



TURTLEBOT3

TurtleBot3

Burger



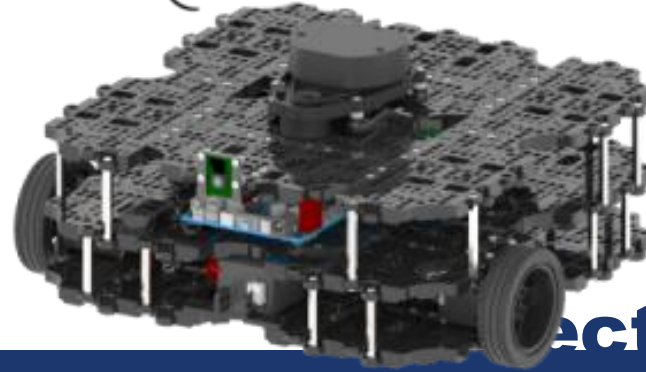
TurtleBot3

Waffle



TurtleBot3

Waffle Pi



GAZEBO

Lecture 6

2022

INFO 802

Master Advanced Mechatronics

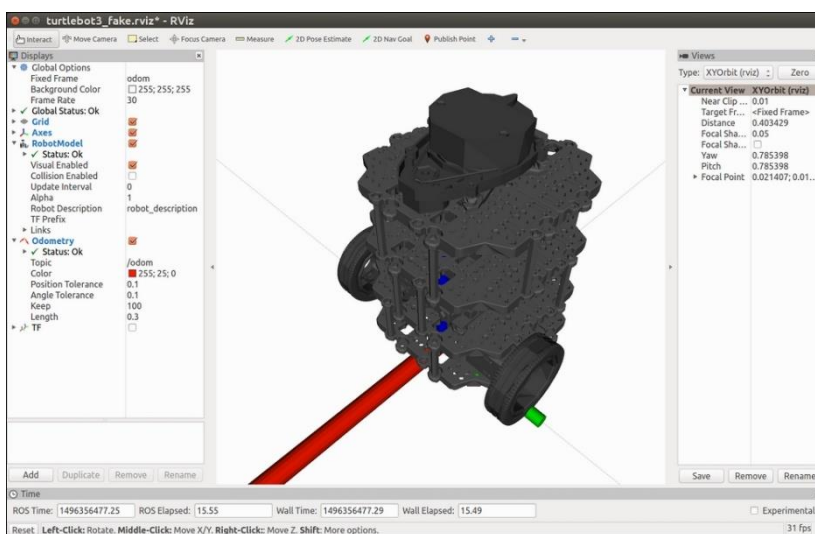
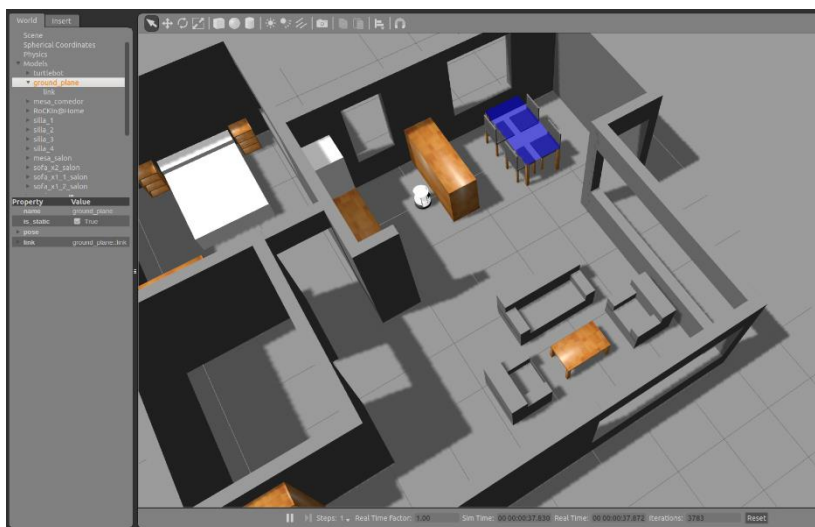
Luc Marechal



ROS

Gazebo Turtlebot3

Robotic simulation scenarios



More Info and tutorials
<http://gazebosim.org/tutorials>
<http://wiki.ros.org/rviz/Tutorials>

Gazebo – TurtleBot3 Simulation

- A multi-robot simulator
- Capable of simulating a population of robots, sensors and objects, in 3D
- Includes an accurate simulation of rigid-body physics and generates realistic sensor feedback
- Allows code designed to operate a physical robot to be executed in an artificial environment
- Gazebo is under active development at the OSRF (Open Source Robotics Foundation)



More Info and tutorials
<http://gazebosim.org/tutorials>

Exercise

Gazebo and TurtleBot3

- Read docs on Gazebo (and Rviz as well if you want to have a larger view of ROS capabilities).
- Launch Gazebo with the `turtlebot_world` environment (the `turtlebot_house` might be too heavy for your computer but you can try) and:
 - the Waffle robot
 - the Burger robot
- What are the differences for these robots?
- Are they using the same topics on Gazebo? Is there any differences?
- Move manually the TurtleBot3 using `teleop_key`
- Show the `rqt_graph` to see the topics and nodes

Gazebo – TurtleBot3 Simulation

Launch Gazebo

```
> roslaunch turtlebot3_gazebo turtlebot3_autorace.launch
```

(also try : turtlebot3_house.launch)

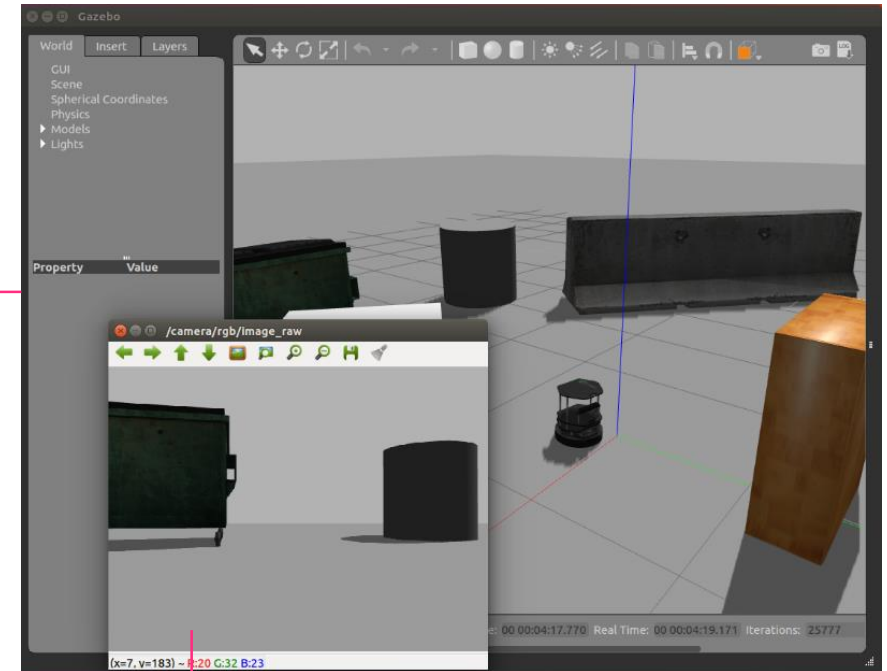
Display the image from the robot

```
> rostopic list # check where the image is published
```

```
> rosrun image_view image_view image:=/camera/image
```

Move the robot with keyboard

```
> roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```



```
Control Your Turtlebot!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

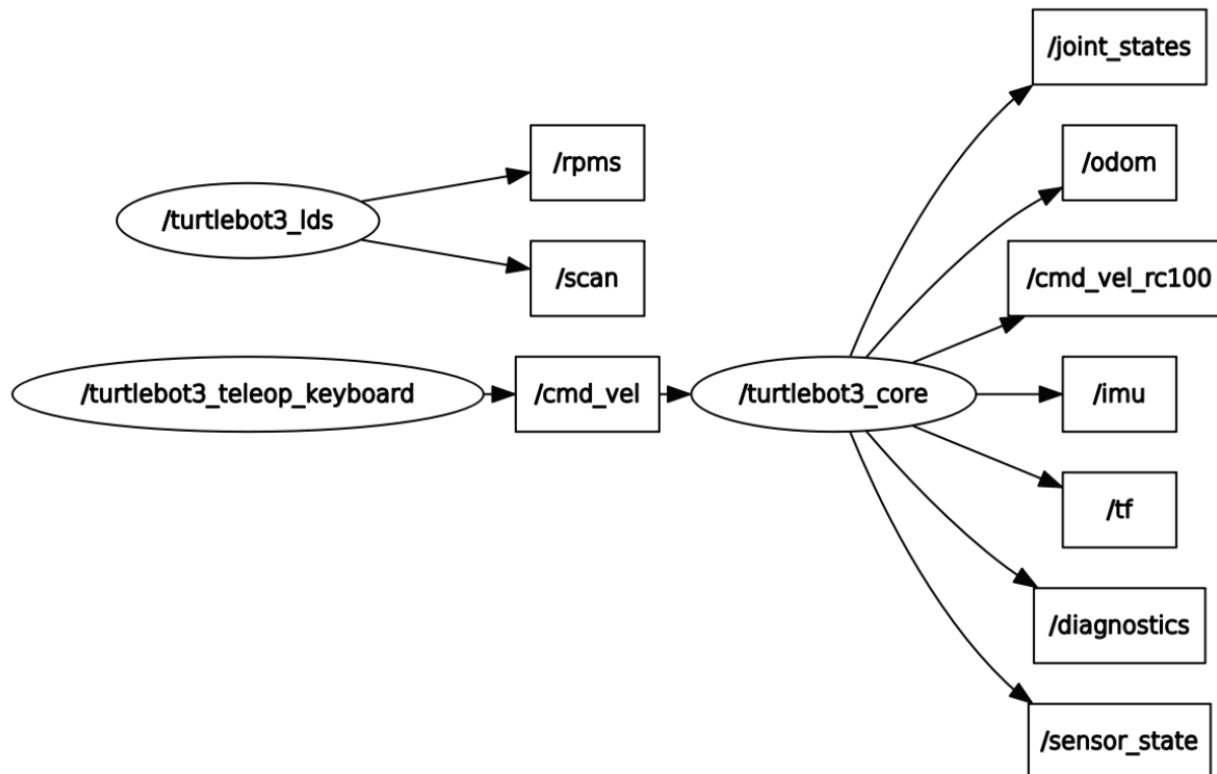
CTRL-C to quit

currently:    speed 0.2      turn 1
```

Gazebo – TurtleBot3 Simulation

Node and topic

```
> rqt_graph
```



```
Luc@USMB:~$ rostopic list
```

```
/camera/camera_info  
/camera/image  
/camera/image/compressed  
/camera/image/compressed/parameter_descriptions  
/camera/image/compressed/parameter_updates  
/camera/image/compressedDepth  
/camera/image/compressedDepth/parameter_descriptions  
/camera/image/compressedDepth/parameter_updates  
/camera/image/theora  
/camera/image/theora/parameter_descriptions  
/camera/image/theora/parameter_updates  
/camera/parameter_descriptions  
/camera/parameter_updates  
/clock  
/cmd_vel  
/gazebo/link_states  
/gazebo/model_states  
/gazebo/parameter_descriptions  
/gazebo/parameter_updates  
/gazebo/set_link_state  
/gazebo/set_model_state  
/imu  
/joint_states  
/odom  
/rosout  
/rosout_agg  
/scan  
/tf
```

Gazebo – TurtleBot3 Simulation

Odometry

- To make a TurtleBot move in ROS we need to **publish**:

Twist messages to the topic **/cmd_vel**

- To get a TurtleBot position and orientation in ROS we need to **subscribe**:

to the topic **/odom** and read **Odometry** message

- Odometry is part of *nav_msg* message package
(don't forget to add `import nav_msg.msg` in your code header)

```
> rosmmsg show Odometry
```

```
luc@USMB:~$ rosmmsg show Odometry
[nav_msgs/Odometry]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
  string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
      float64[36] covariance
  geometry_msgs/TwistWithCovariance twist
    geometry_msgs/Twist twist
      geometry_msgs/Vector3 linear
        float64 x
        float64 y
        float64 z
      geometry_msgs/Vector3 angular
        float64 x
        float64 y
        float64 z
      float64[36] covariance
```

Exercise

Gazebo and TurtleBot3

- Move TurtleBot3 Waffle using publisher node
 - Create your own package named `Tutorial_TurtleBot`
(Recall: New packages must be created in the `src` folder from `catkin_ws`)
 - Create your own Python script for moving TurtleBot3 with two arguments Linear velocity and Angular Velocity
- Getting laser data of the Waffle using ROS commands and Python script
 - Create a new node to subscribe to the topic `scan` and get the information from the laser sensor.
 - Named it `get_laser_data.py`
 - We want to get the value of the scanner in front of the robot ----> `msg.range[0]`

LaserScan

<https://youtu.be/tEayzulupxE>

<https://youtu.be/kze3Z8rTkZo>

move_turtlebot (Python)

move_turtlebot.py

```
#!/usr/bin/python3

import rospy
from geometry_msgs.msg import Twist
import sys

def move_turtlebot(lin_vel, ang_vel):
    rospy.init_node('move_turtlebot', anonymous=False)
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    rate = rospy.Rate(1) # 1hz
    vel = Twist()

    while not rospy.is_shutdown():
        vel.linear.x = lin_vel
        vel.angular.z = ang_vel
        pub.publish(vel)
        rate.sleep()

if __name__ == '__main__':
    try:
        move_turtlebot(float(sys.argv[1]), float(sys.argv[2]))
    except rospy.ROSInterruptException:
        pass
```



LaserScan Message

Getting laser data

- Laser data is published on the topic *scan*. Therefore, to access this data we have to subscribe to this topic, obtain the required data and use it for our desired application.
- Obtain information about the topic (in a separate window):

```
$ rostopic list  
$ rostopic info scan  
$ rosmmsg show LaserScan  
$ rostopic echo scan
```

LaserScan Message

- http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

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

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

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

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

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

LaserScan Message

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

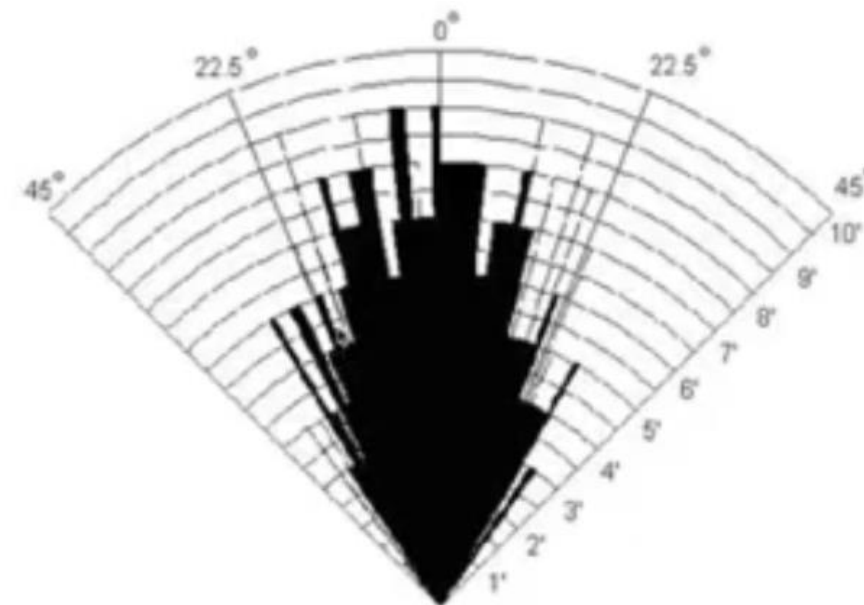
table of float values

(distance in meter to an object)

ranges = [inf, 50.3, 2.1,...,2.6, 0.4,]

example : to get the 3rd value of the table In: ranges[3]

Out: 2.1



To retrieve the range to the nearest obstacle directly in front of the robot, we will select the middle element of the ranges array:

```
range_ahead = msg.ranges[len(msg.ranges)/2]
```

Or, to return the range of the closest obstacle detected by the scanner:

```
closest_range = min(msg.ranges)
```

Exercise

LaserScan

- Create a new node `laser_data.py` to subscribe to the topic `scan` and get the information from the Lidar sensor

```
#!/usr/bin/python3
import rospy
from sensor_msgs.msg import LaserScan
import sys

def scan_callback(msg):
    front_distance = int(len(msg.ranges)/2)
    range_ahead = msg.ranges[front_distance]
    rospy.loginfo("range ahead = %0.2f\n",range_ahead)

def read_laser():
    rospy.init_node('Turtlebot3_Read_Laser', anonymous=False)
    rospy.Subscriber('scan', LaserScan, scan_callback)
    rospy.spin()

if __name__ == '__main__':
    read_laser()
```

Exercise

Gazebo and TurtleBot3

- Make robot avoid obstacles in front of him
 - Make the robot to stop when an obstacle in front of the robot is closer than 0.5 m

- Hints:
 - Create a node which is a publisher and subscriber at the same time.
 - The node should subscribe to the topic scan and publish on the topic cmd_vel
 - Use the code implemented in the previous scripts and put everything together.
 - Use conditionals to make the robot behave as you want

Exercise

Gazebo and TurtleBot3

- You will find help in this Book in the Chapter 7 Wander-bot

