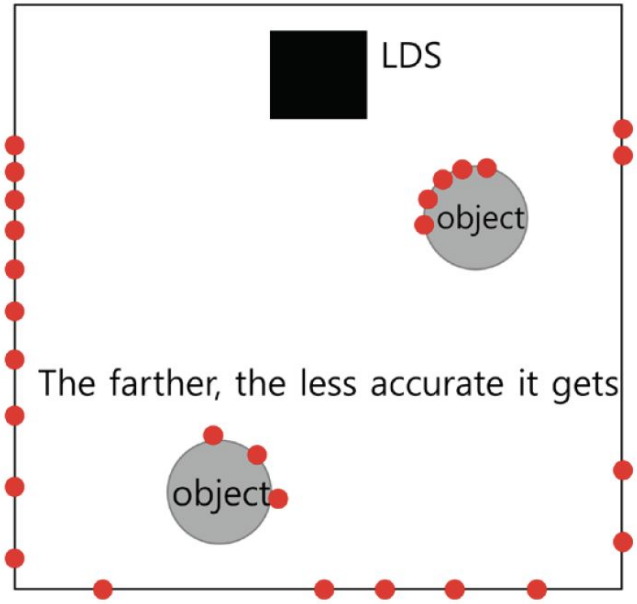


Principle of LIDAR - 2

- 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.

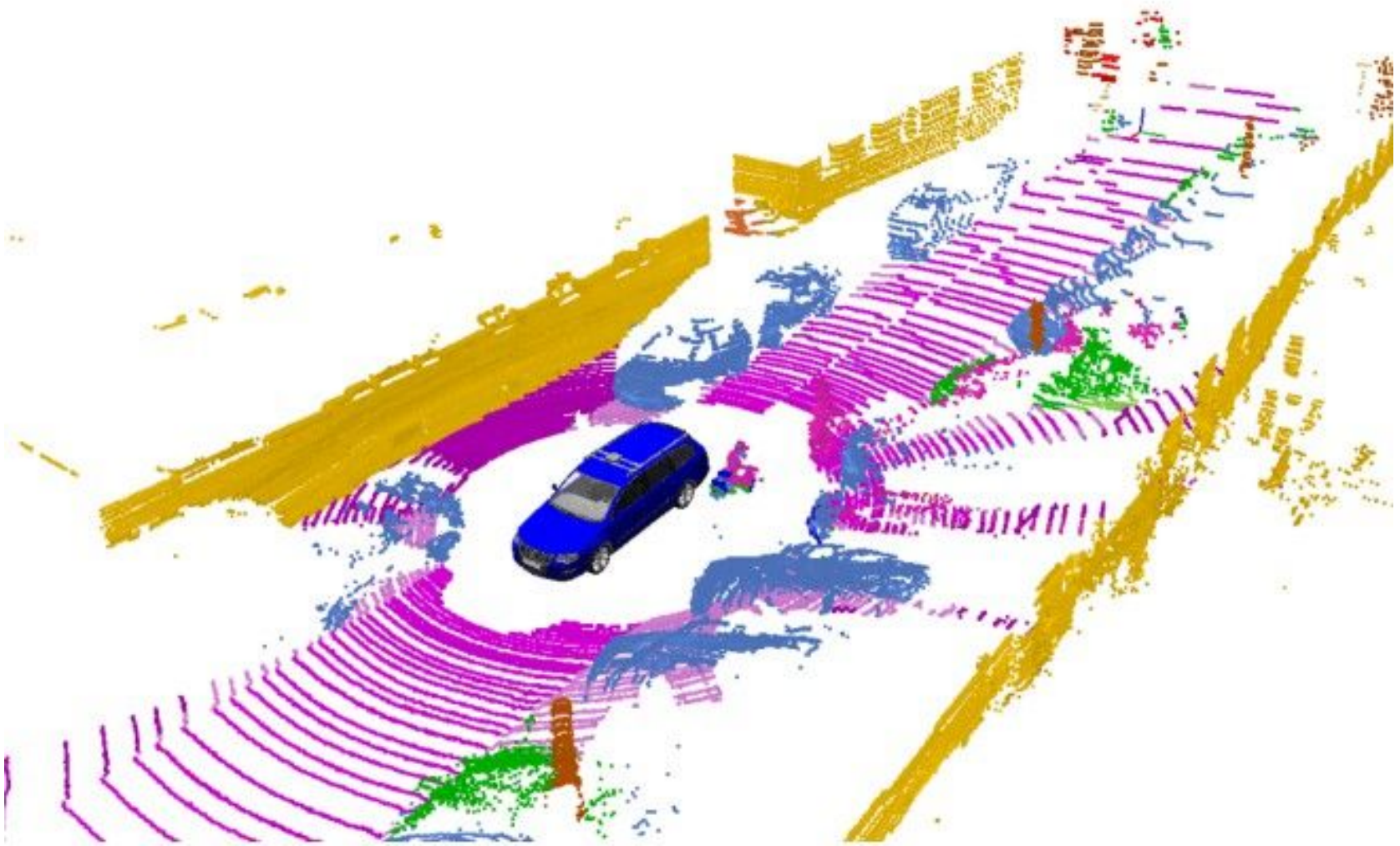






Point Cloud

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

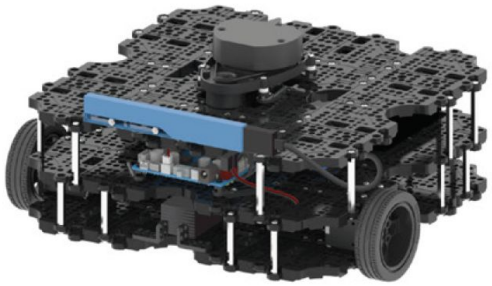
- The RADAR system works in much the same way as the LIDAR, with the only difference being that it uses radio waves instead of laser. In the RADAR instrument, the antenna doubles up as a radar receiver as well as a transmitter. However, radio waves have less absorption compared to the light waves when 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

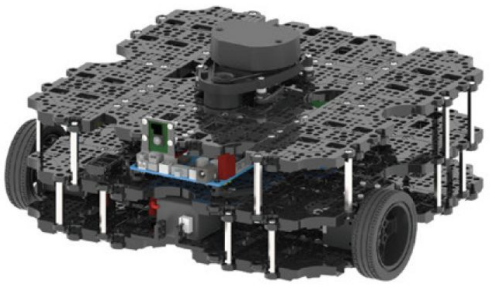
- TurtleBot is a ROS standard platform robot.
- There are three versions of the TurtleBot series.
- TurtleBot1 was developed in 2010 by Tully (Platform Manager at Open Robotics) and Melonee (CEO of Fetch Robotics) from Willow Garage
- In 2012, TurtleBot2 was developed by Yujin Robot based on the research 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



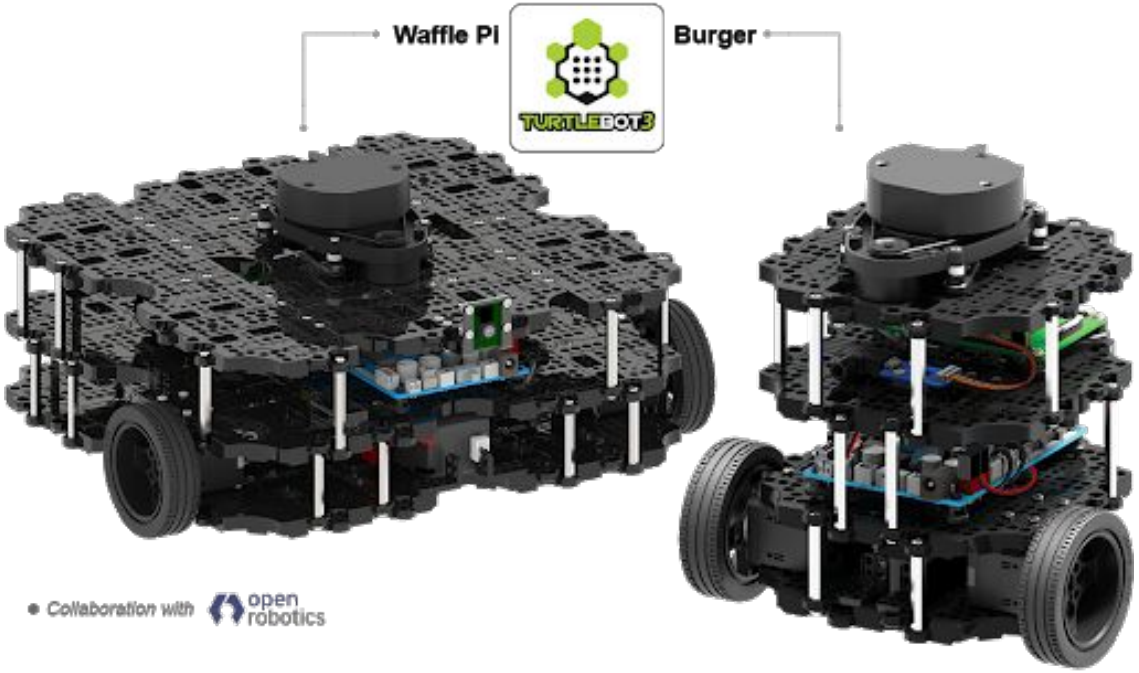
TurtleBot3



TurtleBot3

- The two basic types of model structures of TurtleBot3 are Burger and WafflePi.
- 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.

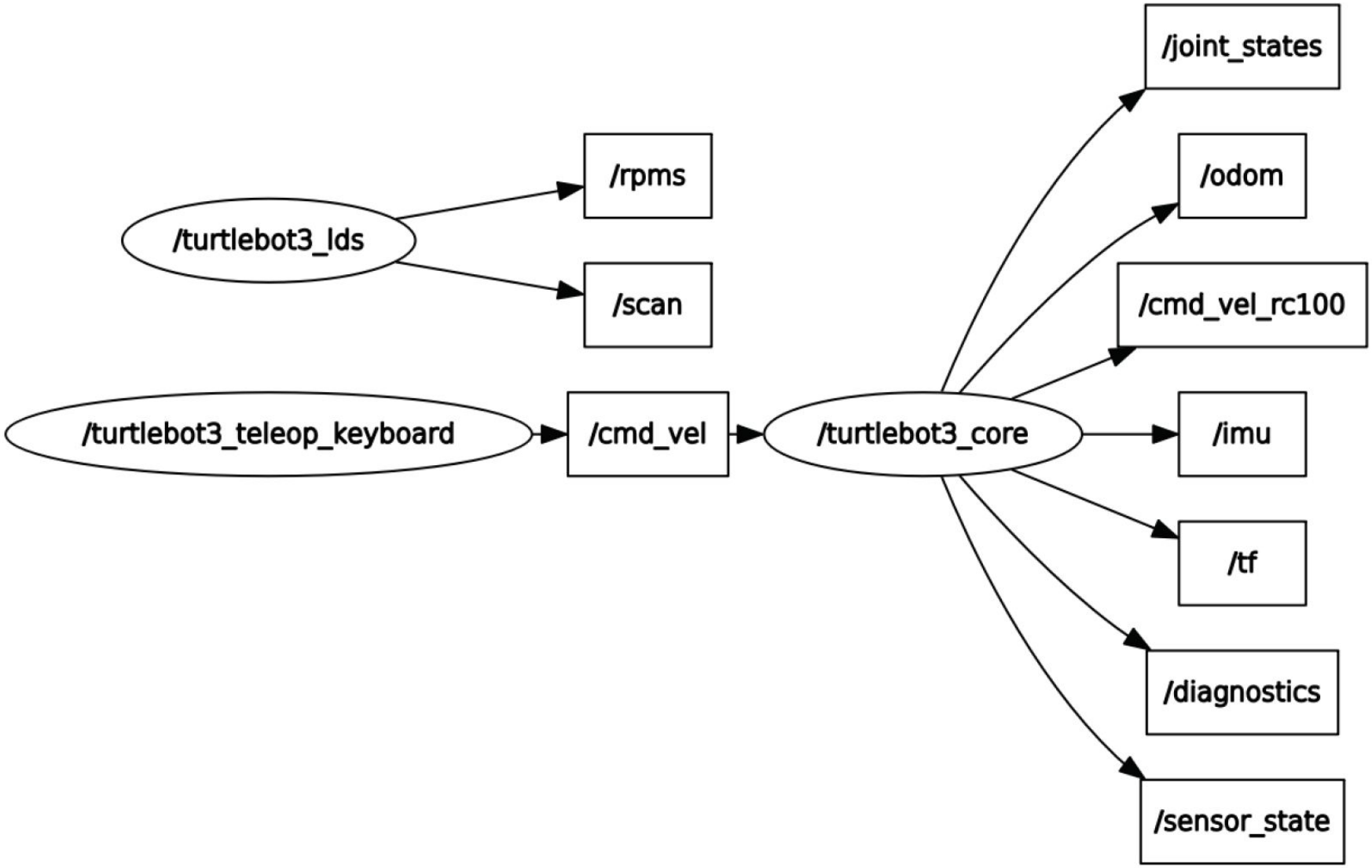
Activity: Install Turtlebot3 Packages on Remote PC

(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

https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

Turtlebot3 Nodes and Topics



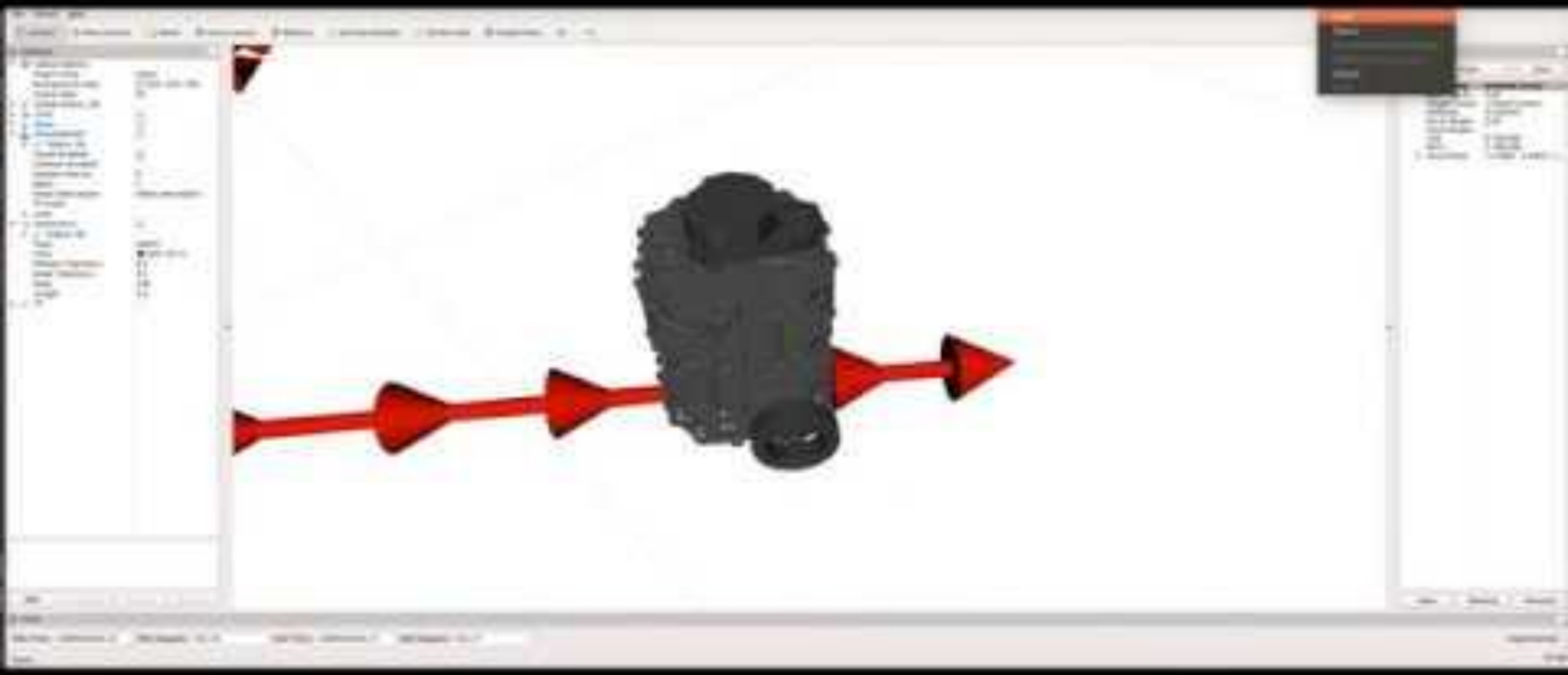
Turtlebot3 Topics

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

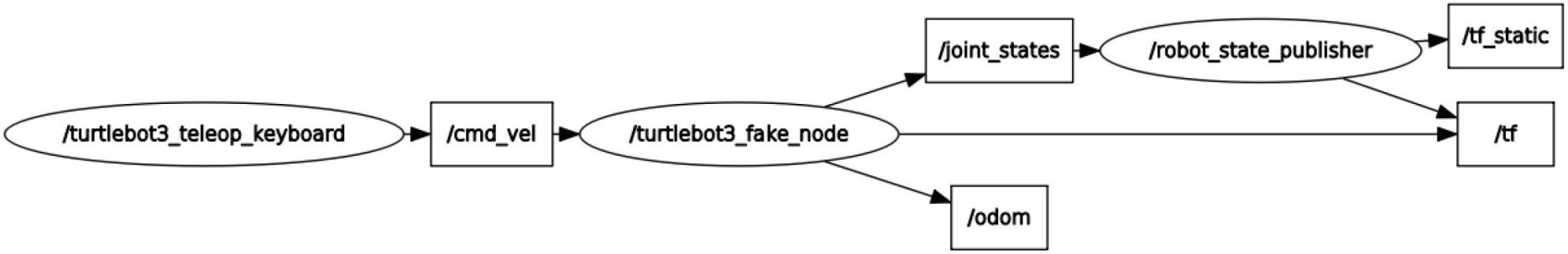
(terminal 2)
$ ~~nano ~/.bashrc~~
$ ~~export TURTLEBOT3_MODEL=burger~~
$ ~~source ~/.bashrc~~
$ roslaunch turtlebot3_fake turtlebot3_fake.launch

(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.

Gazebo Simulation

TURTLEBOT3

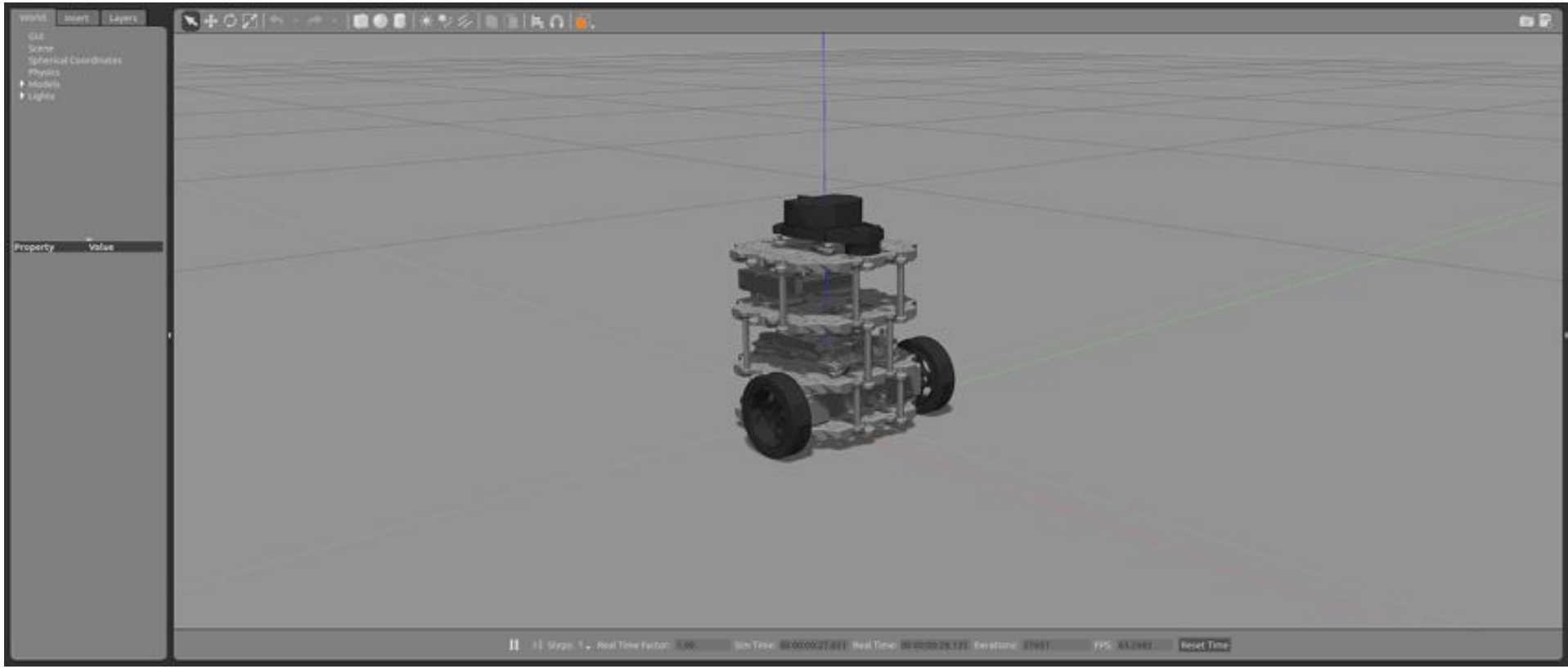
TurtleBot3
Burger

TurtleBot3
Waffle Pi

Gazebo

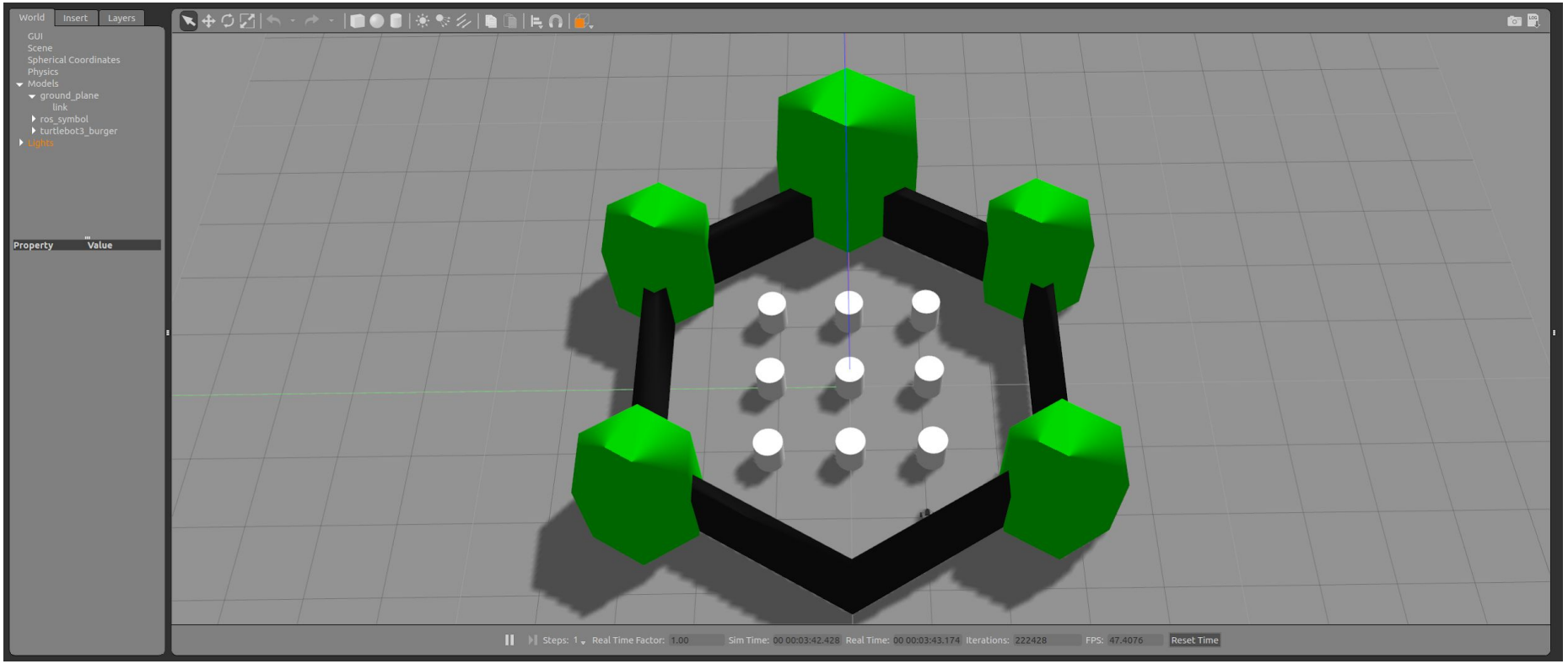
Activity: Gazebo - Empty World

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



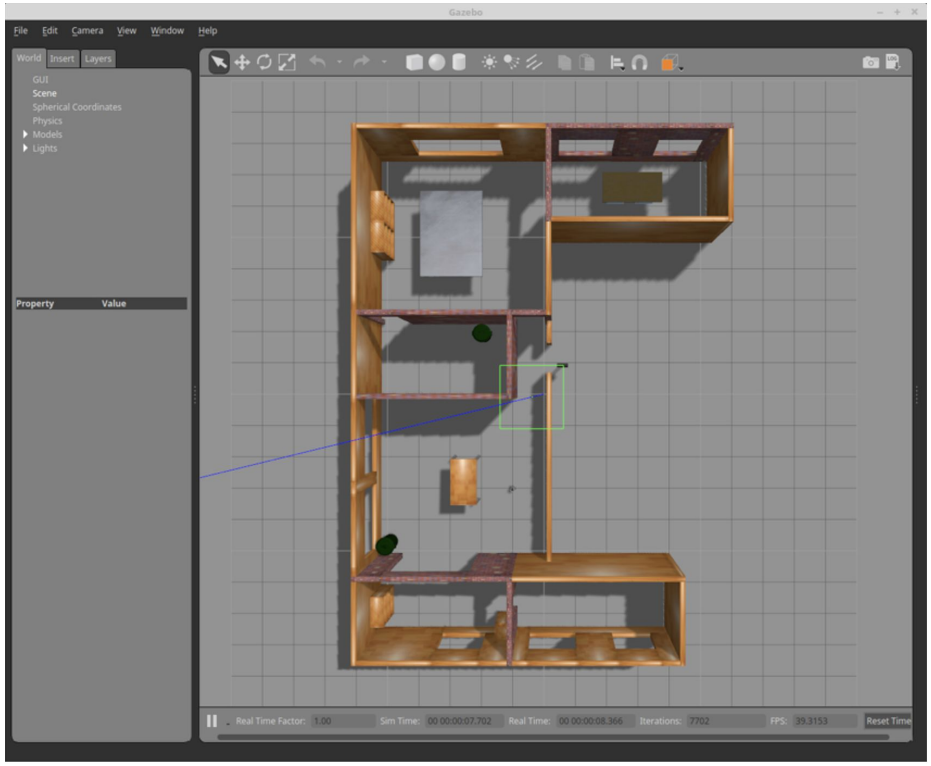
Activity: Gazebo - World

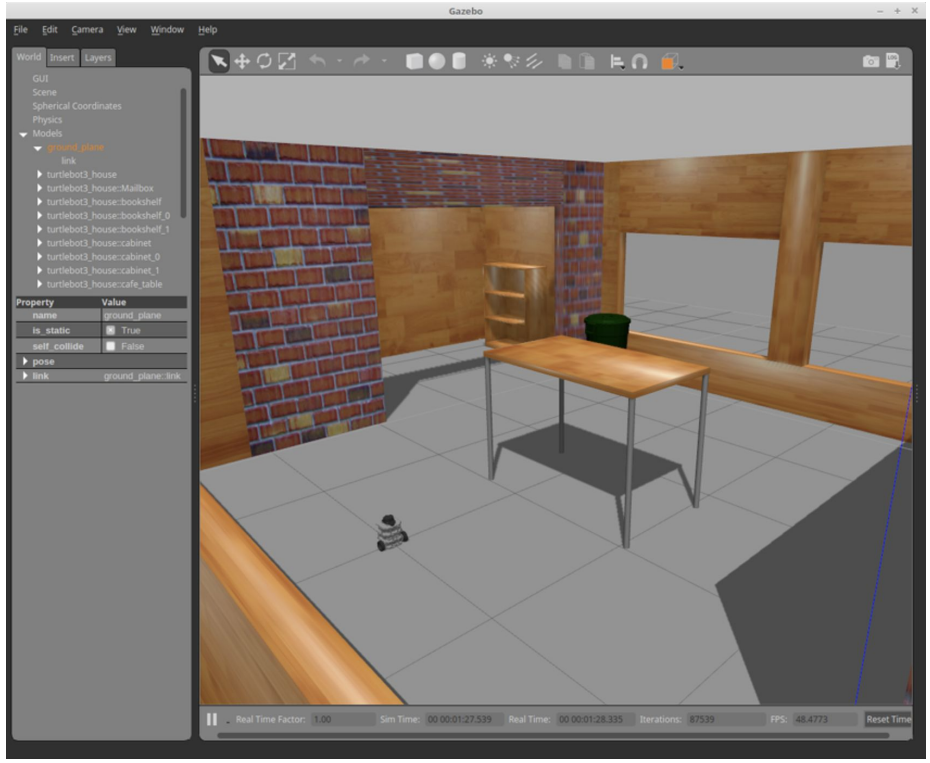
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



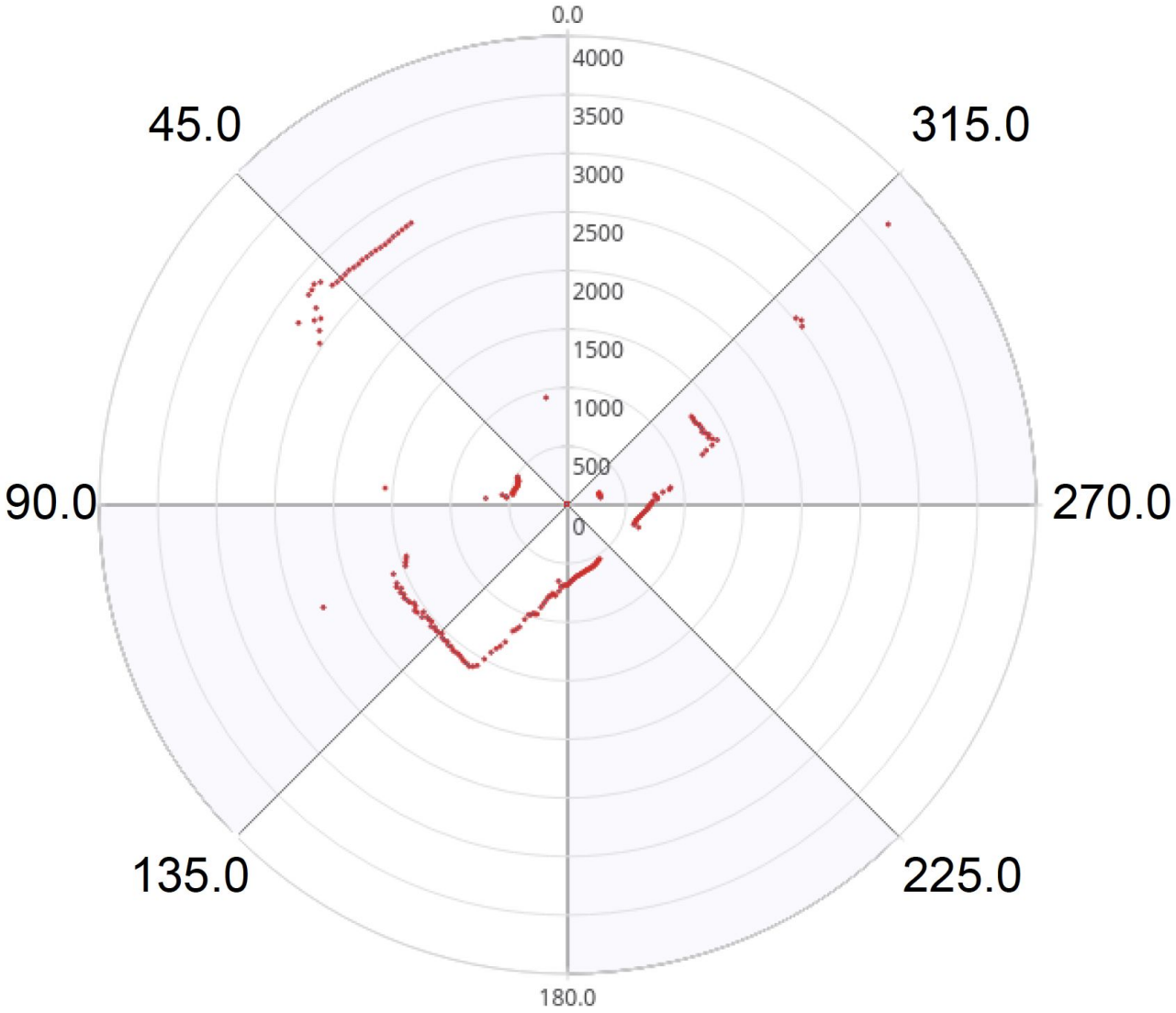
Activity: Gazebo - House

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





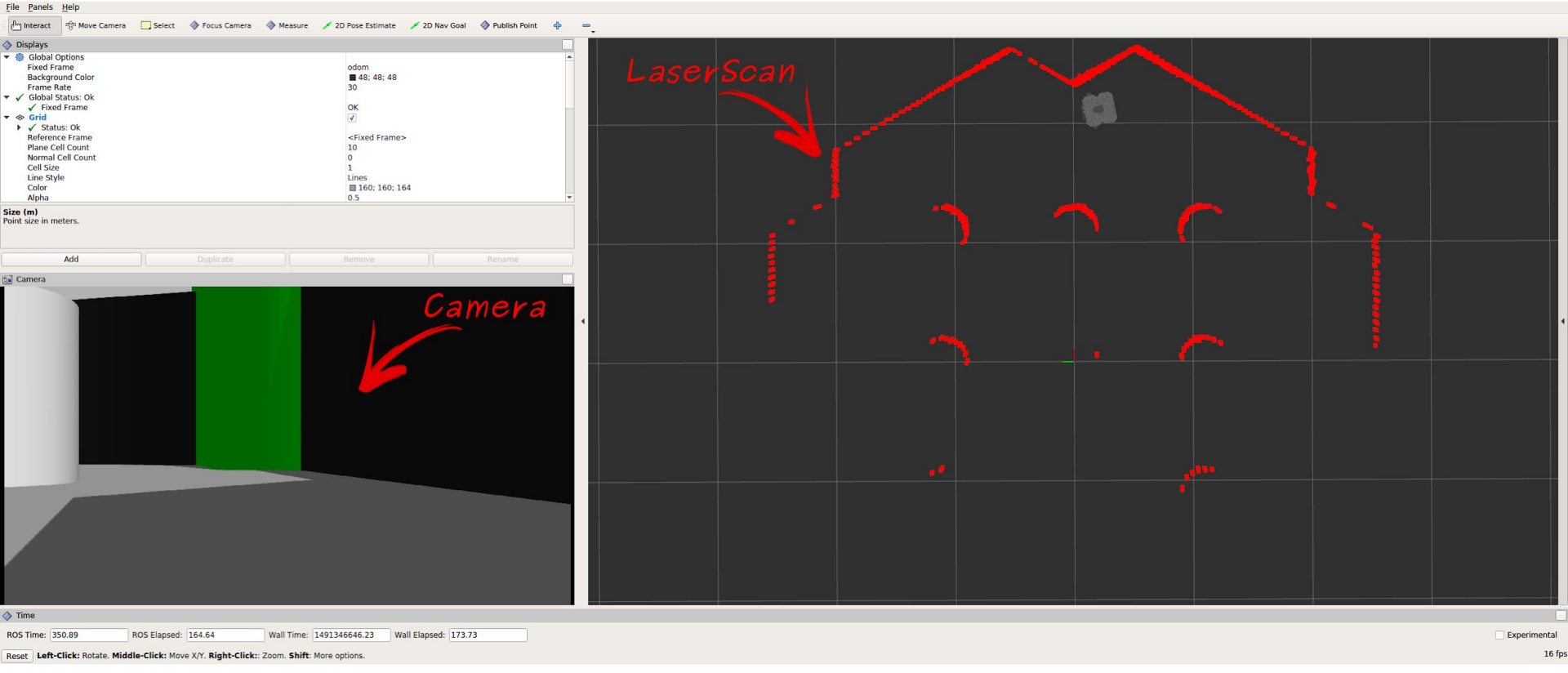
Laser Scan Data from LIDAR



Activity: LiDAR Scan Data in RViz

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

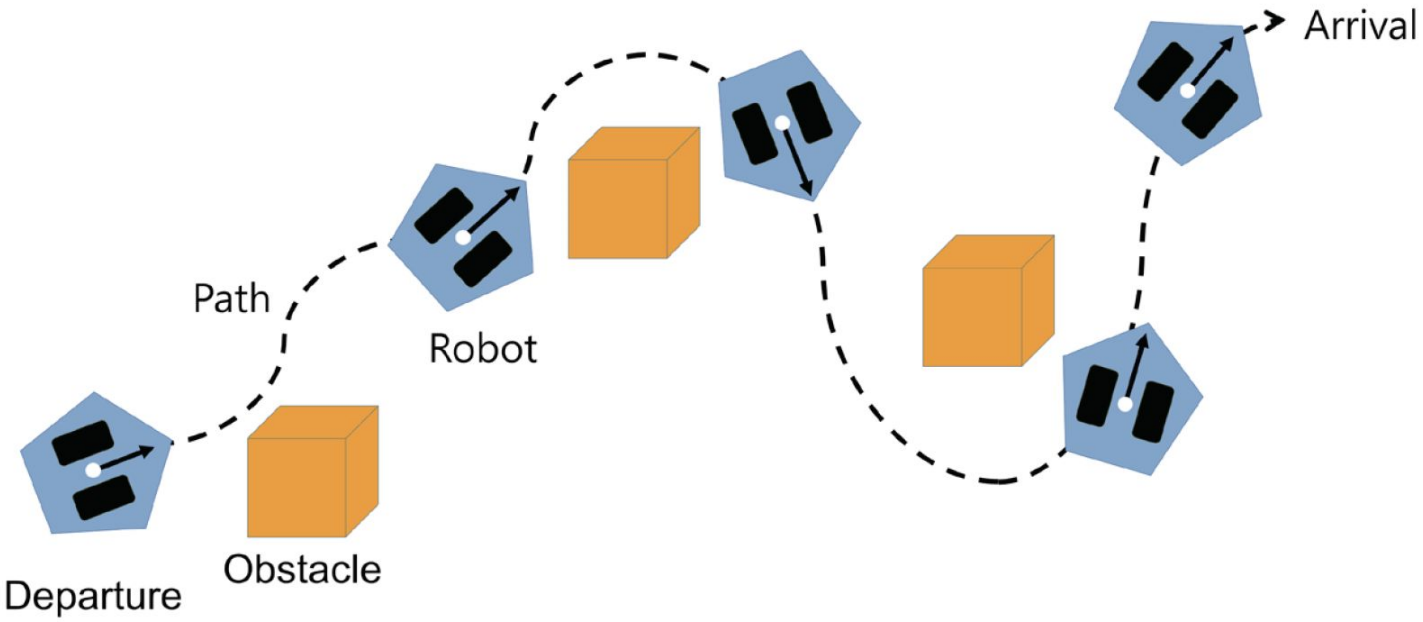
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



Topic 2
SLAM (Simultaneous Localization And Mapping)

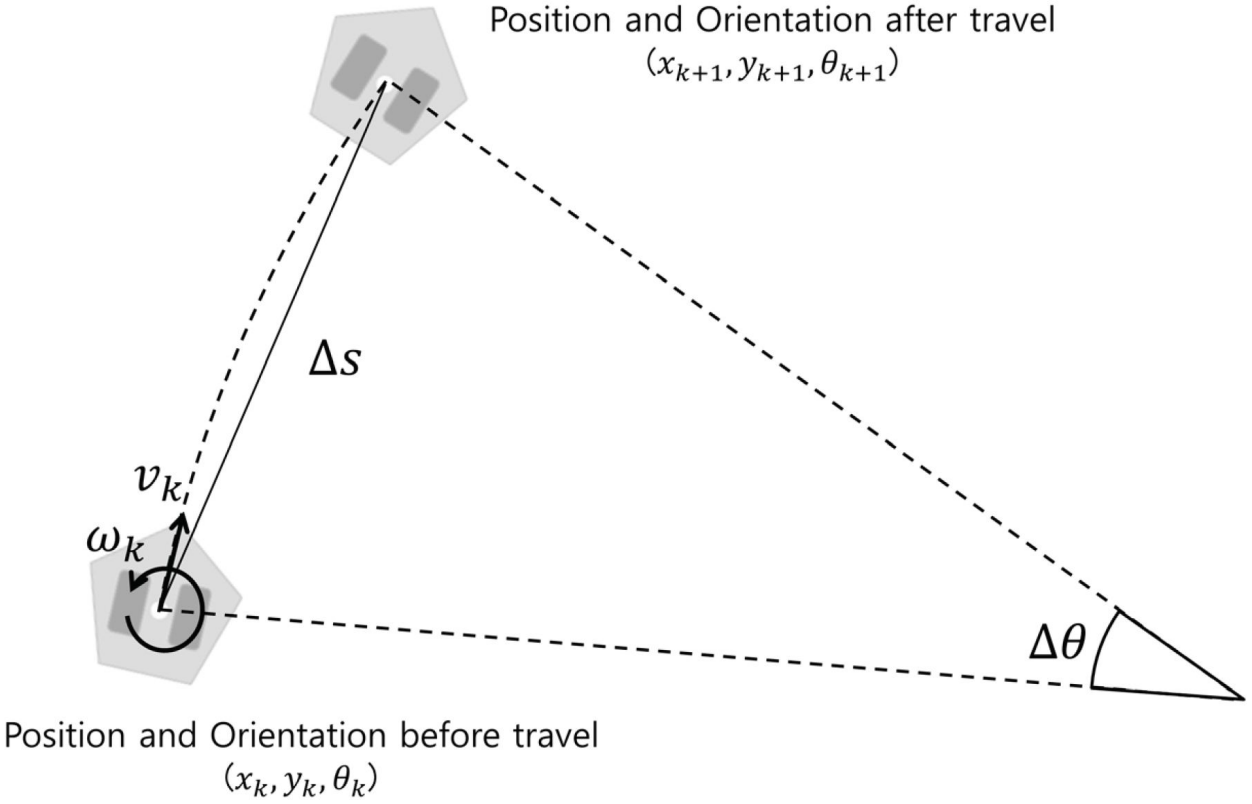
Robot Navigation

- 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



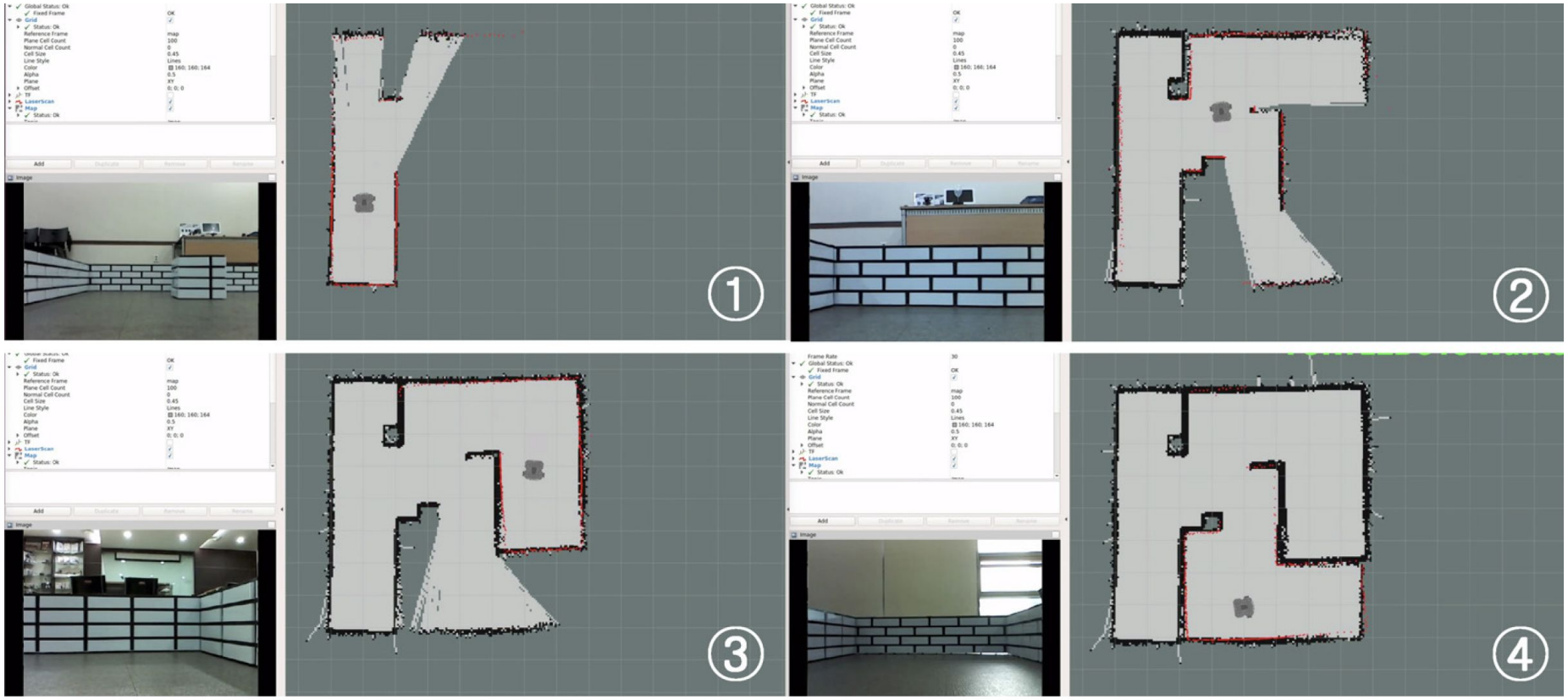
Pose of a Robot

- 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

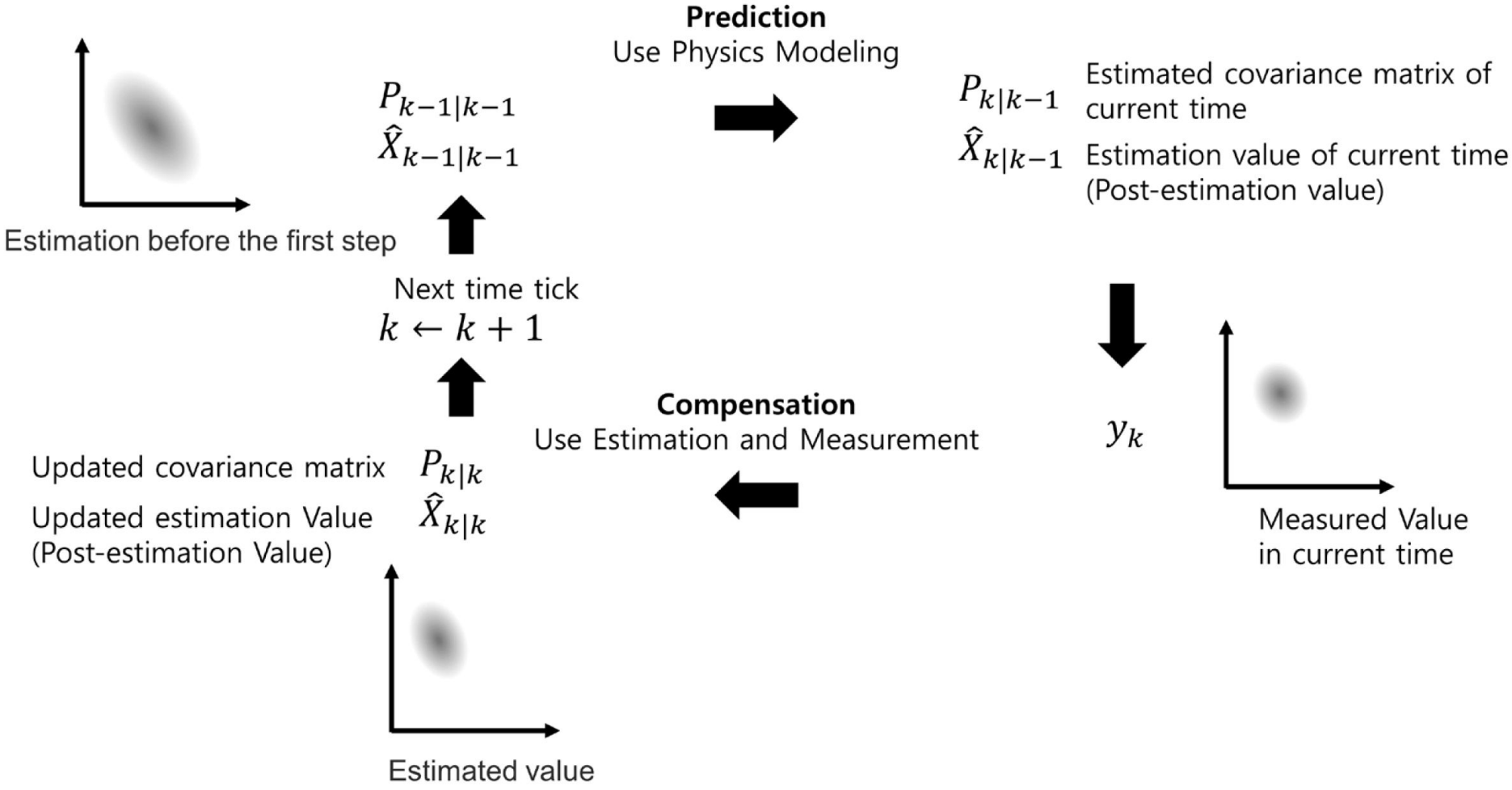
- Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.
- While this initially appears to be a chicken-and-egg problem there are several algorithms known 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.

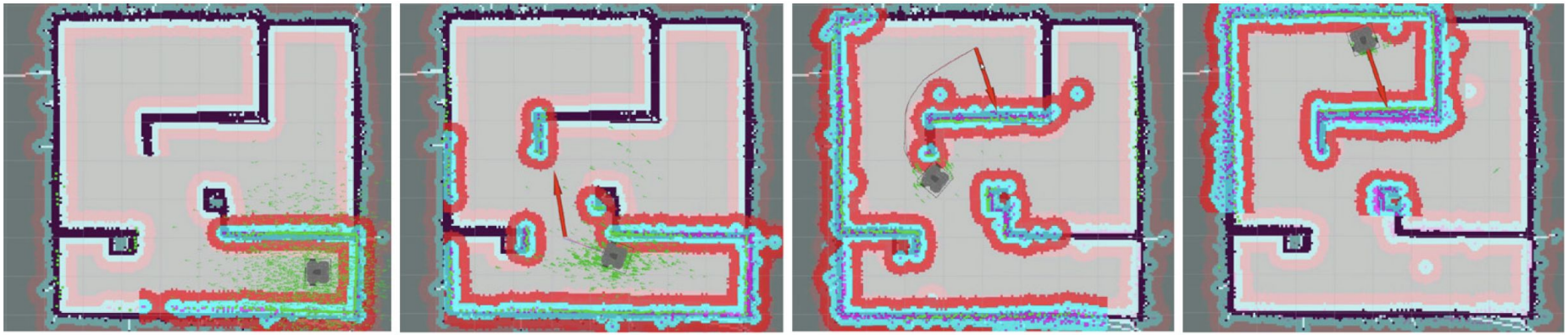


Particle Filter

- 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.

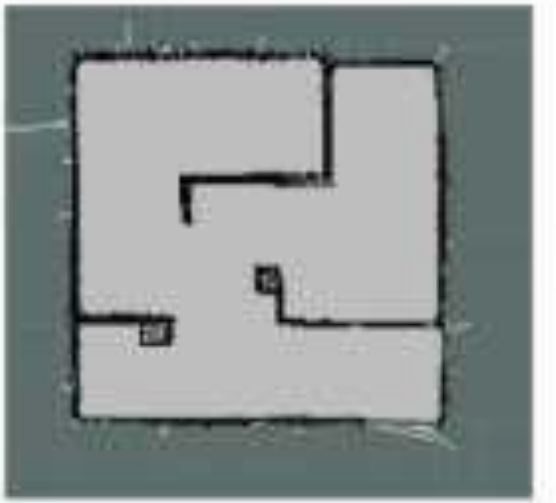
Adaptive Monte Carlo Localization

- 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)

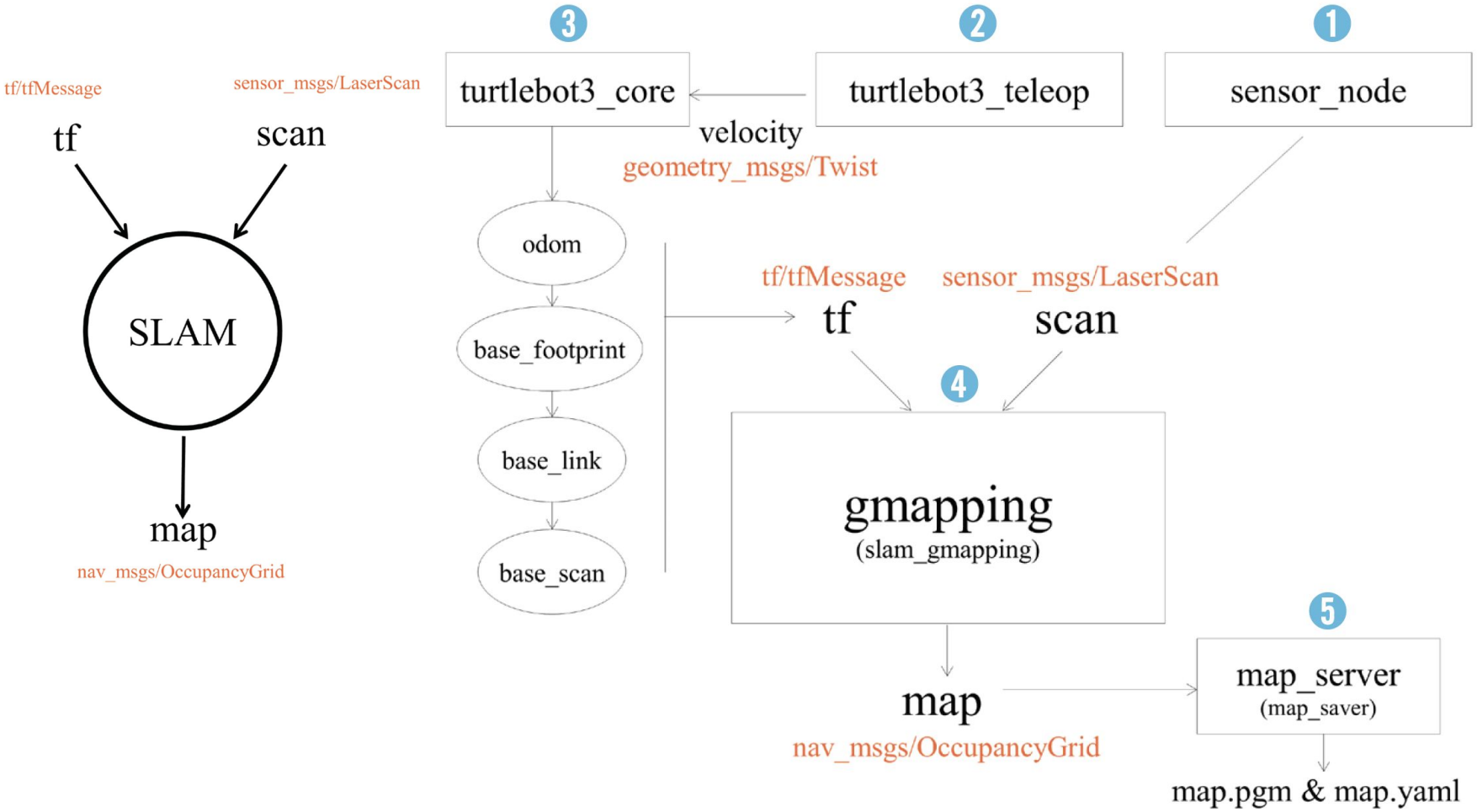
SLAM Demo



SLAM Demo

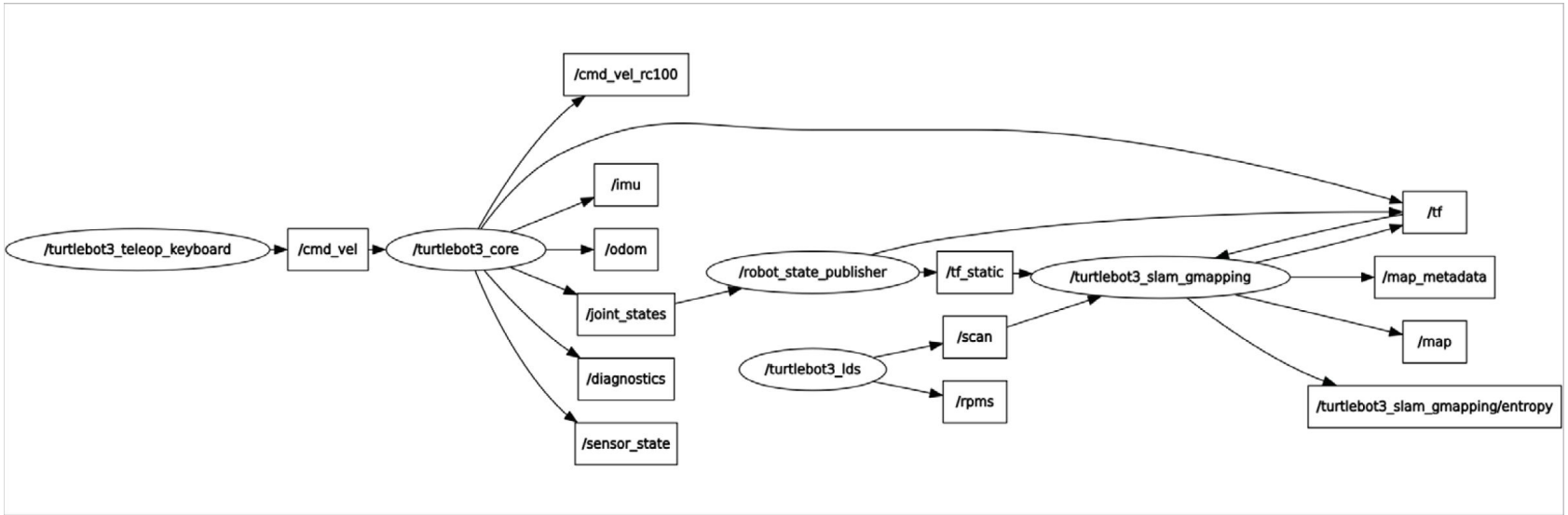
Turtlebot3 SLAM Process

- 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

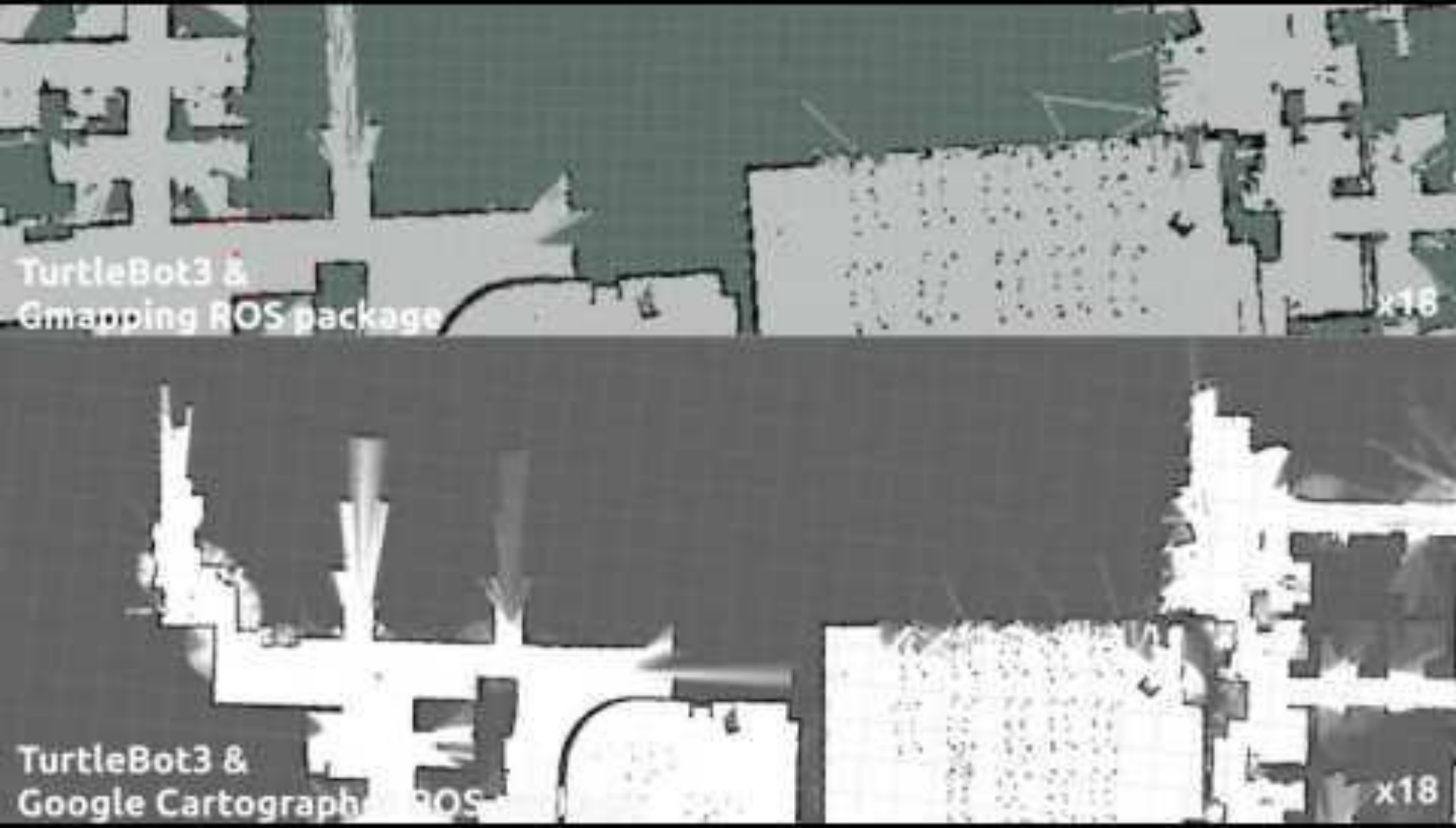


Nodes and Topics Required for SLAM

- 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.

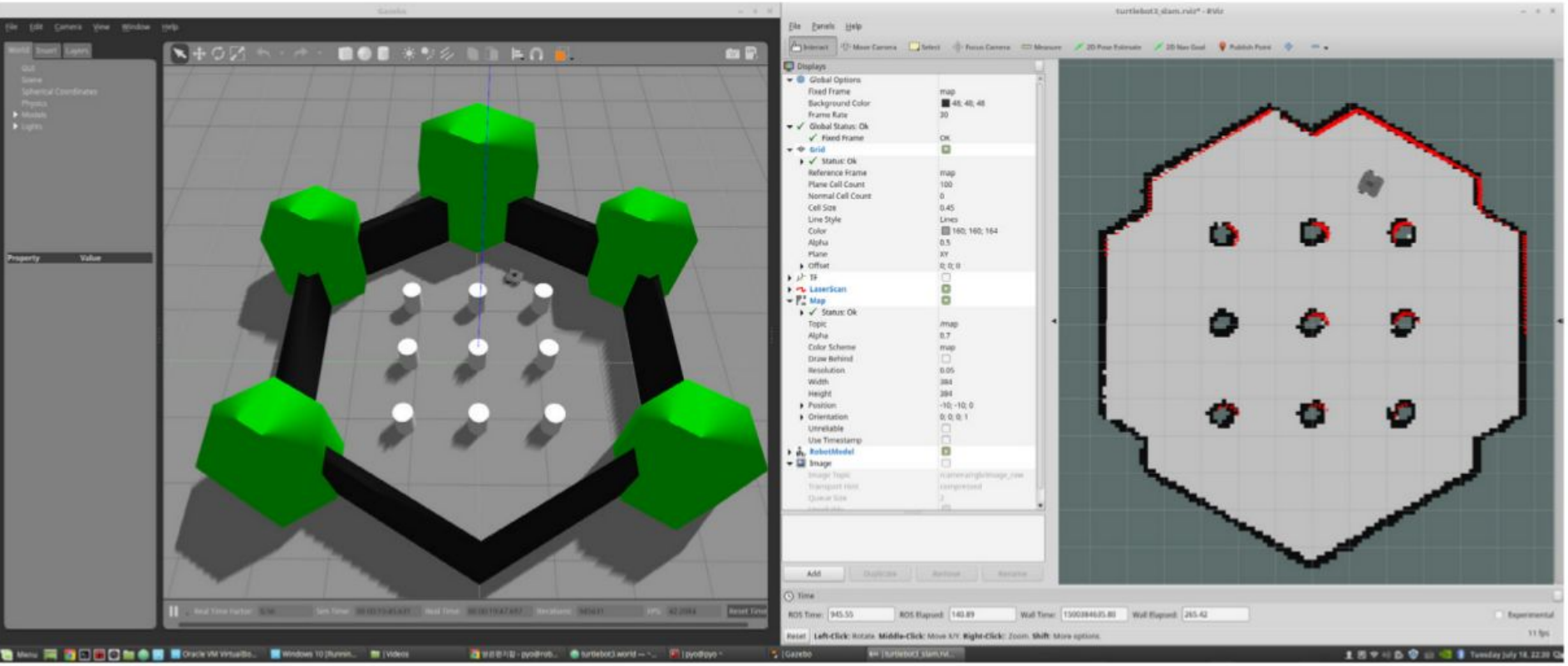


SLAM with GMapping



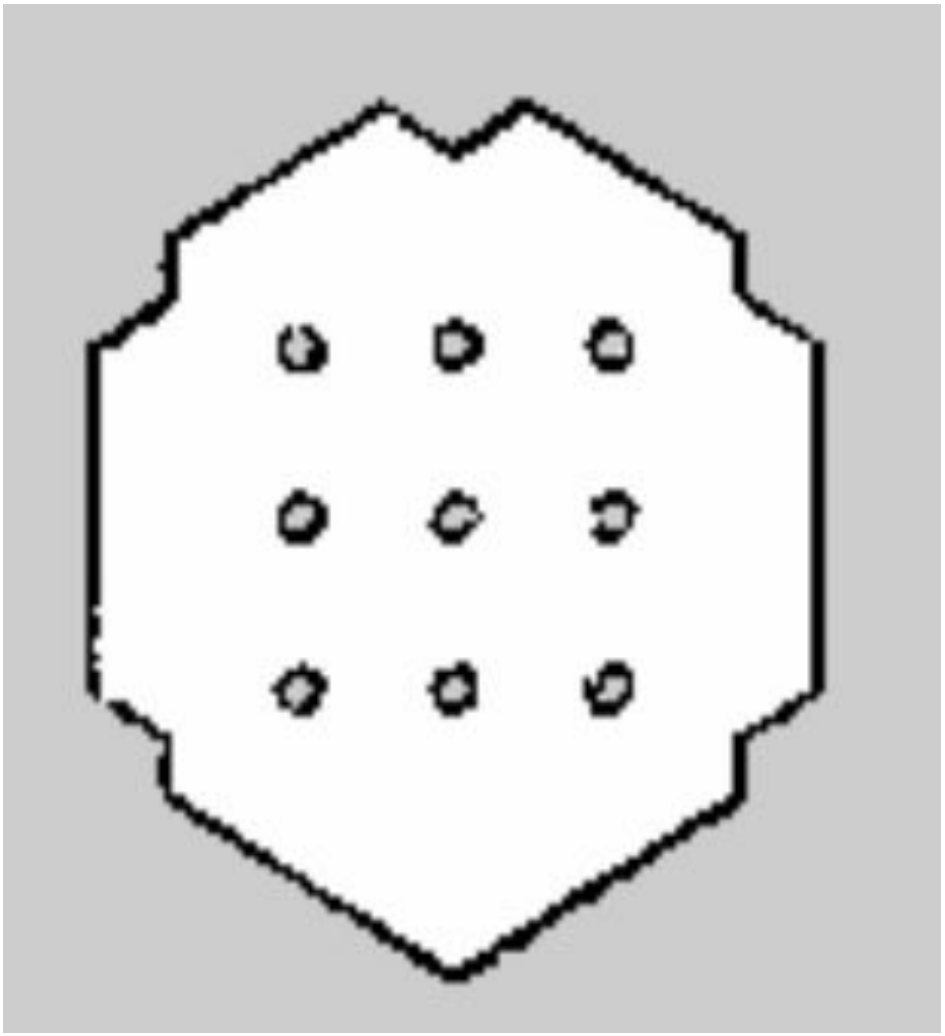
Activity: Virtual SLAM with Gmapping

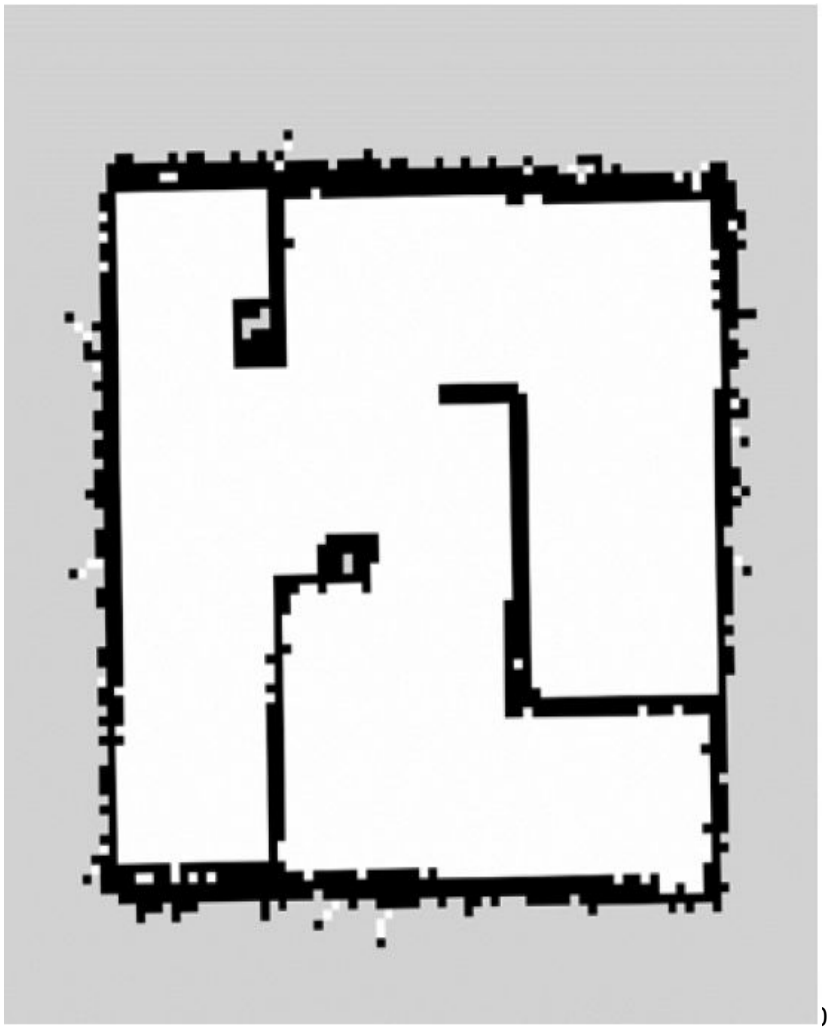
(terminal 1) roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2) roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
(terminal 3) roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
(terminal 4) rosrun map_server map_saver -f ~/my_map



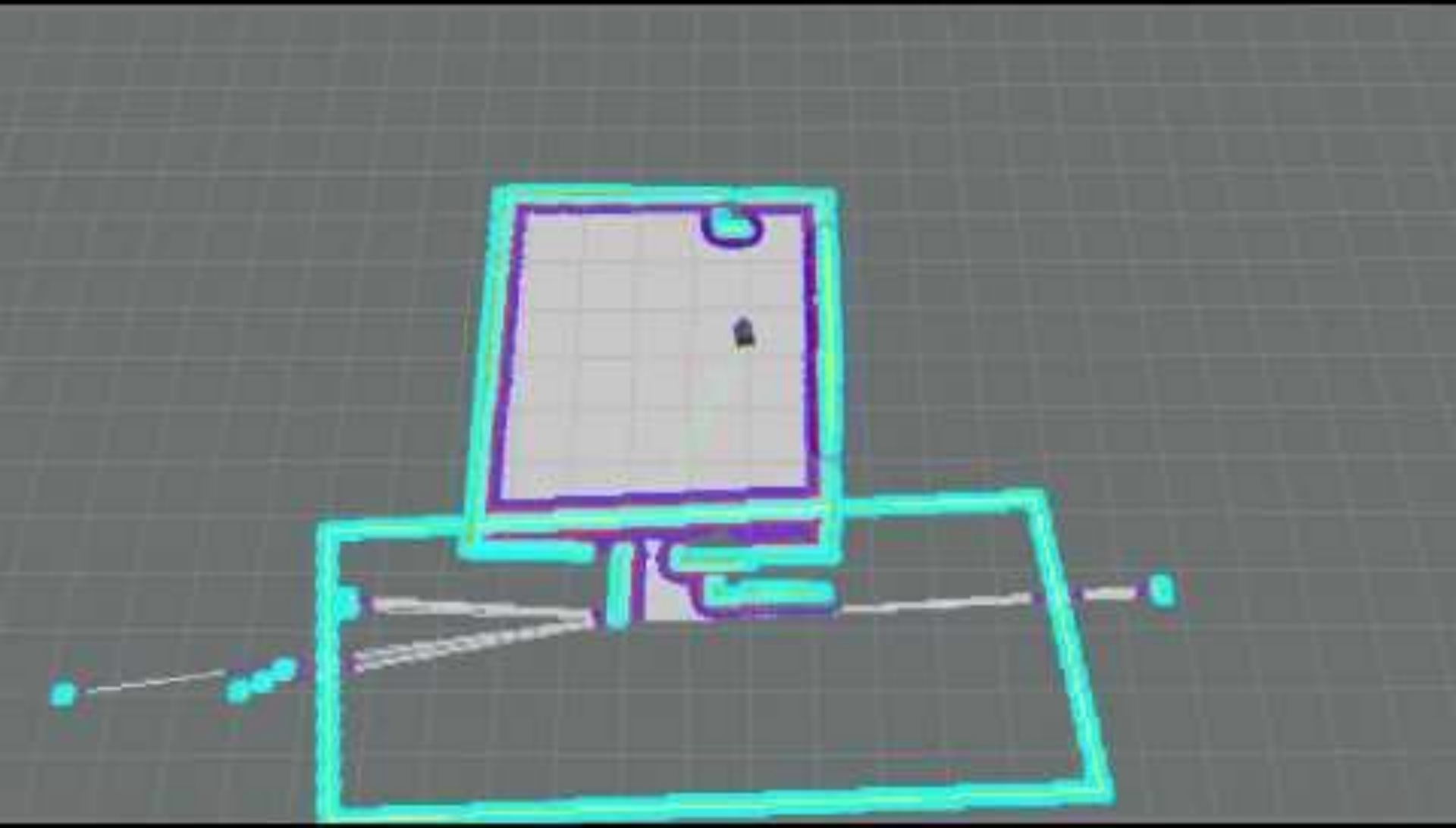
Map

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





SLAM with Frontier Exploration



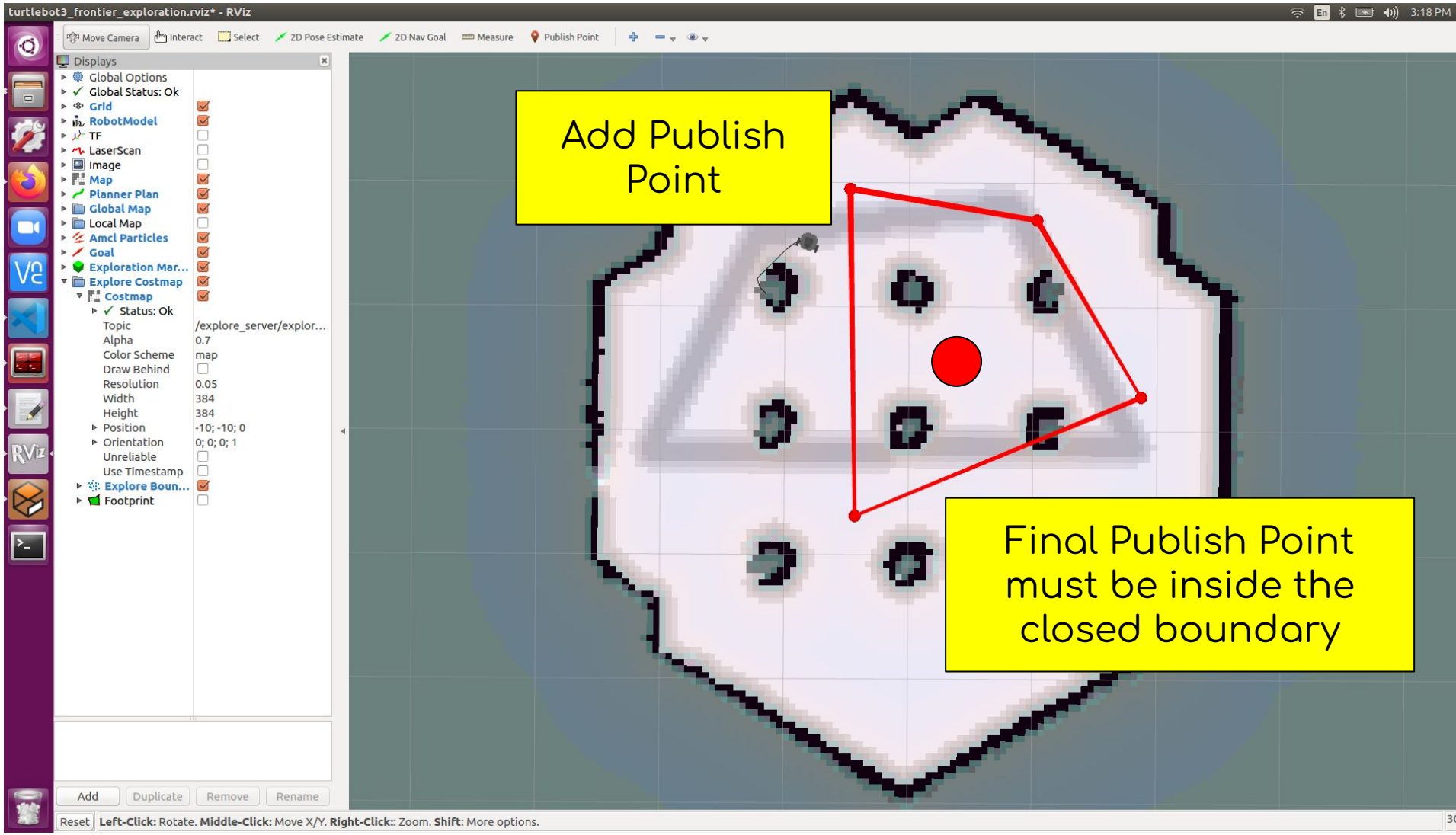
Activity: Frontier Exploration SLAM

(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_map

- 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-get install ros-kinetic-frontier-exploration ros-kinetic-navigation-stage

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

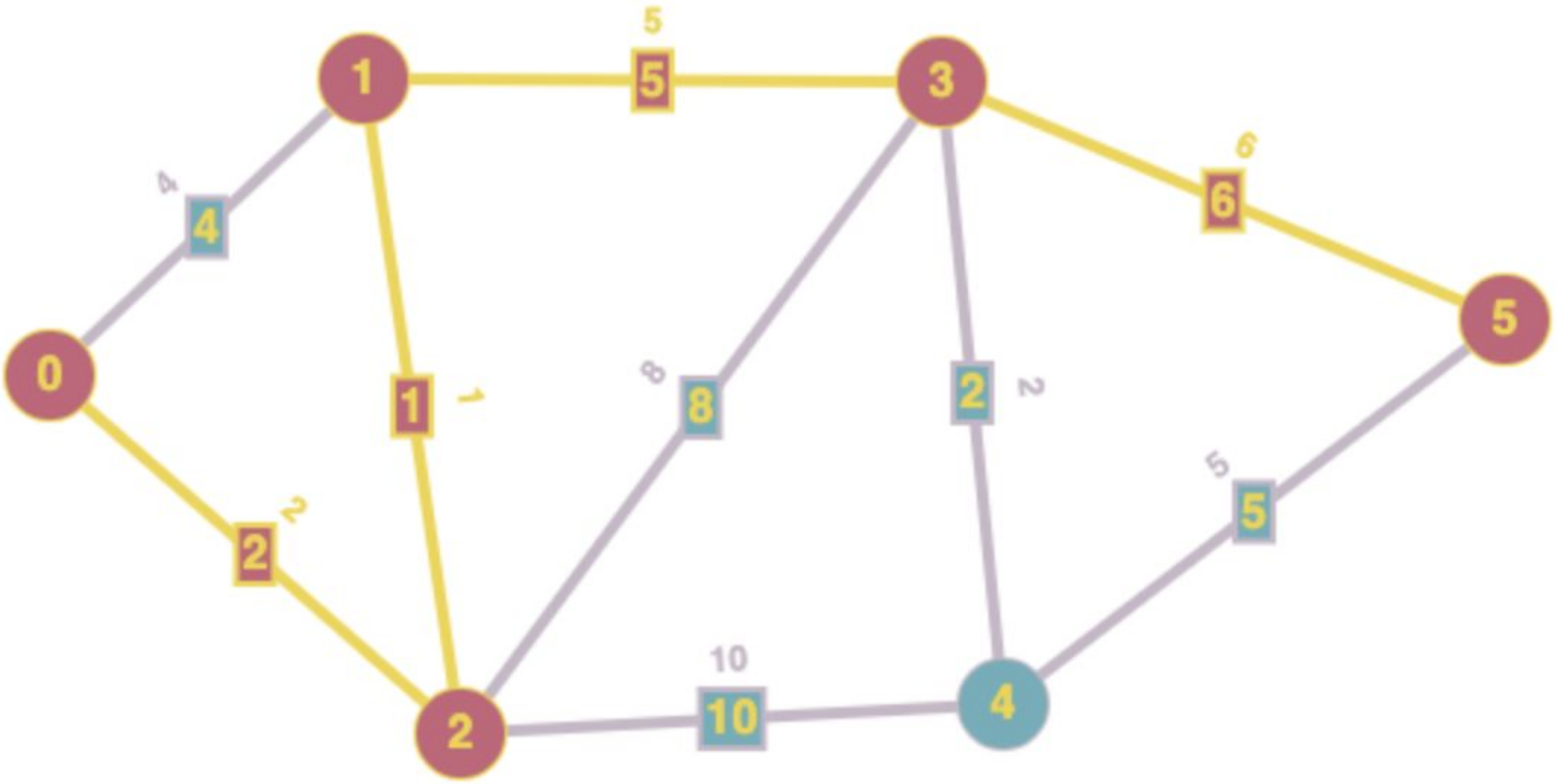
Dijkstra's Algorithm

Dijkstra's algorithm steps:

1. Mark the initial node with a current distance 0 and the rest with infinity
2. Set the unvisited node with the smallest current distance as the current node C
3. For each node N connected to the current node C: add the current distance of C with the weight of the edge connecting C. If it's smaller than the current distance of N, update it as the new current distance of N.
4. Mark the current node C as visited.
5. If there are unvisited nodes. Go to step 2

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

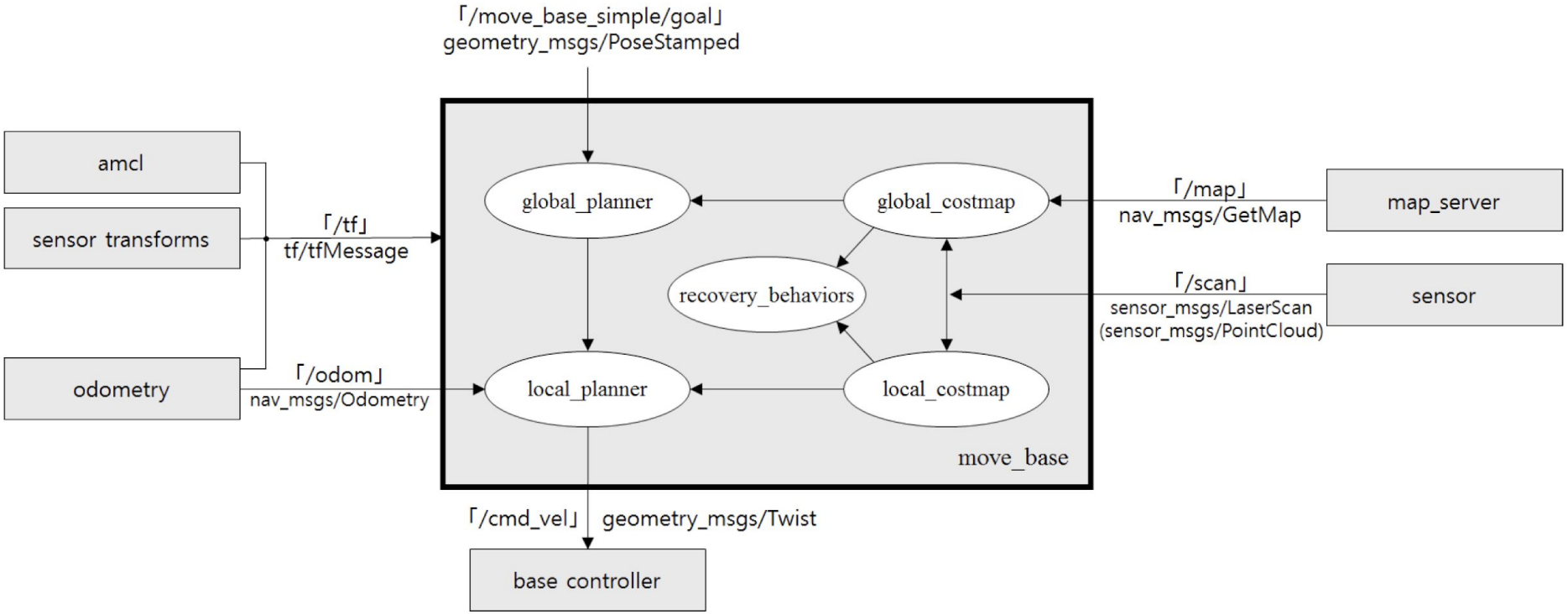


Navigation Demo

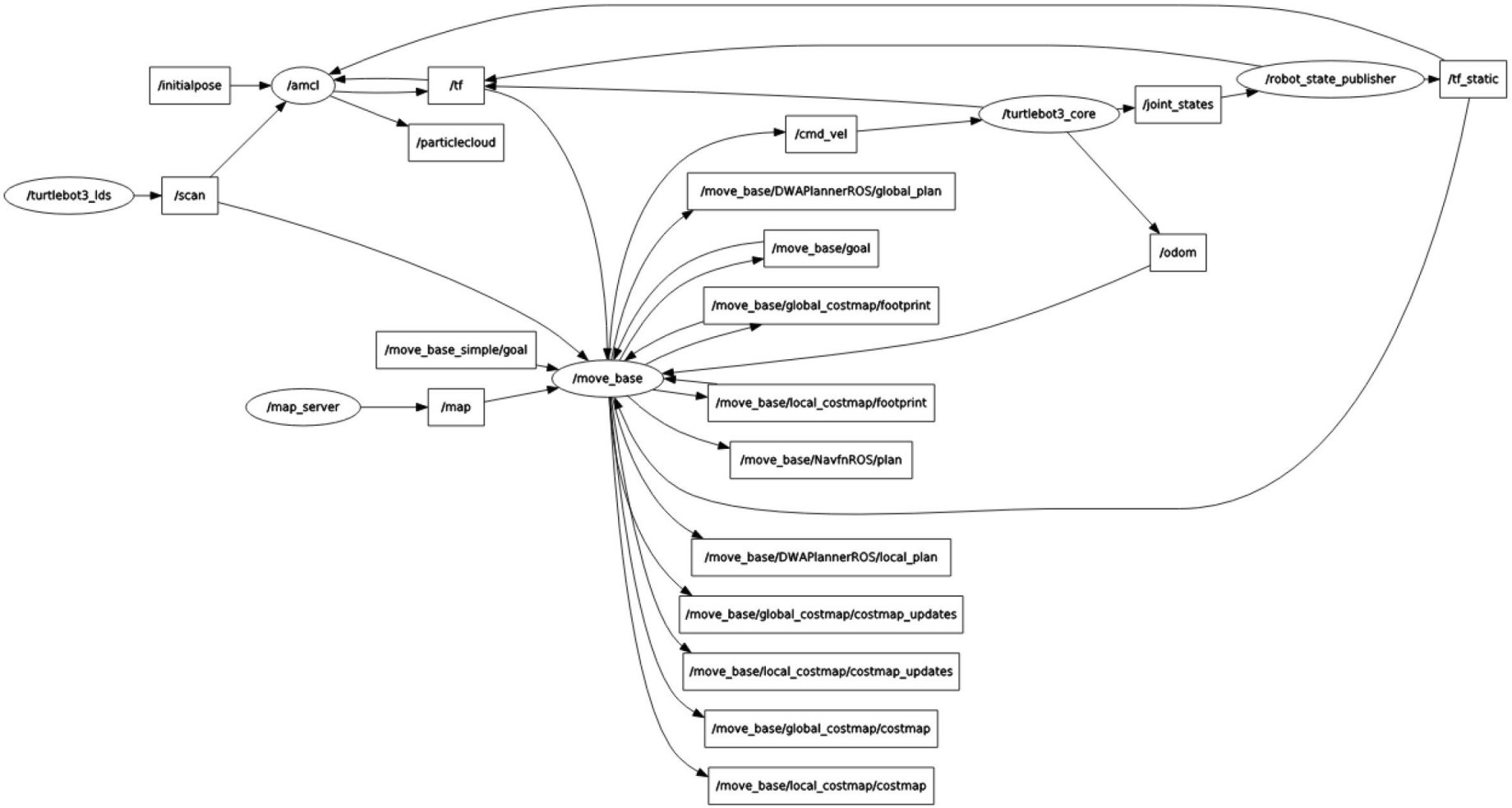
Navigation Demo

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).

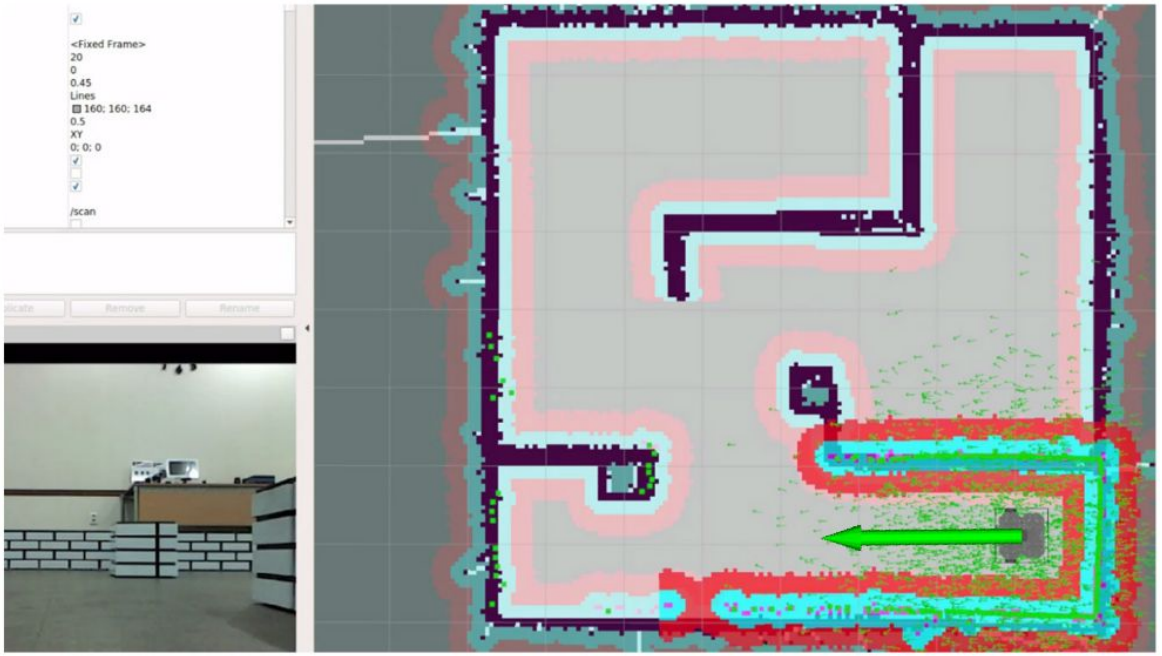


Node and Topic of Turtlebot3 Navigation



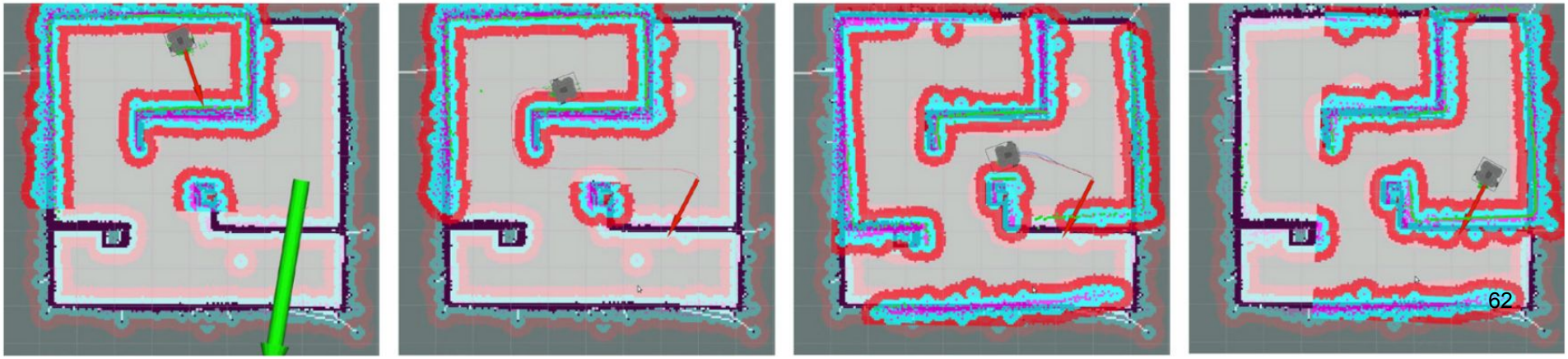
Estimate Initial Pose

- 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.

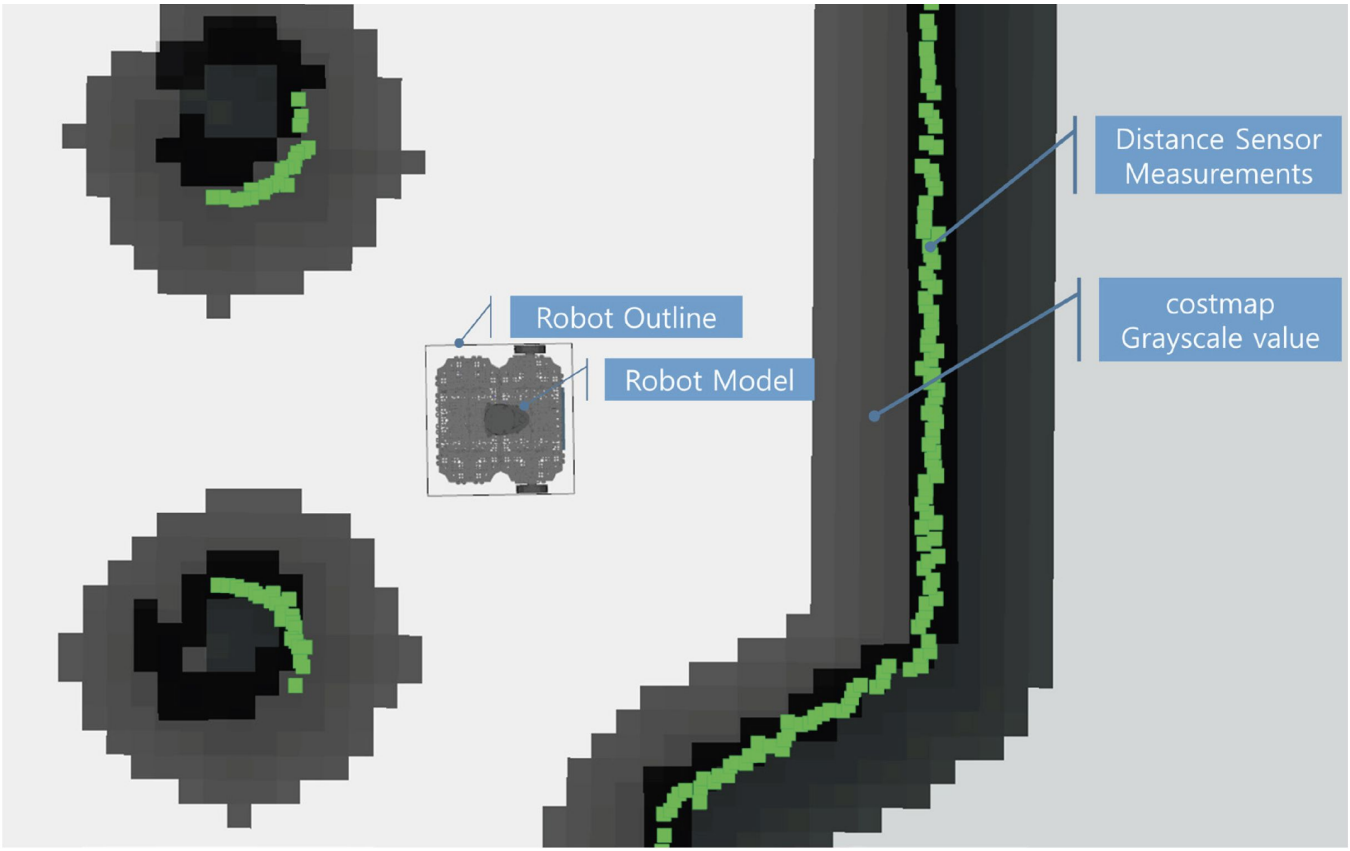


Costmap

- 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

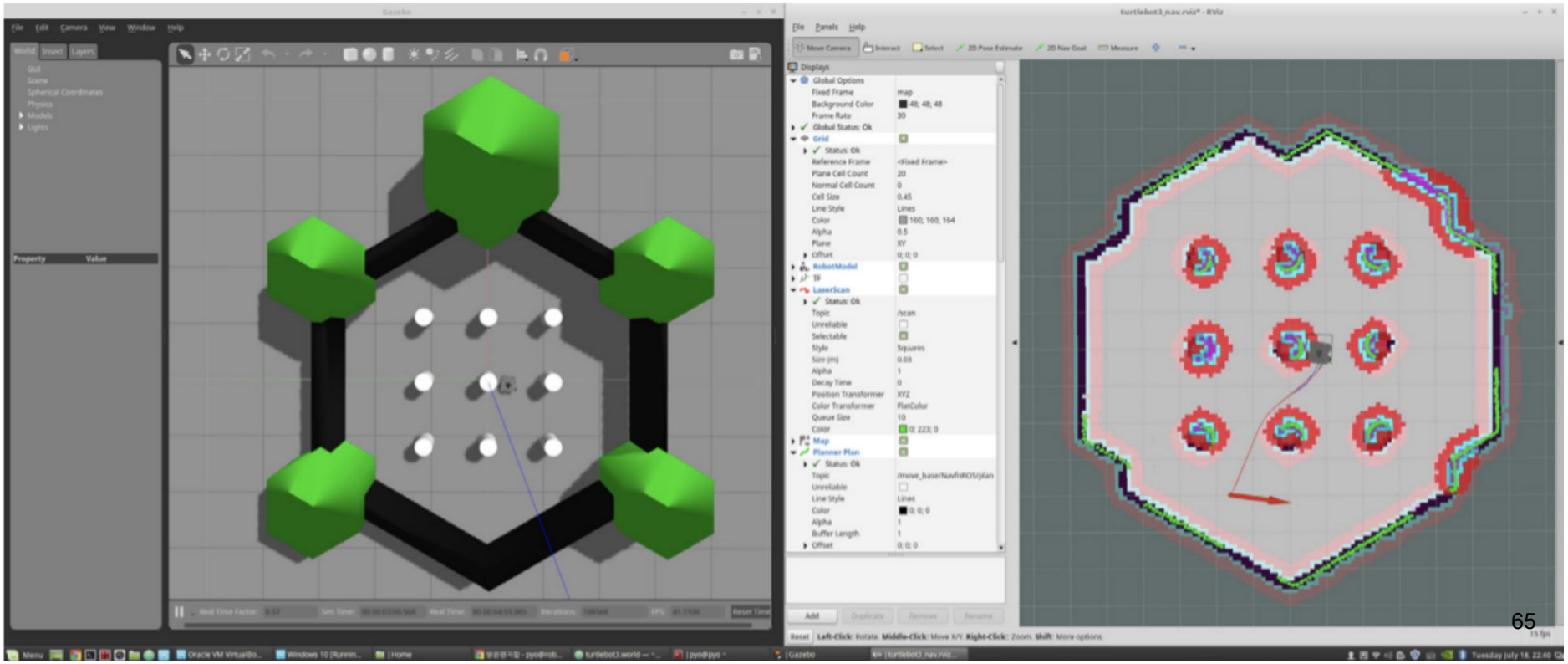
- There is a robot model in the middle and the rectangular box around it corresponds to the outer surface of the robot. When this outline line contacts the wall, the robot will bump into the wall as well.
- Green represents the obstacle with the distance sensor value obtained from the laser senso.
- 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

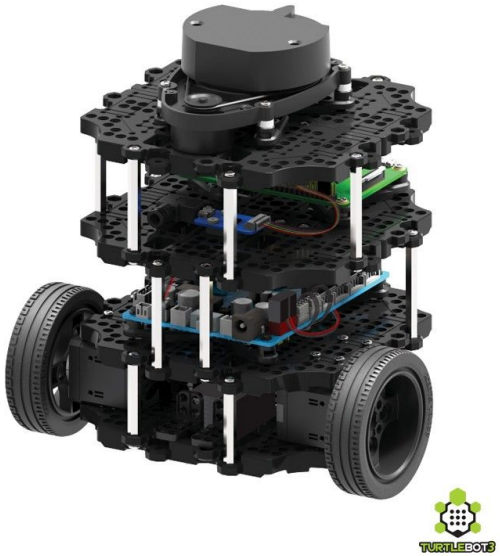
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/my_map.yaml



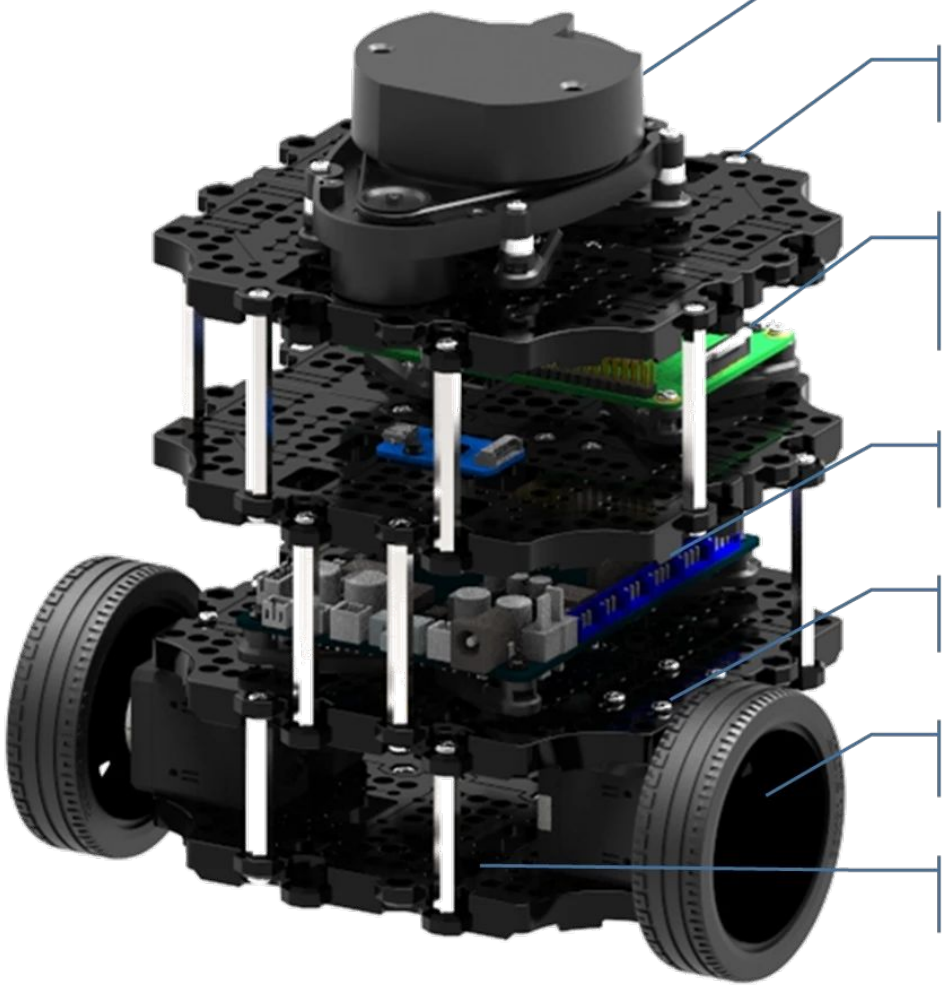
Topic 3
Physical Robot Navigation

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

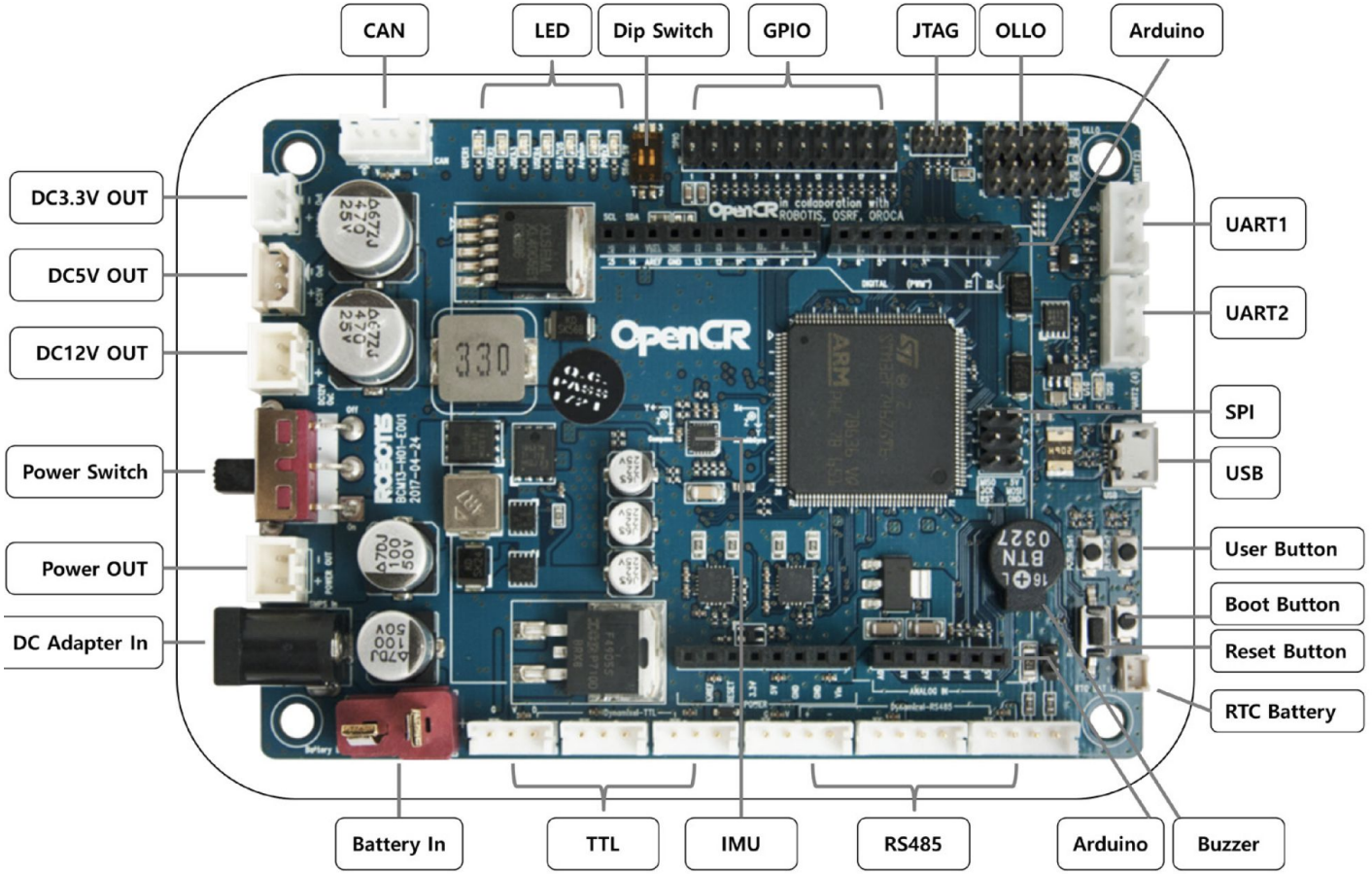


- 360° LiDAR for SLAM & Navigation
- Scalable Structure
- Single Board Computer (Raspberry Pi)
- OpenCR (ARM Cortex-M7)
- DYNAMIXEL x 2 for Wheels
- Sprocket Wheels for Tire and Caterpillar
- Li-Po Battery 11.1V 1,800mAh

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.

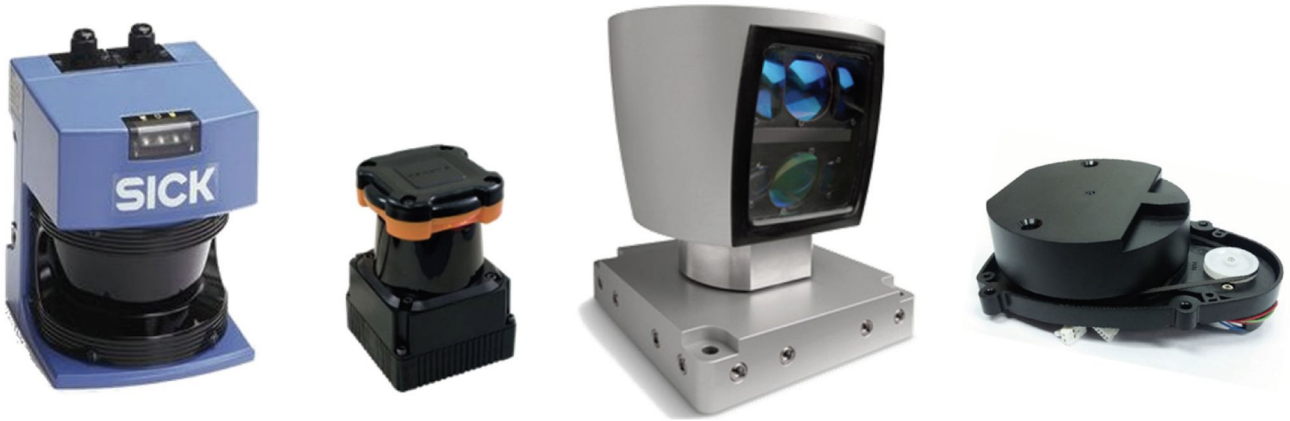


Characteristics of OpenCR Module

- 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

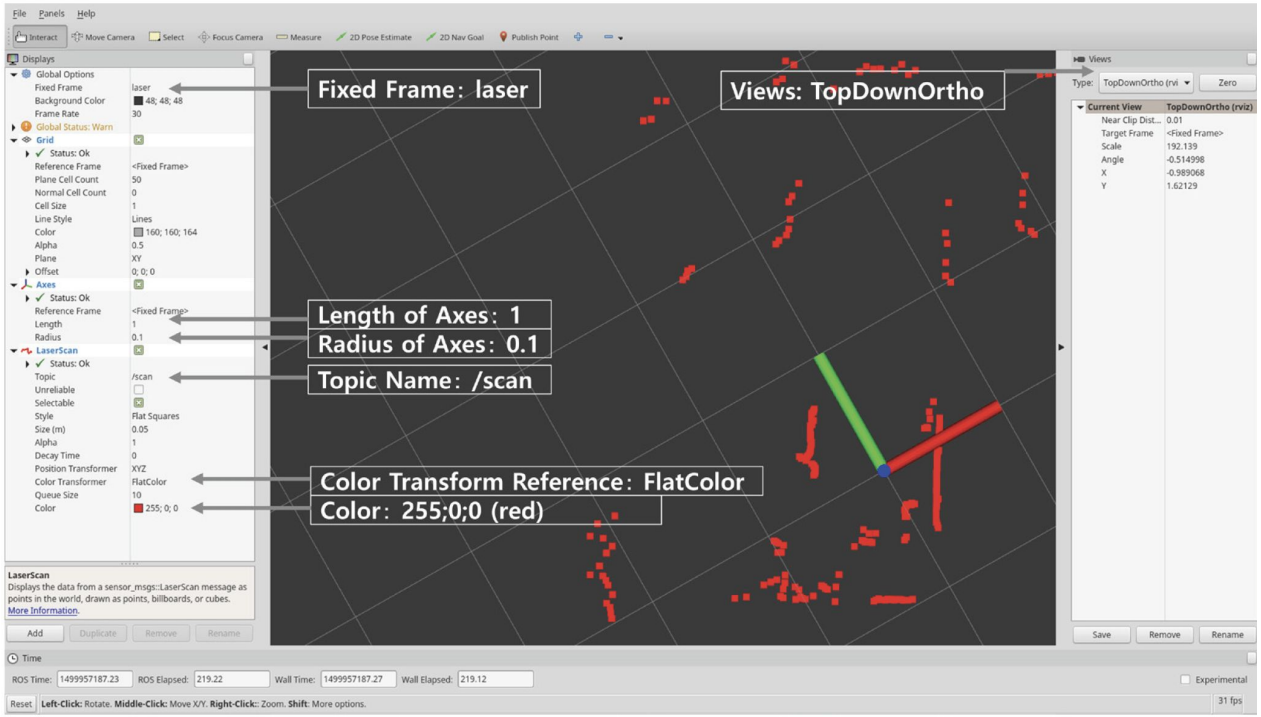
- Laser Distance Sensors (LDS) are referred to various names such as Light Detection And Ranging (LiDAR), Laser Range Finder (LRF) and Laser Scanner. LDS is a sensor used to measure the distance to an object using a laser as its source.
- The LDS sensor has the advantage of high performance, high speed, 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

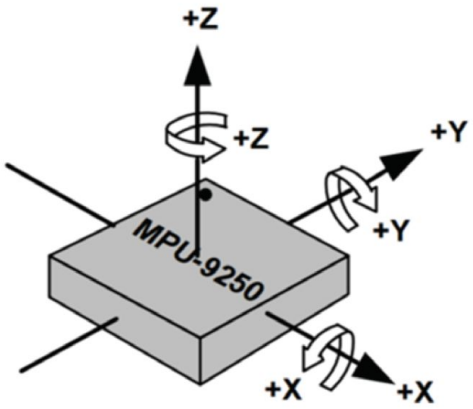
- We are going to test using LDS (HLS-LFCD2) from HLDS (Hitachi-LG Data Storage), so you must install ‘hls_lfcd_lds_driver’ package
- Execute the following commands to the scan data

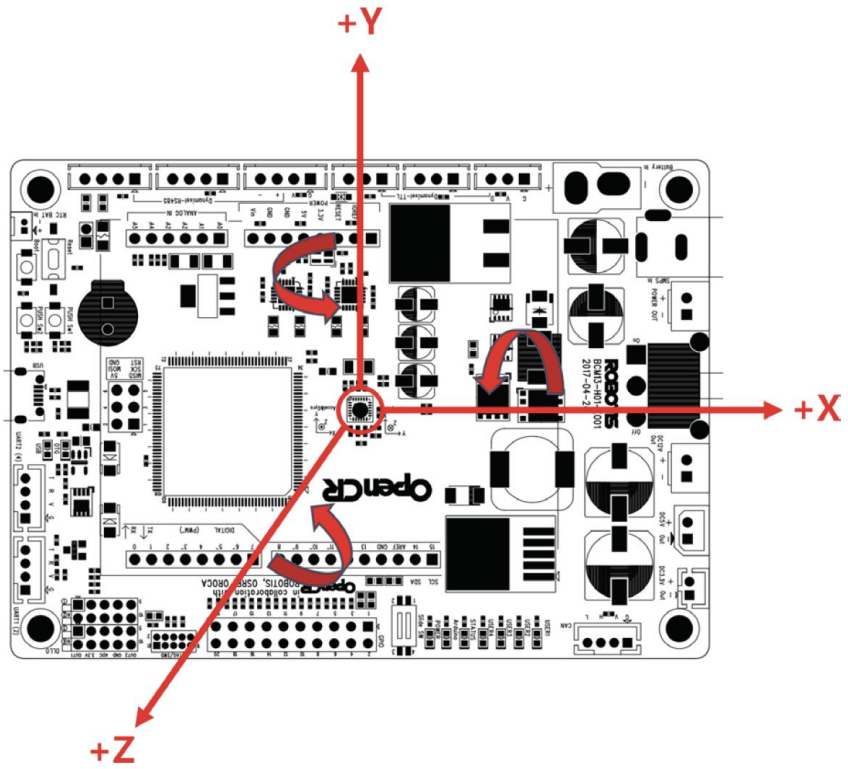
  sudo apt-get install ros-kinetic-hls-lfcd-lds-driver
  roslaunch hls_lfcd_lds_driver hlds_laser.launch
  rostopic echo /scan
  rviz



IMU Sensor

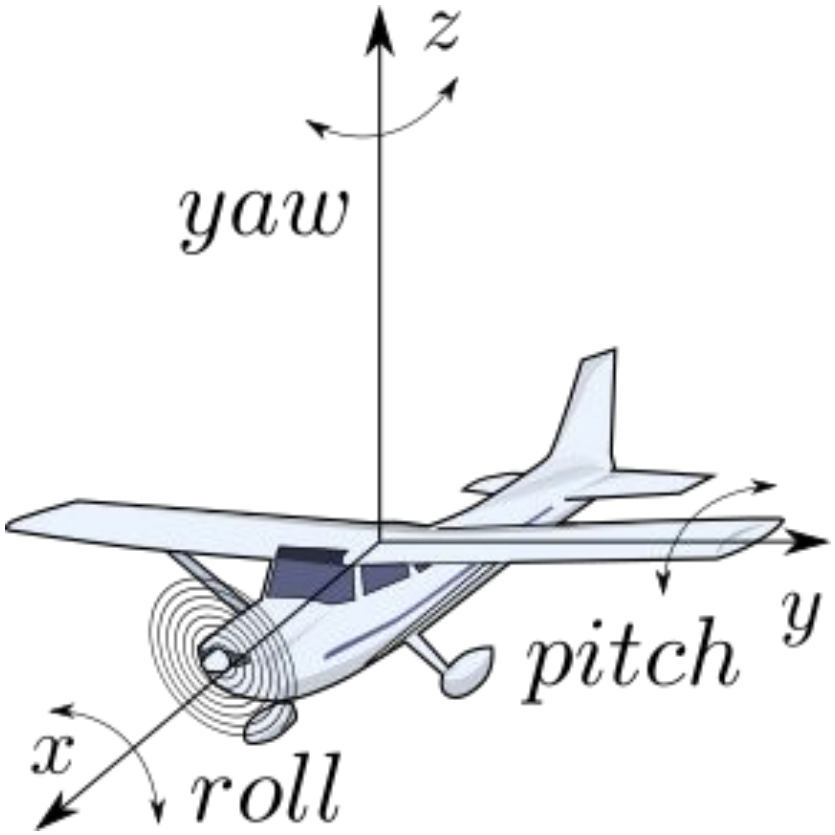
- MPU9250 sensor from InvenSense is located at the center of the OpenCR board for accurate measurement.
- The MPU9250 has built-in gyroscope, accelerometer, and magnetometer sensors on one chip.

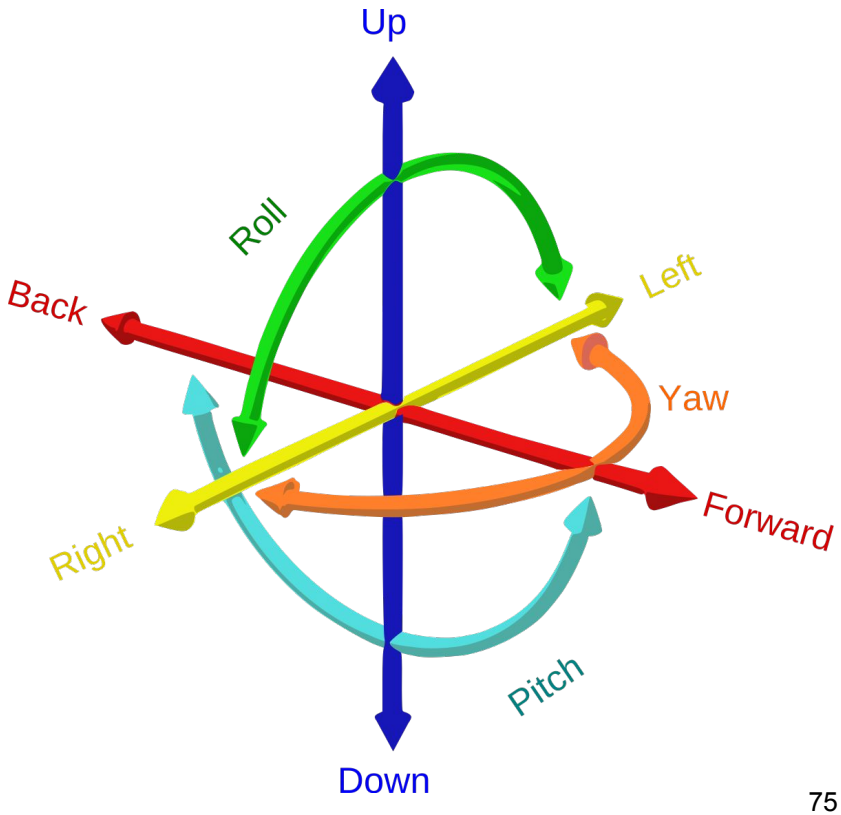




Inertial Measurement Units (IMU)

- 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)



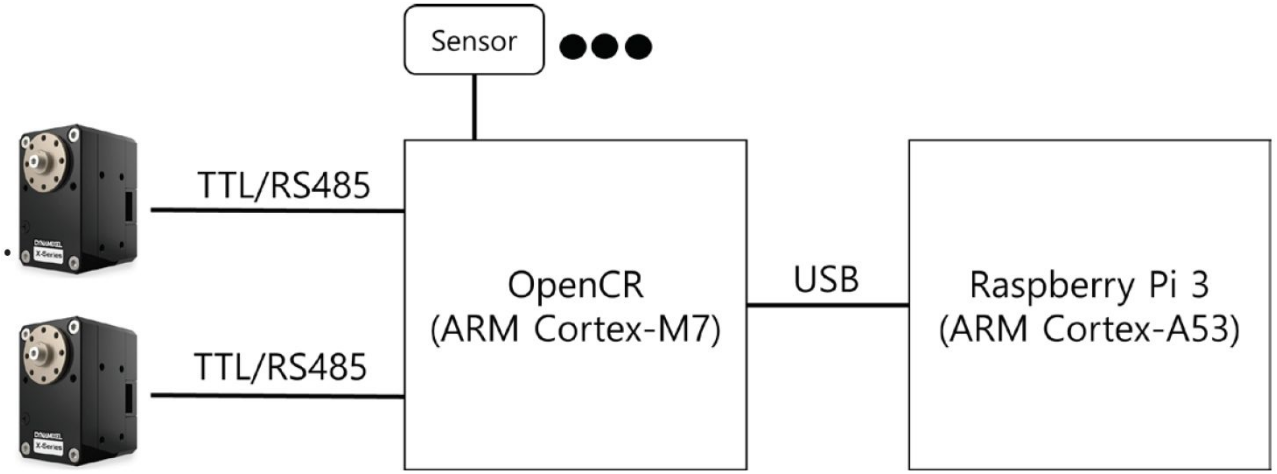


Dynamixel Actuator

- 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.

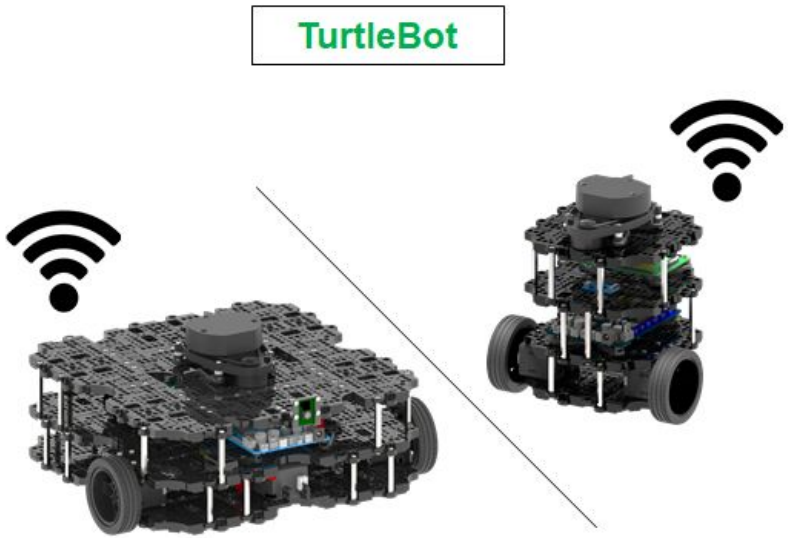
Turtlebot3 Embedded System

- 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



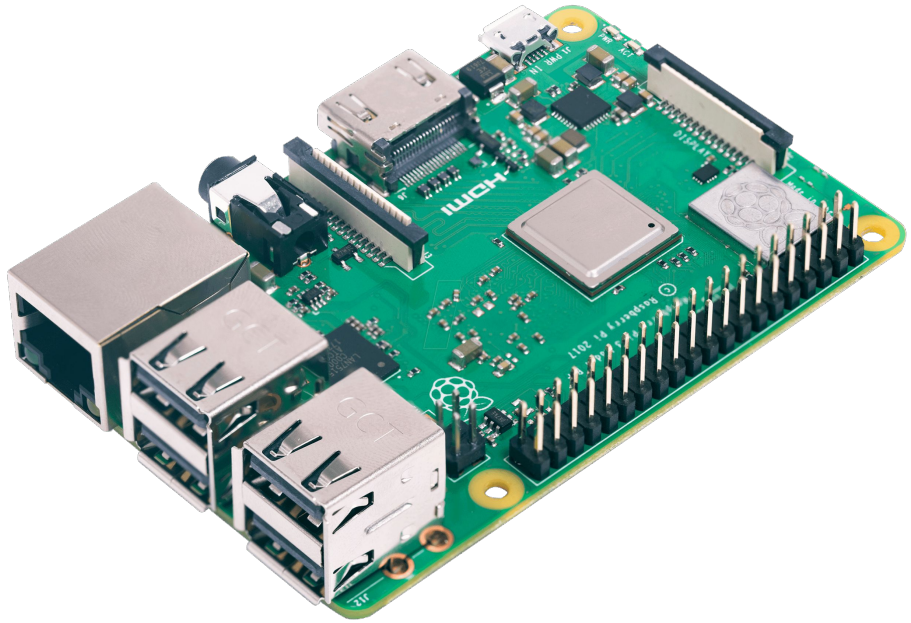
ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_TURTLEBOT

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)
```

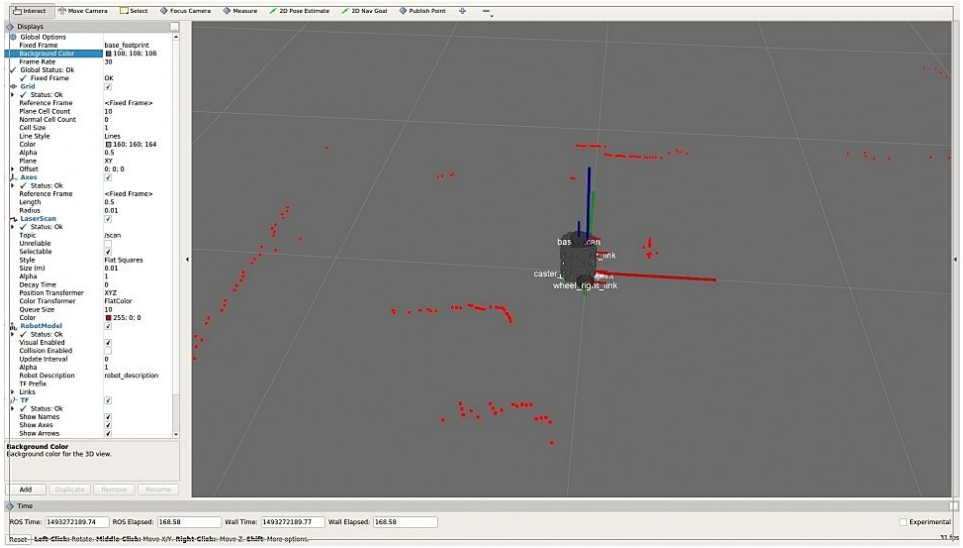
```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# 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
```

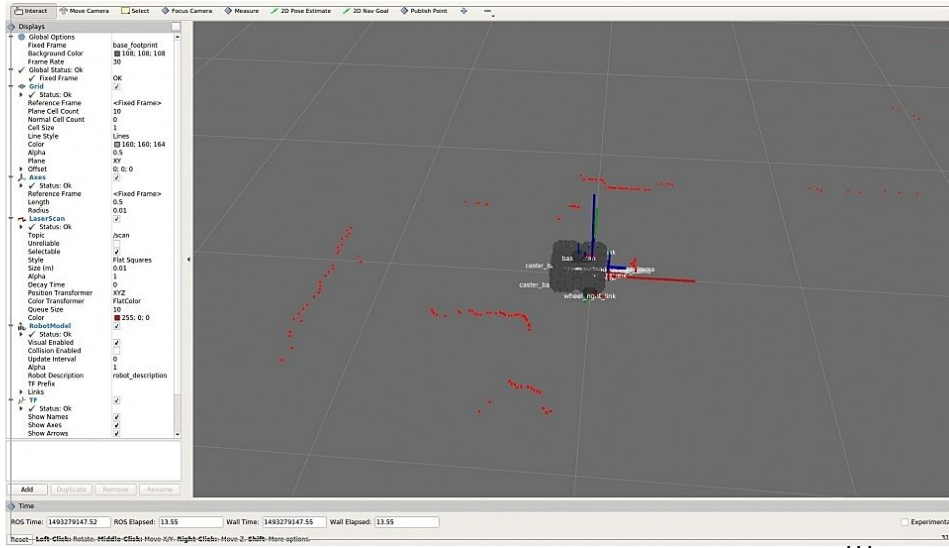
Bring Up Turtlebot3

[terminal 1]
$ roscore

[terminal 2]
$ ssh pi@[RASPBERRY PI_IP] eg ssh pi@192.168.1.127
$ password: turtlebot
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

[terminal 3]
$ rosrun rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz

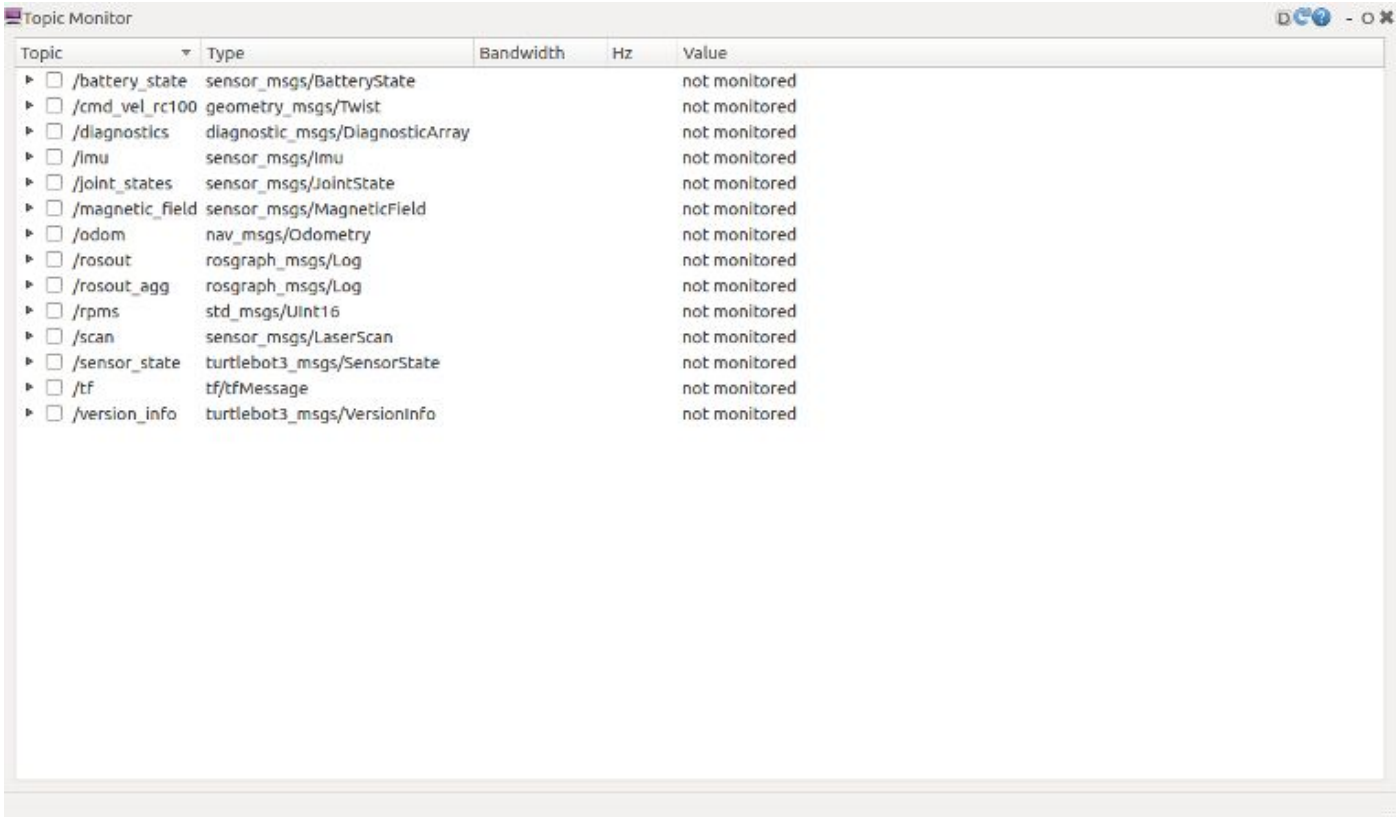




Topic Monitoring

- 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

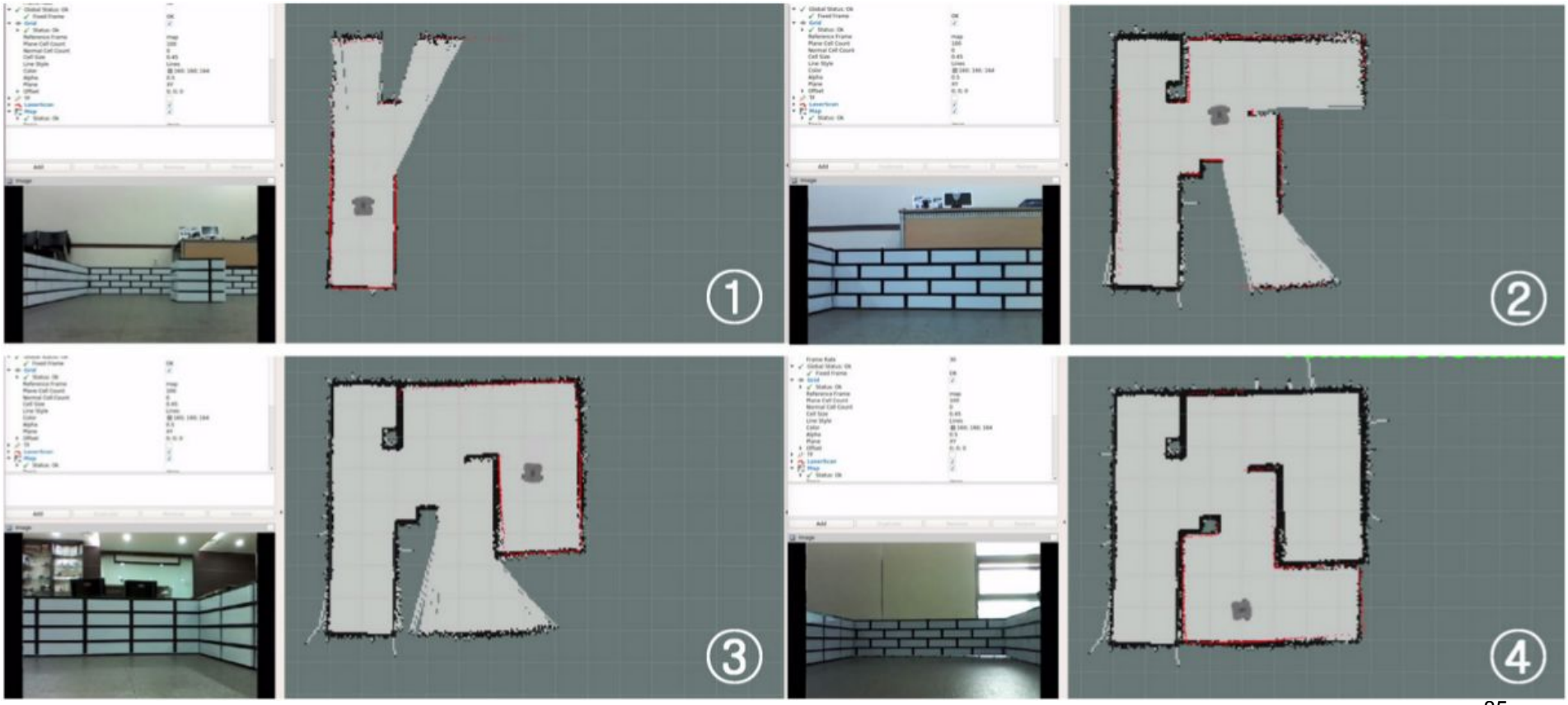
- The TurtleBot3 can be moved or stopped by LDS data. When the TurtleBot3 moves, it stops when it detects an obstacle ahead.

$ 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.

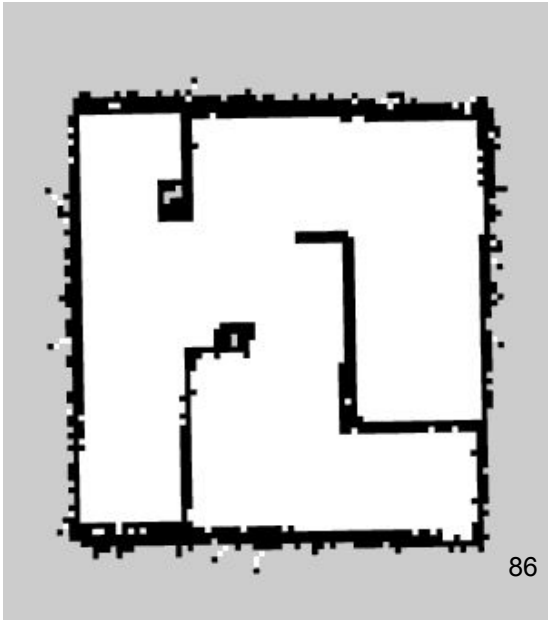
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping



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/classroom.yaml

Topic 4
ROS Navigation with Python

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 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 demo1 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 demo1 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

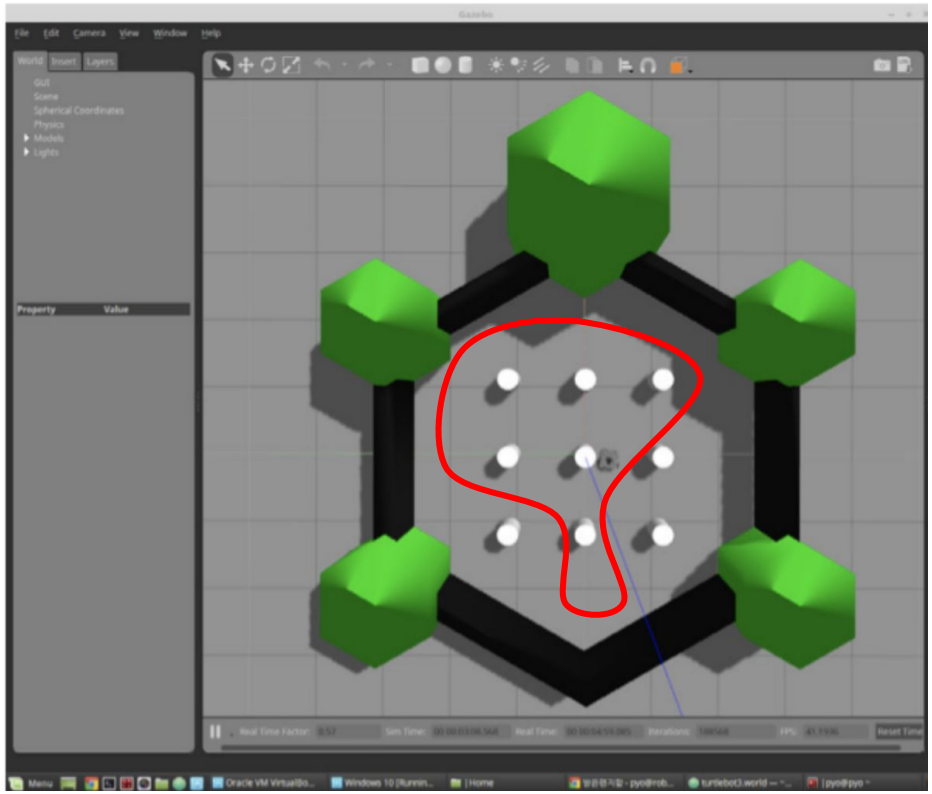
Activity: Avoid Obstacle

(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
$ roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
(terminal 3)
$ python demo1 avoid_obstacle.py

Activity: Path Planning

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

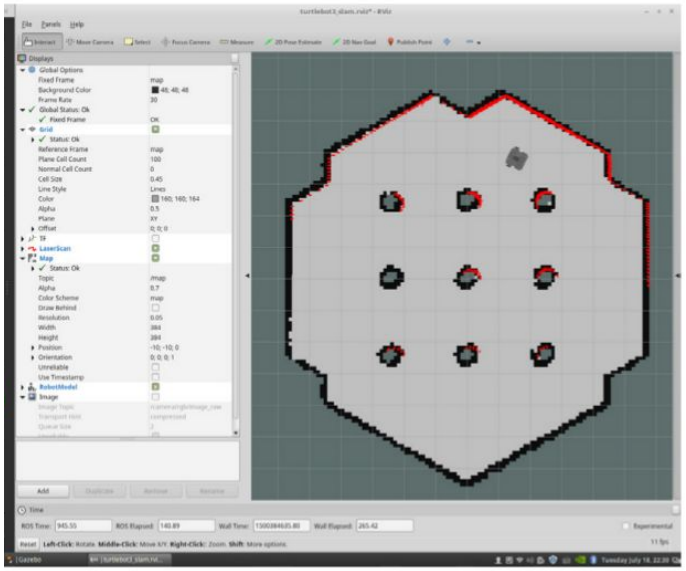
(terminal 1)
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
(terminal 2)
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/my_map.yaml
(terminal 3)
$ rosrun demo1 initial_pose.py
$ rosrun demo1 path_planning.py



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 demo1 autonnomous_exploring.py
(terminal 4) rosrun map_server map_saver -f ~/my_map2



Topic 5
ROS Navigation IoT

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

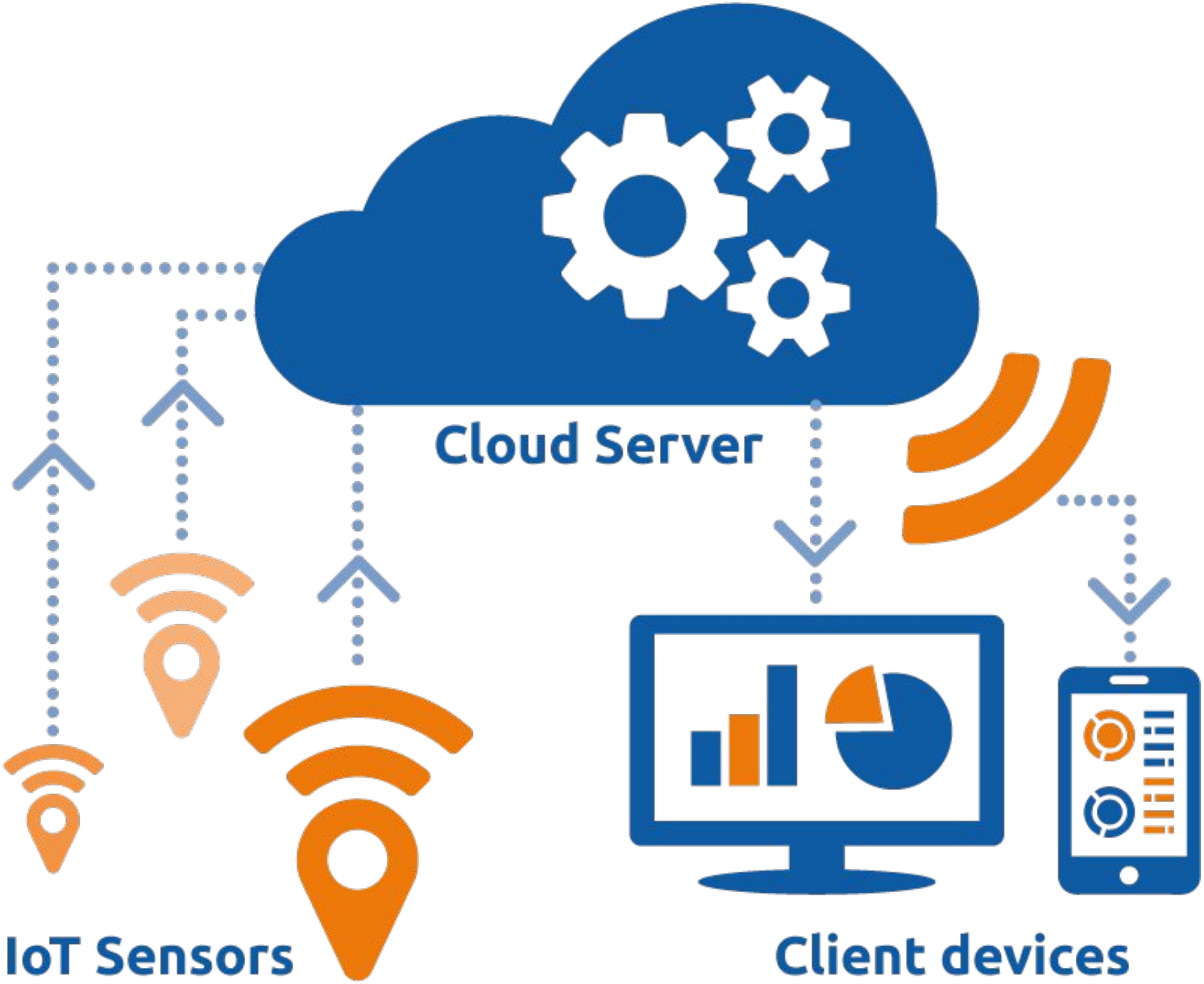
3rd Industrial Revolution:

- Electronics, computers
- Automation
- Information technology

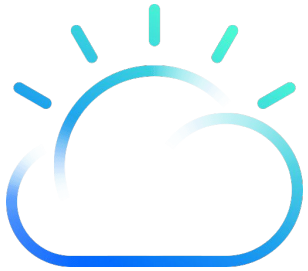
4th Industrial Revolution:

- 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

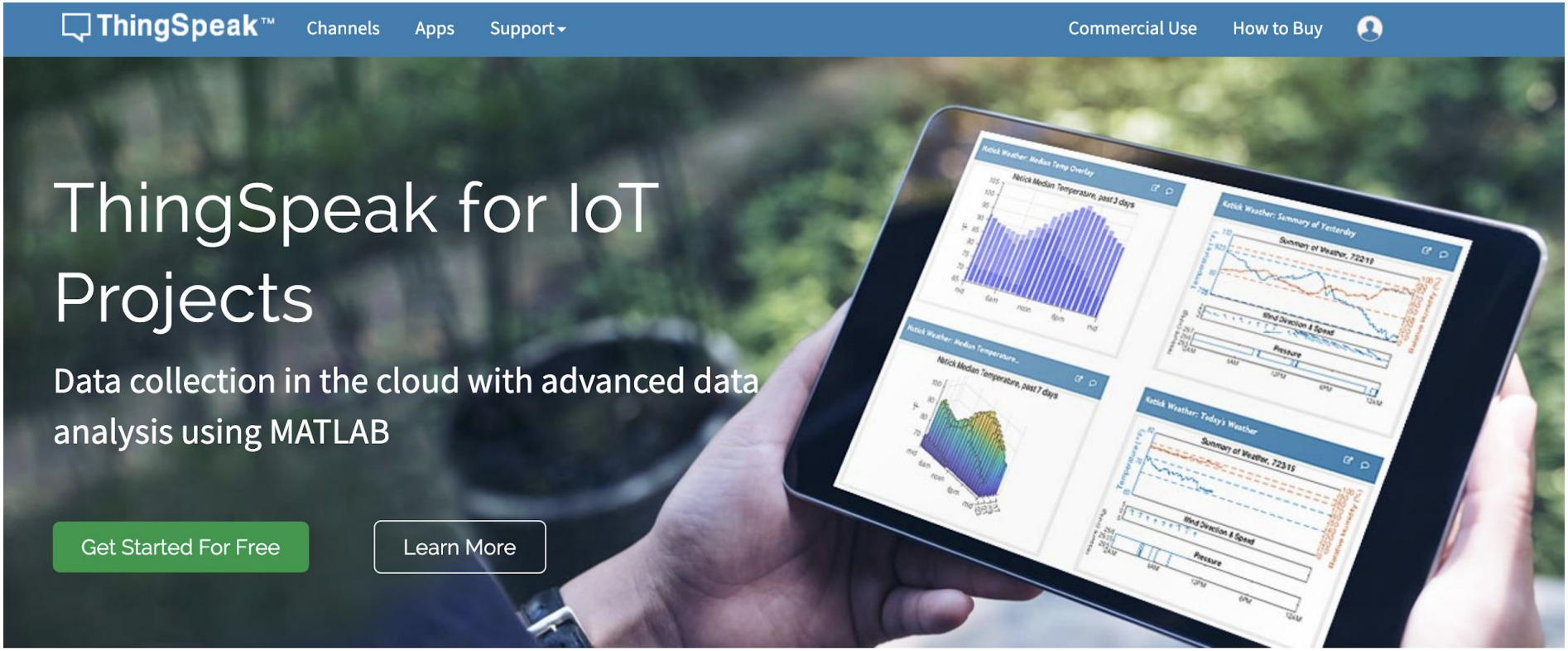


Google Cloud

Alibaba Cloud

thingworx®

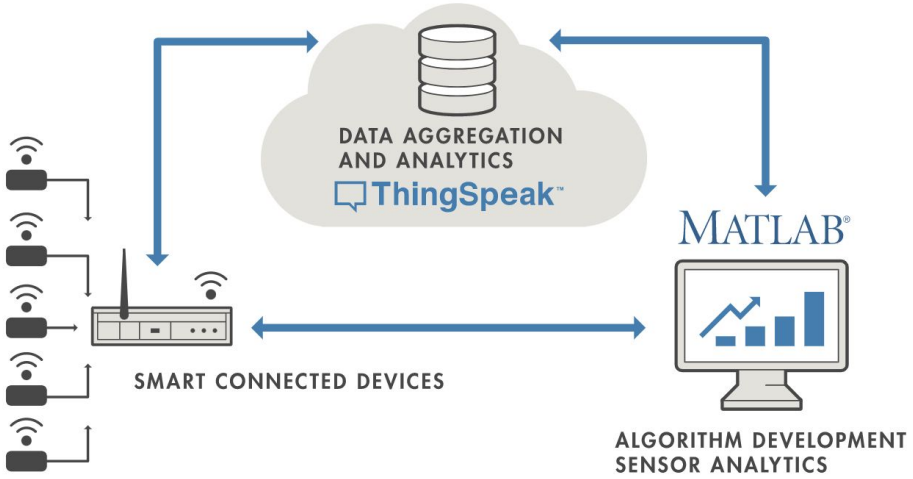
ThingSpeak



- 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

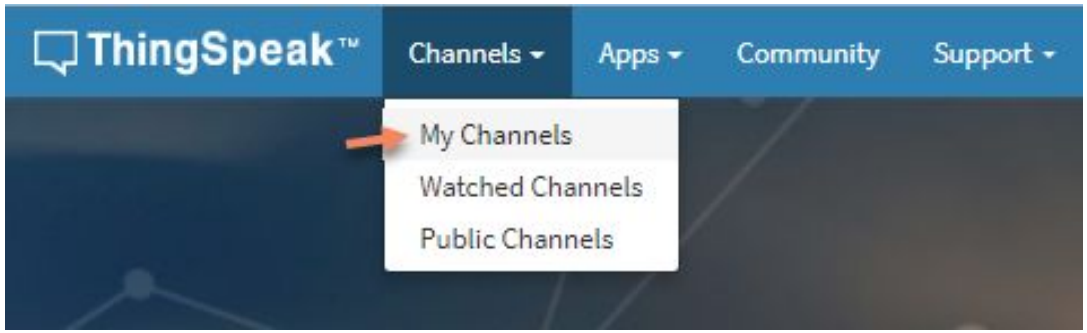
ThingSpeak Features

- 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/2.7/get-pip.py" -o "get-pip.py"
$ python get-pip.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

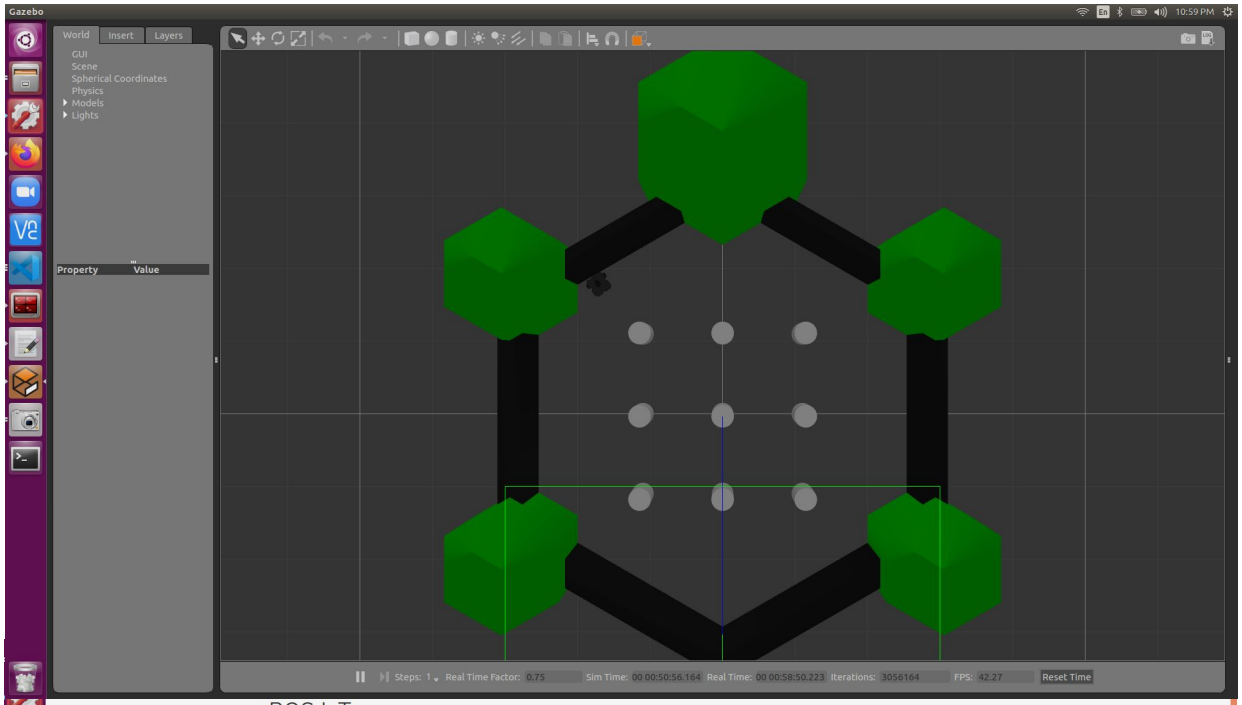
$ rosrun my_robotics iot.py

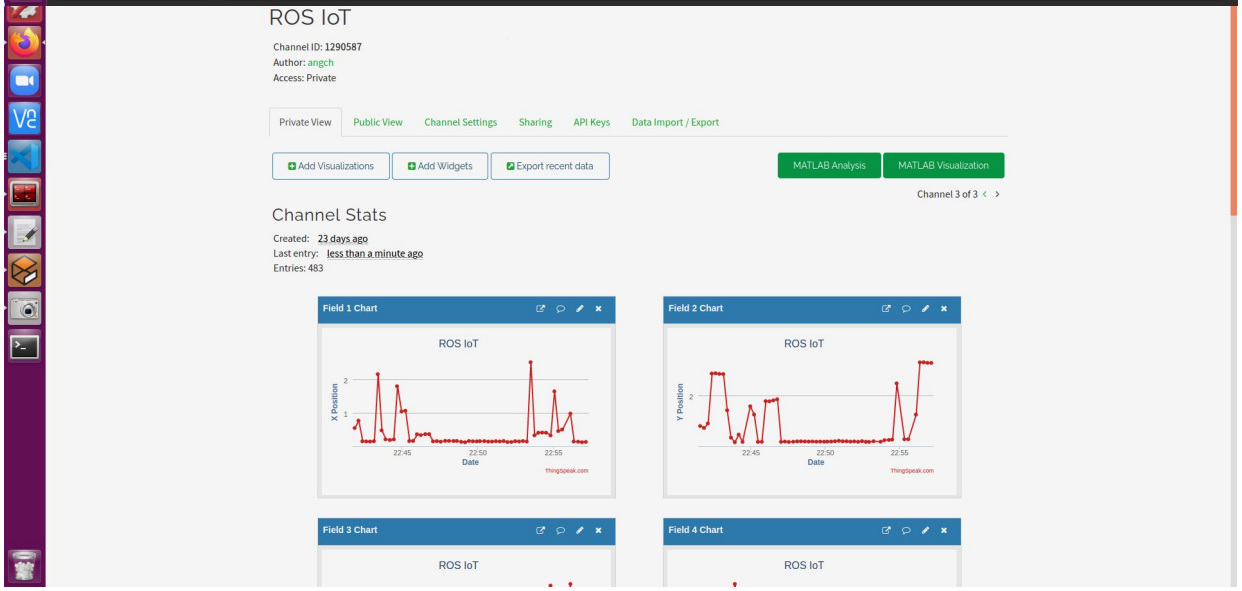
- Check if your ThingSpeak Channel receives the data from your running node

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

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

ROS IoT on ThingSpeak





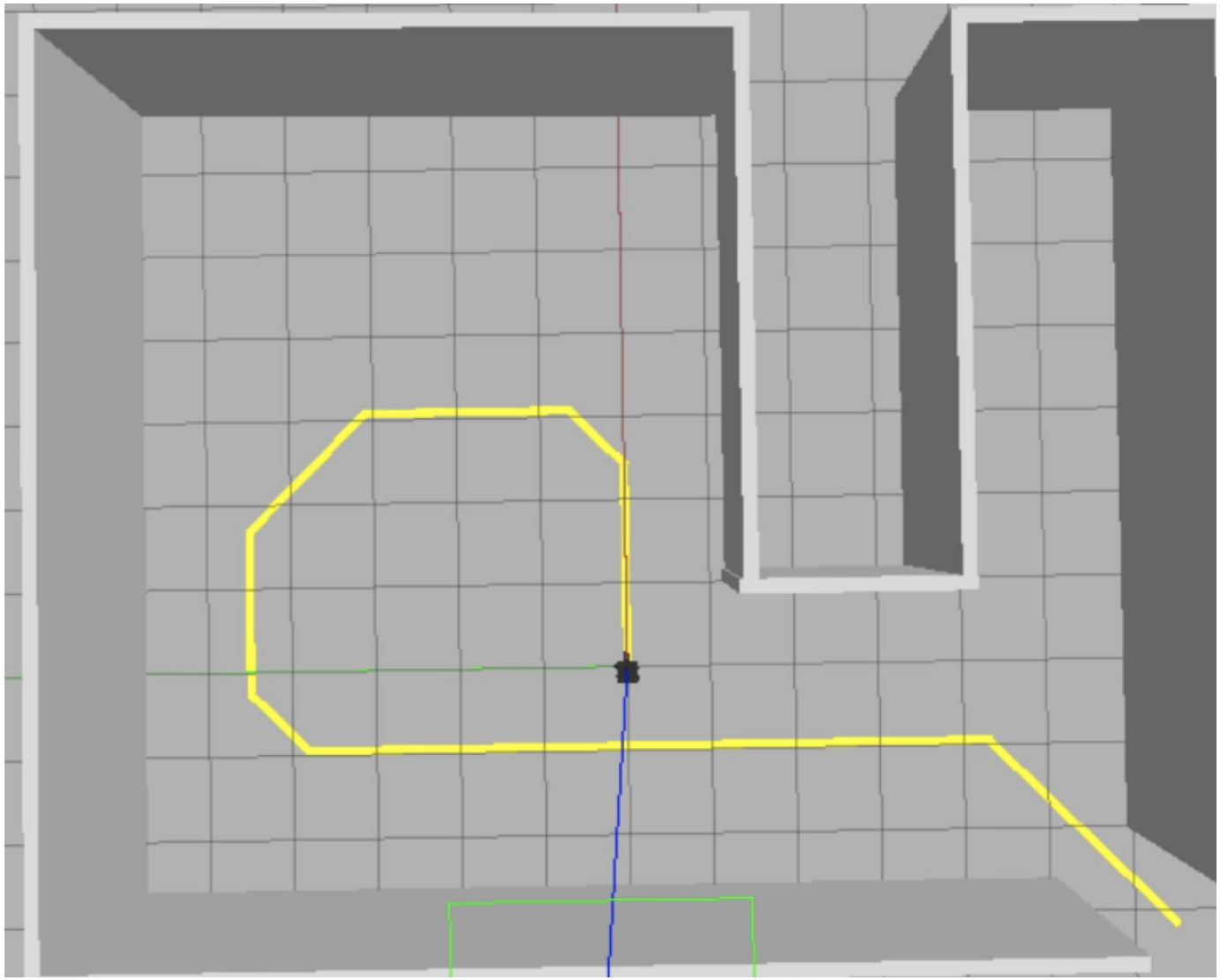
Challenge: ROS IoT

- Modify the iot1.py code to allow the turtlebot to move by itself and avoid the obstacles
- Send the odometry data to the ThingSpeak

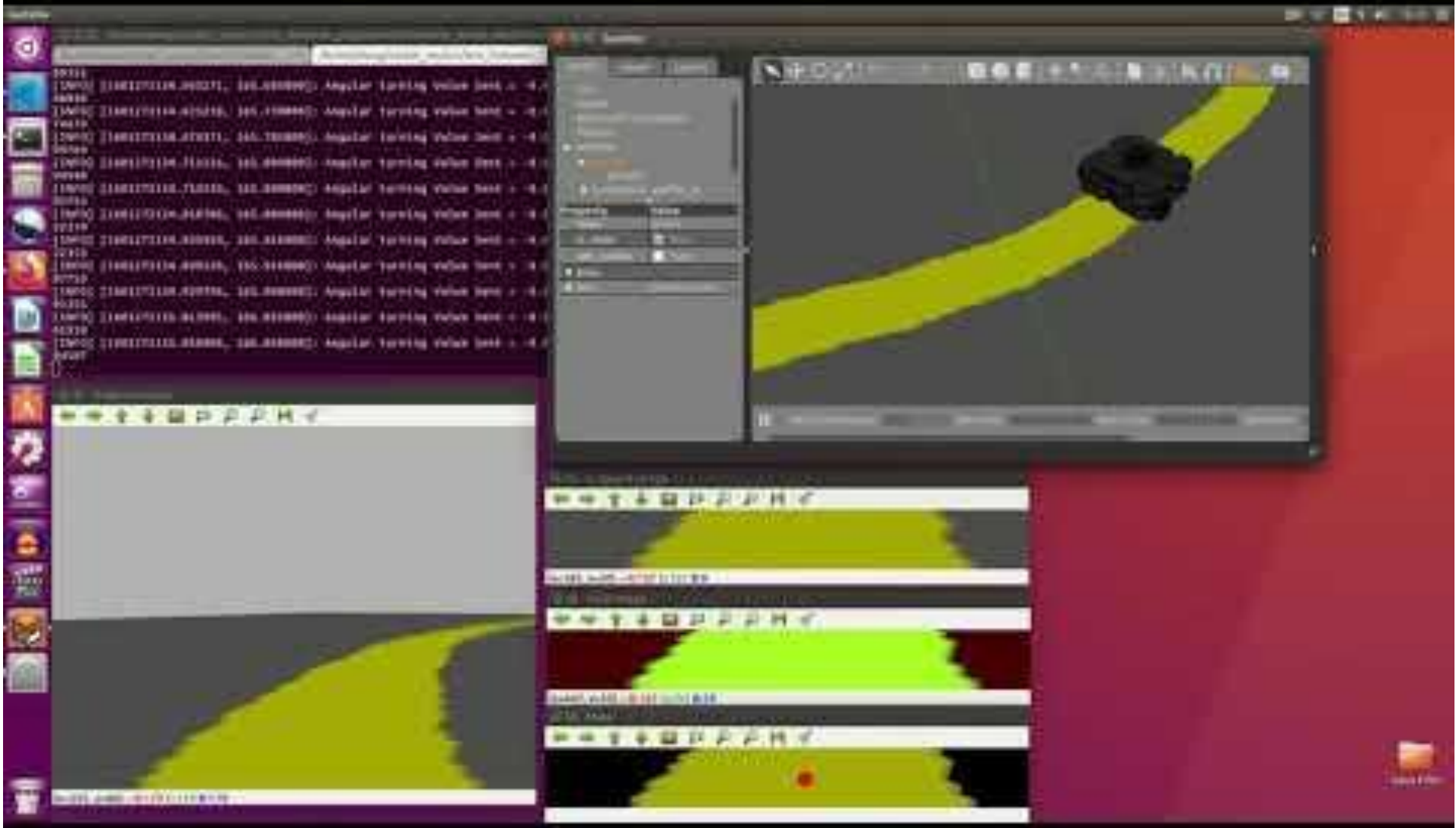
Topic 6
ROS Navigation with Vision

Line Following

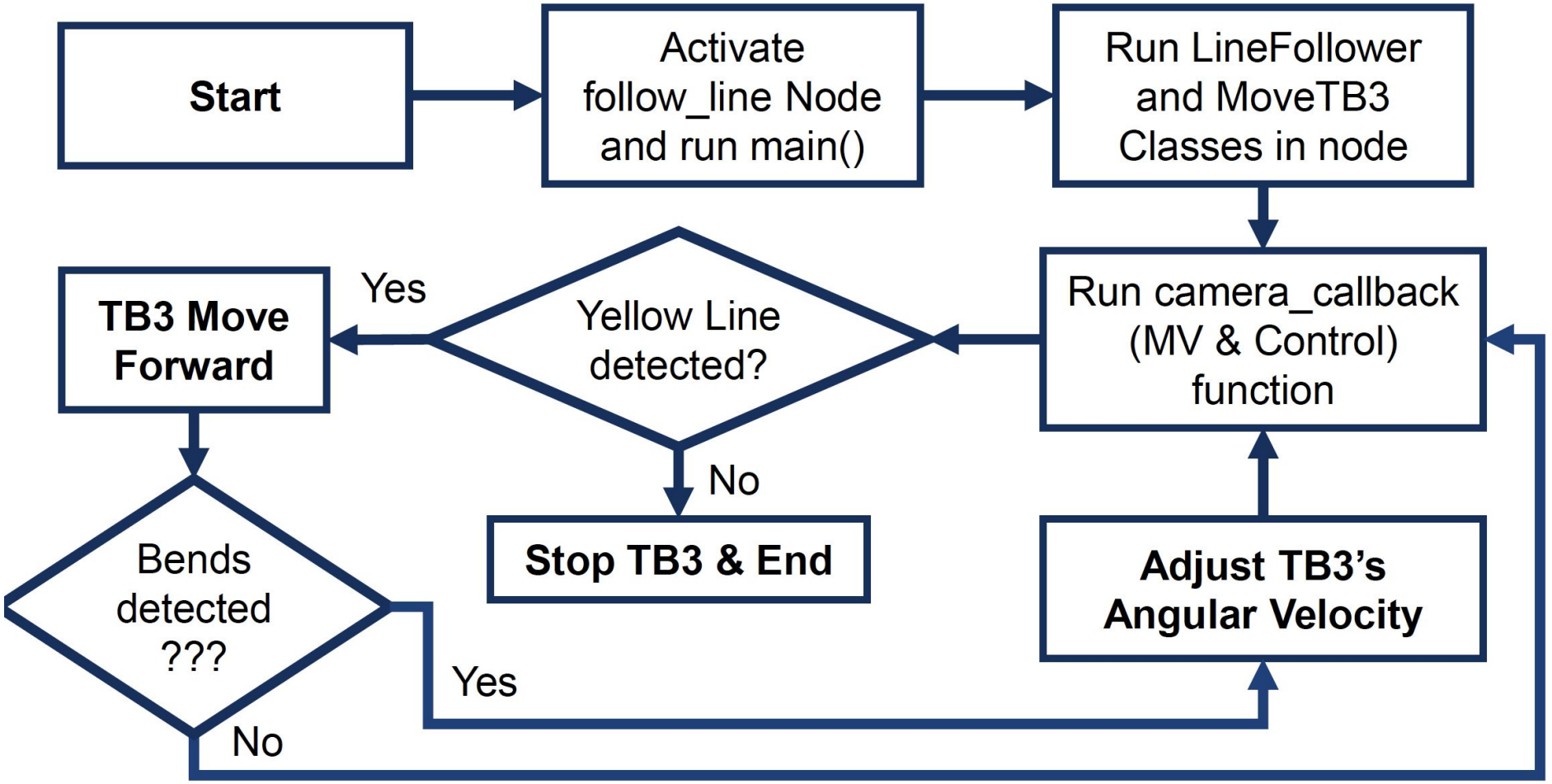
Get the Waffle Pi with OM to track and follow the yellow line.



Line Following



Line Following Algorithm



Install OpenCV Packages

$ sudo apt-get install ros-kinetic-image-transport ros-kinetic-cv-bridge ros-kinetic-vision-opencv python-opencv libopencv-dev ros-kinetic-image-proc

Setup Webcam in Virtualbox

Refer to following website on instructions:
https://scribles.net/using-webcam-in-virtualbox-guest-os-on-windows-host/

1 From the VirtualBox Download page (i.e. http://download.virtualbox.org/virtualbox/), download the extension pack which has the same version as your VirtualBox. In my case, my VirtualBox is v6.1.4 so I downloaded this: http://download.virtualbox.org/virtualbox/6.1.4/Oracle_VM_VirtualBox_Extension_Pack-6.1.4.vbox-extpack

2 Launch "Oracle VirtualBox Manager" and navigate to "File" -> "Preferences".
3 In 'Preferences' window, select 'Extensions'.
4 Press 'Add new package' icon.
5 Navigate and select the downloaded extension pack and install it.
6 On WINDOWS Command Prompt (search command prompt in search bar in your WINDOWS platform), type the following:
cd c:\Program Files\Oracle\VirtualBox

Setup Webcam in Virtualbox

7 Next, type the following to list the available cameras
VBoxManage list webcams
Take note of the camera that you desire to use (e.g. .1 or .2, etc)

8 Next, type the following to attach webcam(s) you want to use on your Virtual Machine.
VboxManage controlvm "Ubuntu 16.04" webcam attach .X
Change "X" to the camera no. that you are using!
*Note that you have to attach the webcam(s) each time you boot your Ubuntu system on the virtual machine

9 You may check if the procedures are done properly by searching on Ubuntu for Cheese Webcam Booth application; open it and see if your webcam can be used

Activity: Line Following

1. Navigate to Home >> .gazebo (type clt-H to review hidden files) >> models

2. Copy and paste all files under “models” in unzipped file to this folder

4. Copy “line.launch” to the launch folder

5. Copy “follow_line.py” and “move_robot_turtlebot3.py” files to the src folder

6. Copy “line.world” file to src >> turtlebot3_simulations >> turtlebot3_gazebo >> worlds

7. Load TB3 with OM in line.world in Gazebo
$ roslaunch my_robotics line.launch

8. Run node example to utilize OpenCV to detect the yellow line and make the TB3 follow it
$ rosrun my_robotics follow_line.py

Gesture Control

Gesture Control Navigation

- Launch Gazebo in TurtleBot3 World

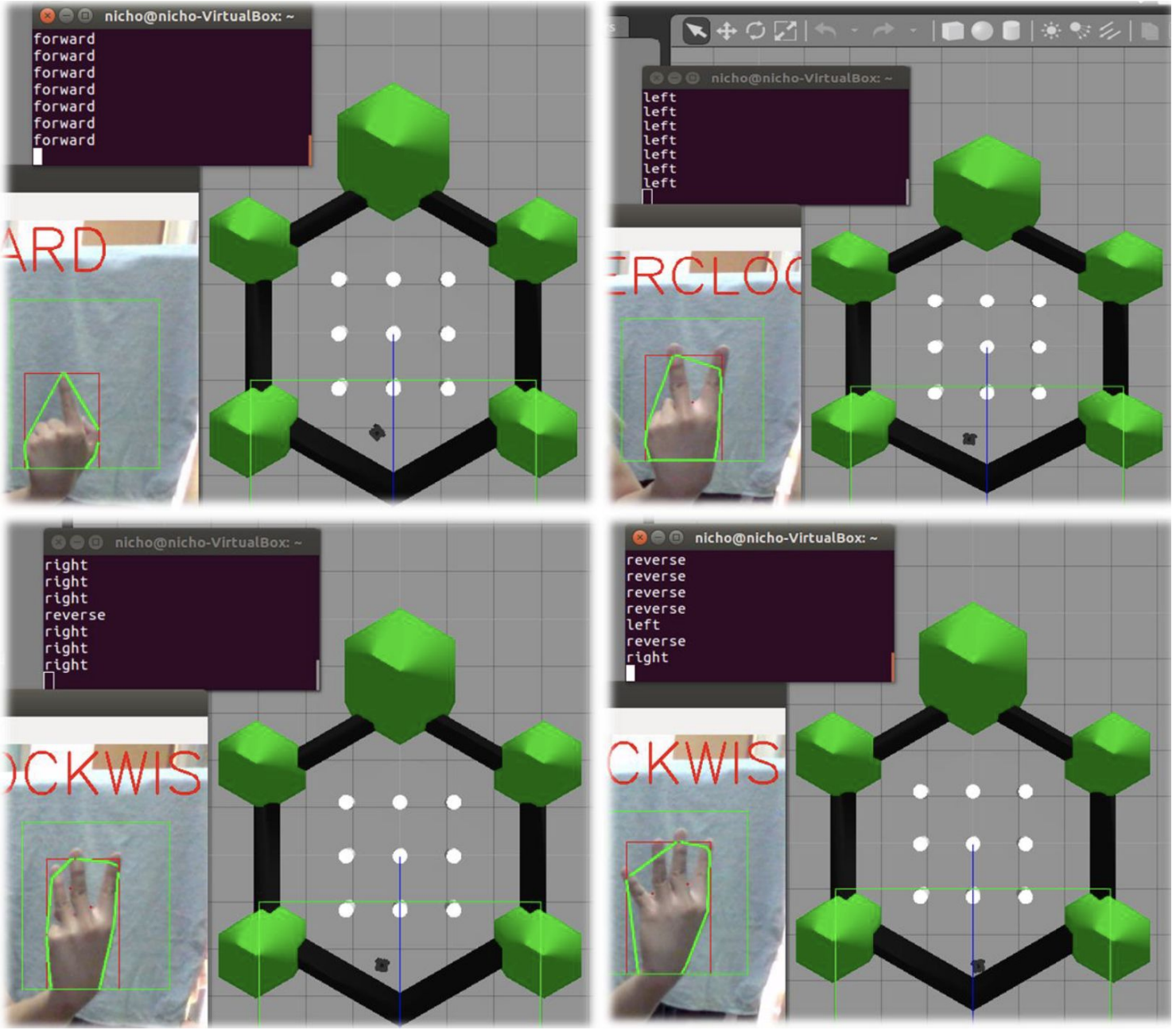
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch

- Run node that allows you to use finger gestures to control the TB3

$ rosrun my_robotics gesture_control.py

- Adjust camera until the bounding box is within blue background
- Try the gesture control by changing your finger gestures into 1, 2, 3 or 4 as illustrated in the demo or pictures; 1, 2, 3, 4 are represented by the following controls:
  - 1: Forward
  - 2: Left (Counterclockwise)
  - 3: Right (Clockwise)
  - 4: Backwards
  - No Input: Brake

Gesture Control Navigation



Activity: Gesture Control Navigation

- Modify the iot1.py code to allow the turtlebot to follow the hand gesture and avoid the obstacles

Summary Q&A

Feedback

https://goo.gl/R2eumq

TINFOTECH

CERTIFICATE

of ACCOMPLISHMENT

You will receive a digital certificate in your email
after the completion of the class
If you did not receive the digital certificate,
please send your request to
enquiry@tertiaryinfotech.com

Thank You!

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