Lecture 4

2021

INFO 802

Master Advanced Mechatronics

Luc Marechal

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

ROS

Robot control Turtlesim

Course 4 : Robot control

Outline

- Turtlesim topic, messages and commands
- Move Turtle
- Gazebo TurtleBot simulation

Course 4 : Robot control

Objectives

- Know which topics are at stake in a node
- Know what type of message is at stake and what is the source package
- Know how to use Twist, Pose, Odometry messages
- Write a publisher function
- Write a subscriber function and understand its callback

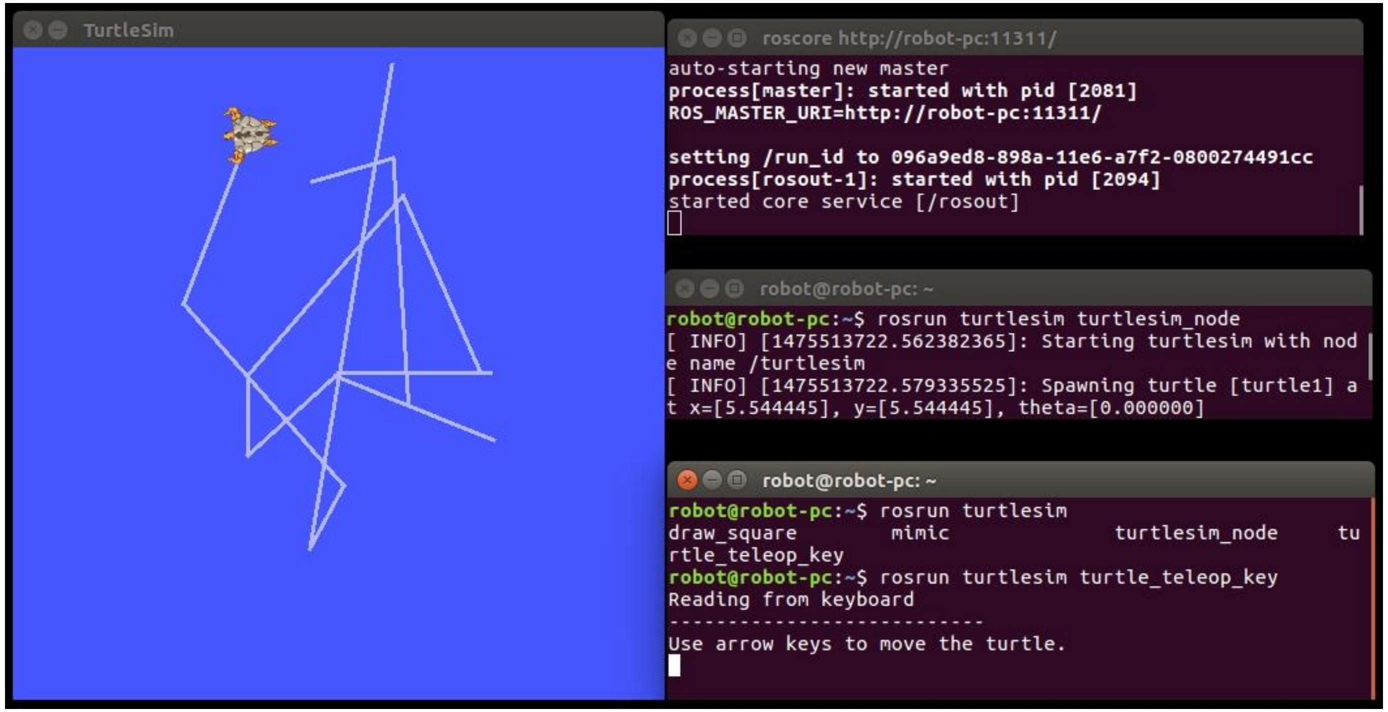
Turtlesim

Recall: Open a terminal for each command

```
> roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```



```
auto-starting new master
process[master]: started with pid [2081]
ROS_MASTER_URI=http://robot-pc:11311/

setting /run_id to 096a9ed8-898a-11e6-a7f2-0800274491cc
process[rosout-1]: started with pid [2094]
started core service [/rosout]
```

```
robot@robot-pc:~$ rosrun turtlesim turtlesim_node
[ INFO] [1475513722.562382365]: Starting turtlesim with nod
e name /turtlesim
[ INFO] [1475513722.579335525]: Spawning turtle [turtle1] a
t x=[5.544445], y=[5.544445], theta=[0.000000]
```

```
robot@robot-pc:~$ rosrun turtlesim
draw_square        mimic              turtlesim_node     tu
rtle_teleop_key
robot@robot-pc:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

Turtlesim

- Questions to answer:

Which topic is the velocity command published to?

Which topic is the position information available from?

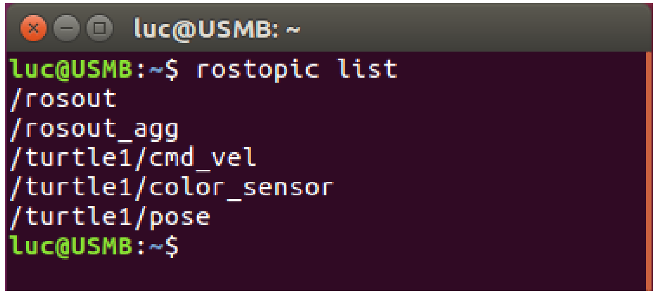
What kind of messages are used?

Which message packages are they from?

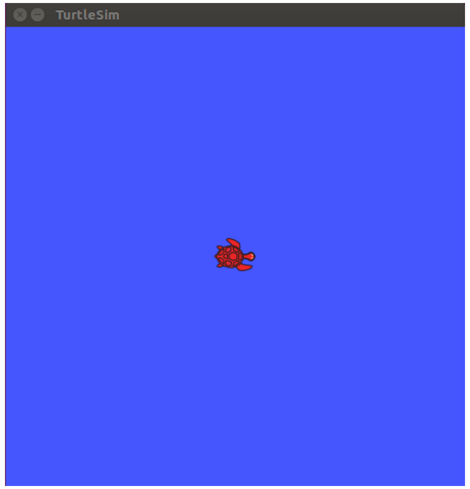
```
> rosrun turtlesim turtlesim_node
> rostopic list
> rostopic type [topic]
> rosnode info turtlesim_node
```

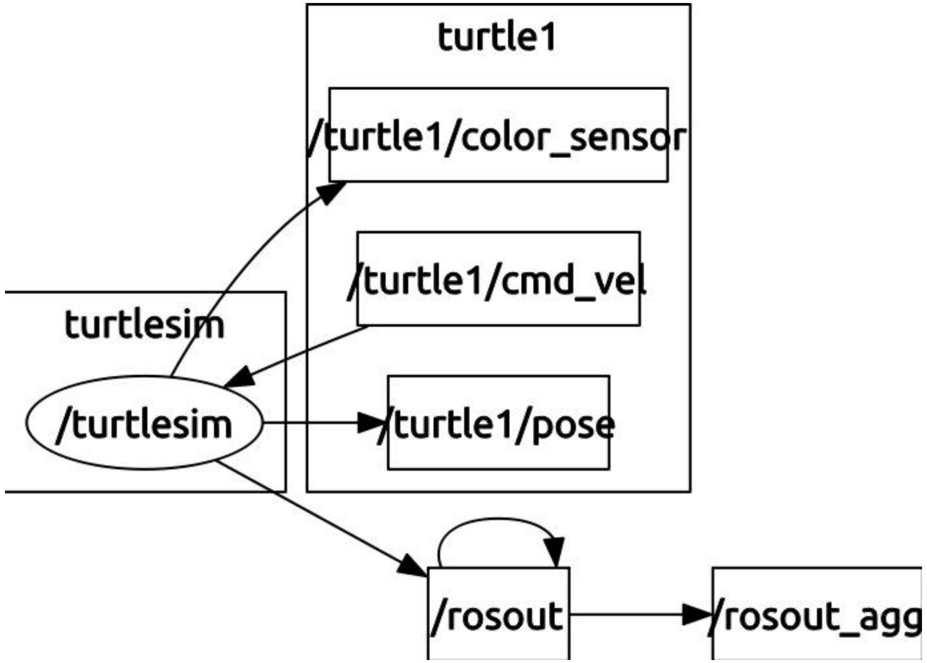
Visualize node and topic

```
> rqt_graph
```



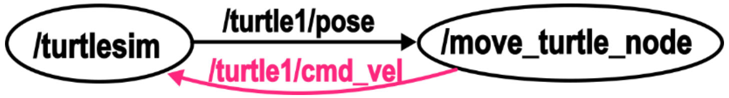
```
luc@USMB:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
luc@USMB:~$
```



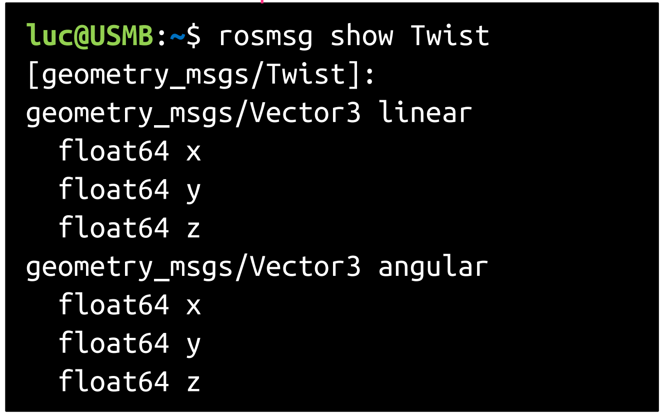


Turtlesim

Twist



```
> rostopic type /turtle1/cmd_vel
> rosmsg show Twist
```



```
luc@USMB:~$ rosmsg show Twist
[geometry_msgs/Twist]:
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

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

  Twist messages to the topic ***/turtle1/cmd_vel***

- This message has:

  - a linear component for the (x,y,z) velocities,

  - an angular component for the angular rate about the (x,y,z) axes

- Twist is part of *geometry_msgs* message package

  (don't forget to add *import geometry_msgs.msg* in your code header)

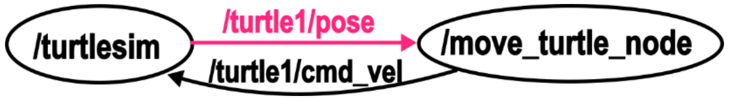
Example of use

create a *Twist* object
set the linear velocity along x
set the angular rate about z

```
vel = Twist()
vel.linear.x = 1.0
vel.angular.z = 0.4
```

Turtlesim

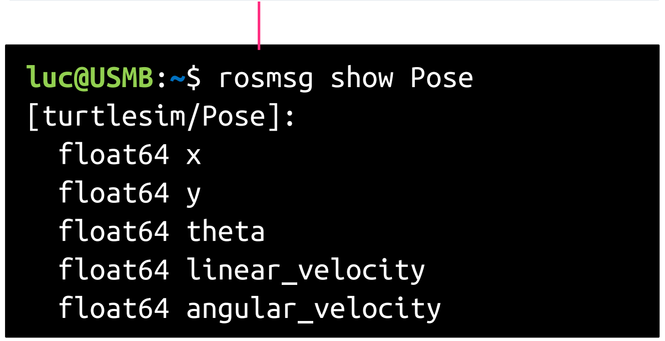
Pose



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

  to the topic ***/turtle1/Pose*** and read ***Pose*** message

```
> rostopic type /turtle1/Pose
> rosmsg show Pose
```



```
luc@USMB:~$ rosmsg show Pose
[turtlesim/Pose]:
  float64 x
  float64 y
  float64 theta
  float64 linear_velocity
  float64 angular_velocity
```

- This message has:

  - a linear component for the (x,y) 2D coordinates,

  - an angular component theta about the z axes

- Pose is among others part of *turtlesim* message package

  (don't forget to add *import turtlesim.msg* in your code header)

Example of use

create a *Pose* object — `pose = Pose()`

get the x position of turtle — `robot_x = pose.x`

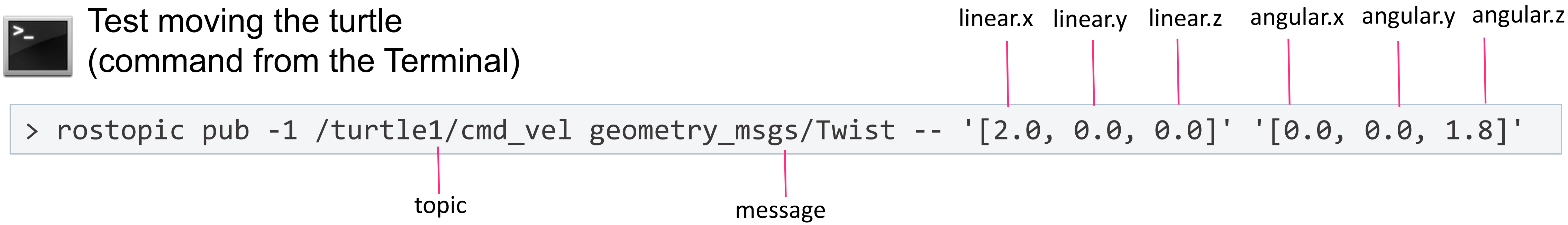
get the y position of turtle — `robot_y = pose.y`

```
pose = Pose()
robot_x = pose.x
robot_y = pose.y
```

Turtlesim

Twist / Pose

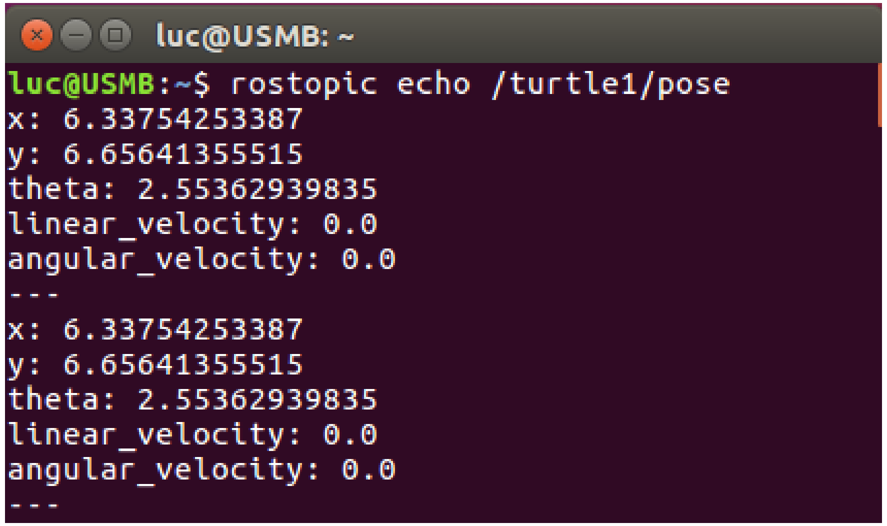
Test moving the turtle
(command from the Terminal)



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

Test printing out the turtle position
(command from the Terminal)

```
> rostopic echo /turtle1/pose
```



```
luc@USMB:~$ rostopic echo /turtle1/pose
x: 6.33754253387
y: 6.65641355515
theta: 2.55362939835
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 6.33754253387
y: 6.65641355515
theta: 2.55362939835
linear_velocity: 0.0
angular_velocity: 0.0
---
```

move_turtle_linear_node (Python)

Writing the Node

Create package

```
> cd ~/catkin_ws/src/
> catkin_create_pkg turtlesim_tutorials rospy
```

Edit script

```
> cd ~/catkin_ws/src/turtlesim_tutorials
> mkdir scripts
> subl move_turtle_linear_node.py
```

Make script executable

```
> cd ~/catkin_ws/src/turtlesim_tutorials/scripts
> sudo chmod +x move_turtle_linear_node.py
```

Make package and source environment

```
> cd ~/catkin_ws
> catkin_make
> source ~/catkin_ws/devel/setup.bash
```

move_turtle_linear_node.py

```
#! /usr/bin/env python3

import rospy
XXXXXXXXXXXX  # import Twist message

def move_turtle():
    # Initialize node
    XXXXXXXXXXXX

    # Create a publisher to "talk" to Turtlesim
    pub = XXXXXXXXXXXX

    # Create a Twist message and add linear x values
    vel = Twist()
    vel.linear.x = 1.0  # Move along the x axis only

    # Save current time and set publish rate at 10 Hz
    tStart = rospy.Time.now()
    rate = rospy.Rate(10)

    # For the next 6 seconds publish vel move commands to Turtlesim
    while rospy.Time.now() < tStart + rospy.Duration.from_sec(6):
        XXXXXXXXXXXX # publish velocity command to Turtlesim
        rate.sleep()

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

https://https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_linear_node.py

move_turtle_linear_node (Python)

Writing the Node

Create package

```
> cd ~/catkin_ws/src/
> catkin_create_pkg turtlesim_tutorials rospy
```

Edit script

```
> cd ~/catkin_ws/src/turtlesim_tutorials
> mkdir scripts
> subl move_turtle_linear_node.py
```

Make script executable

```
> cd ~/catkin_ws/src/turtlesim_tutorials/scripts
> sudo chmod +x move_turtle_linear_node.py
```

Make package and source environment

```
> cd ~/catkin_ws
> catkin_make
> source ~/catkin_ws/devel/setup.bash
```

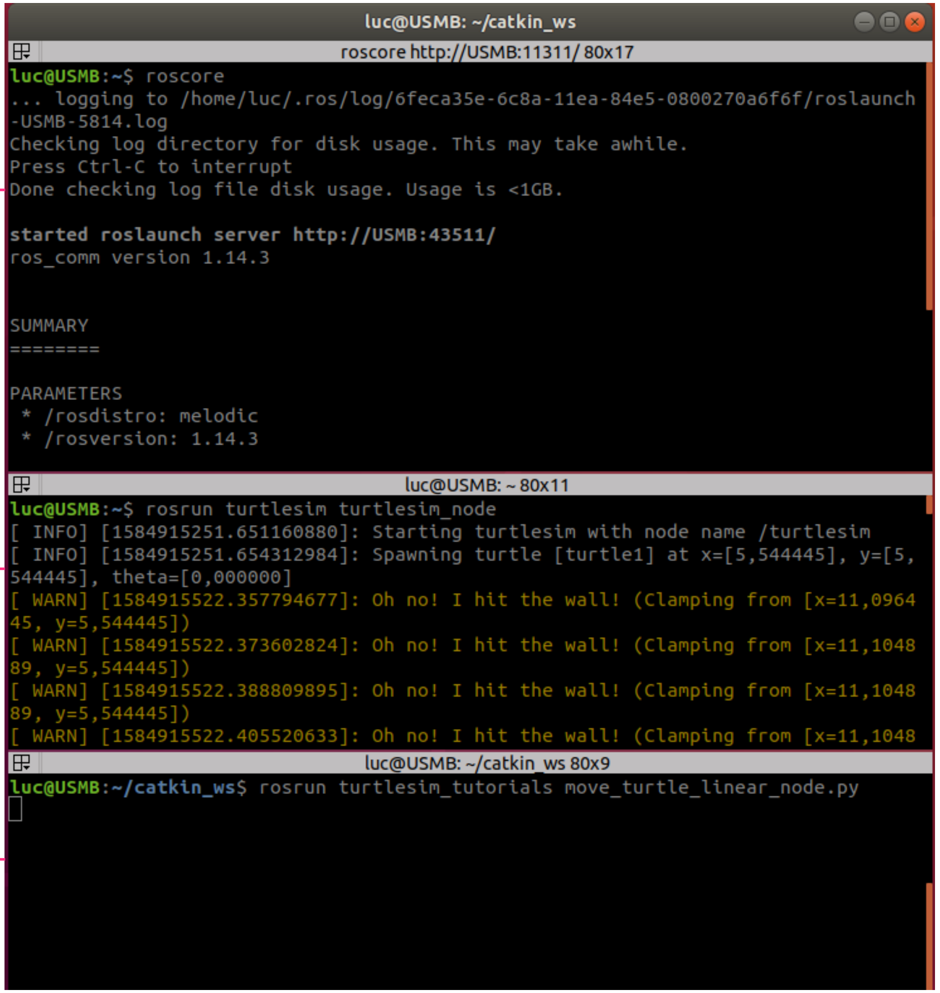
move_turtle_linear_node.py

```python
#! /usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist  # import Twist message

def move_turtle():
    # Initialize node
    rospy.init_node('move_turtle_linear_node', anonymous=False)

    # Create a publisher to "talk" to Turtlesim
    pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size=1)

    # Create a Twist message and add linear x values
    vel = Twist() # Creates a Twist object
    vel.linear.x = 1.0  # Move along the x axis only

    # Save current time and set publish rate at 10 Hz
    tStart = rospy.Time.now()
    rate = rospy.Rate(10)

    # For the next 6 seconds publish vel move commands to Turtlesim
    while rospy.Time.now() < tStart + rospy.Duration.from_sec(6):
        pub.publish(vel) # publish velocity command to Turtlesim
        rate.sleep()

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

https://https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_linear_node.py

move_turtle_linear_node (Python)

Run the Node



start roscore

```
luc@USMB:~$ roscore
... logging to /home/luc/.ros/log/6feca35e-6c8a-11ea-84e5-0800270a6f6f/roslaunch
-USMB-5814.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://USMB:43511/
ros_comm version 1.14.3


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3
```

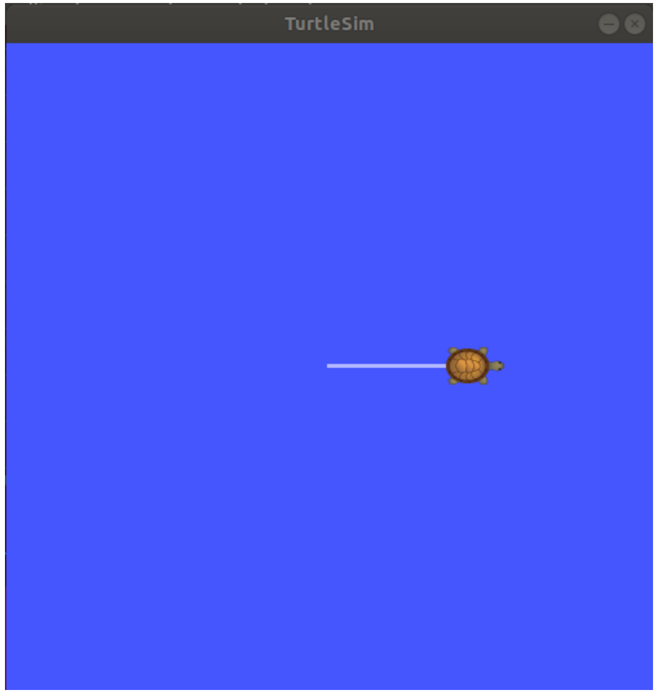
run the turtlesim_node

```
luc@USMB:~$ rosrun turtlesim turtlesim_node
[ INFO] [1584915251.651160880]: Starting turtlesim with node name /turtlesim
[ INFO] [1584915251.654312984]: Spawning turtle [turtle1] at x=[5,544445], y=[5,
544445], theta=[0,000000]
[ WARN] [1584915522.357794677]: Oh no! I hit the wall! (Clamping from [x=11,0964
45, y=5,544445])
[ WARN] [1584915522.373602824]: Oh no! I hit the wall! (Clamping from [x=11,1048
89, y=5,544445])
[ WARN] [1584915522.388809895]: Oh no! I hit the wall! (Clamping from [x=11,1048
89, y=5,544445])
[ WARN] [1584915522.405520633]: Oh no! I hit the wall! (Clamping from [x=11,1048
```

run your node !

```
luc@USMB:~/catkin_ws$ rosrun turtlesim_tutorials move_turtle_linear_node.py
```

move_turtle_linear_node.py



https://https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_linear_node.py

move_turtle_command_node (Python)

Adding command line arguments

move_turtle_command_node.py

```
#! /usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist

# Handling command line arguments
import sys # Python sys module to get the command-line arguments
           # inside our code

def move_turtle(lin_vel, ang_vel):
    rospy.init_node('move_turtle_command', anonymous=False)

    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    rate = rospy.Rate(10) # 10hz

    vel = Twist()  # creates a Twist object

    while not rospy.is_shutdown():

        # Adding linear and angular velocity to the message
        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0

        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = ang_vel
```

Display information in the Console

```
        rospy.loginfo("Linear Vel = %f: Angular Vel =%f",lin_vel,ang_vel)

        #Publishing Twist message
        pub.publish(vel)

        rate.sleep()


if __name__ == '__main__':
    #Providing linear and angular velocity through command line
    move_turtle(float(sys.argv[1]),float(sys.argv[2]))
```

Run the node

```
> rosrun turtlesim_tutorials move_turtle_command_node.py 0.5 0.2
```

arguments

linear.x angular.z

https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_command_node.py

move_turtle_printout_node (Python)

Adding the turtle position print out

move_turtle_printout_node.py

```python
#! /usr/bin/env python3

import rospy
XXXXXXXXXXXX  # import Twist message
XXXXXXXXXXXX  # import Pose message

import sys

# callback for topic /turtle1/Pose
def pose_callback(XXXXX):
    XXXXXXXXXX # printout in the console the pose of turtle1

def move_turtle(lin_vel,ang_vel):
    rospy.init_node('move_turtle', anonymous=False)

    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

    # Creating new subscriber. Topic name: /turtle1/pose
    #                          Callback name: pose_callback
    XXXXXXXXXXXXXXXXXXX

    rate = rospy.Rate(10) # 10hz

    vel = Twist()
```

```python
    while not rospy.is_shutdown():
        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0

        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = ang_vel

        rospy.loginfo("Linear Vel = %f: Angular Vel
=%f",lin_vel,ang_vel)

        pub.publish(vel)

        rate.sleep()

if __name__ == '__main__':
    # Providing linear and angular velocity through command line
    move_turtle(float(sys.argv[1]),float(sys.argv[2]))
```

https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_printout_node.py

move_turtle_printout_node (Python)

Adding the turtle position print out

move_turtle_printout_node.py

```
#! /usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist    # import Twist message
from turtlesim.msg import Pose         # import Pose message

import sys

# callback for topic /turtle1/Pose
def pose_callback(pose):
    rospy.loginfo("Robot X = %f : Y=%f : Z=%f\n",pose.x,pose.y,pose.theta)

def move_turtle(lin_vel,ang_vel):
    rospy.init_node('move_turtle', anonymous=False)

    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

    # Creating new subscriber. Topic name: /turtle1/pose
    #                          Callback name: pose_callback
    rospy.Subscriber('/turtle1/pose', Pose, pose_callback)

    rate = rospy.Rate(10) # 10hz

    vel = Twist()
```

```
    while not rospy.is_shutdown():
        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0

        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = ang_vel

        rospy.loginfo("Linear Vel = %f: Angular Vel
=%f",lin_vel,ang_vel)

        pub.publish(vel)

        rate.sleep()

if __name__ == '__main__':
 # Providing linear and angular velocity through command line
    move_turtle(float(sys.argv[1]),float(sys.argv[2]))
```

https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_printout_node.py

move_turtle_feedback_node (Python)

Adding the position feedback

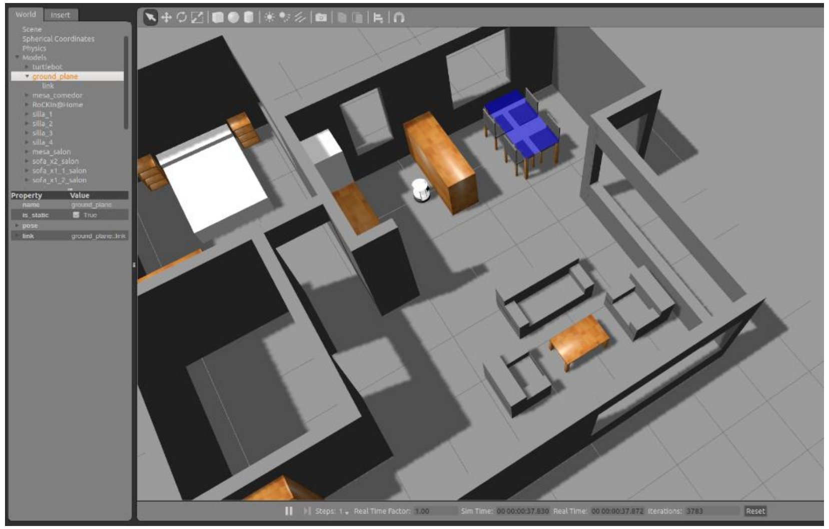
move_turtle_feedback_node.py

```
#! /usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose

import sys

robot_x = 0

# callback for topic /turtle1/Pose
def pose_callback(pose):
    global robot_x
    rospy.loginfo("Robot X = %f\n", pose.x)
    robot_x = pose.x

def move_turtle(lin_vel,ang_vel,distance):

    global robot_x

    rospy.init_node('move_turtle', anonymous=False)
    pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    rospy.Subscriber('/turtle1/pose',Pose, pose_callback)
    rate = rospy.Rate(10) # 10hz
    vel = Twist()
```

```
    while not rospy.is_shutdown():

        vel.linear.x = lin_vel
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = ang_vel

# Checking the robot distance is greater than the commanded distance
# If it is greater, stop the node
        if(robot_x >= distance):
            rospy.loginfo("Robot Reached destination")
            rospy.logwarn("Stopping robot")

            break

        pub.publish(vel)
        rate.sleep()

if __name__ == '__main__':
    #Providing linear and angular velocity through command line
    move_turtle(float(sys.argv[1]),float(sys.argv[2]),
float(sys.argv[3]))
```

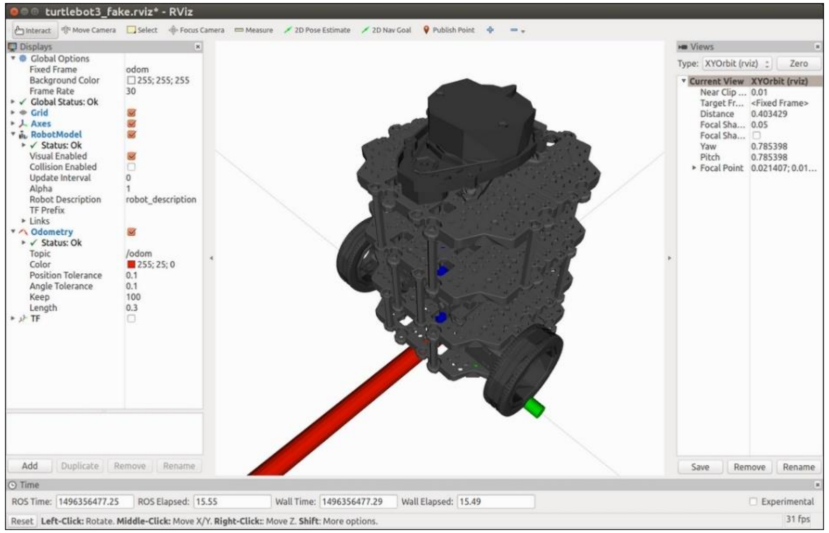
https://raw.githubusercontent.com/LucMarechal/USMB_INFO802_ROS_Lectures/master/turtlesim_tutorials/move_turtle_feedback_node.py

Robotic simulation scenarios



Property	Value
name	ground_plane
is_static	☑ True
pose	
link	ground_plane::link

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:37.830 Real Time: 00 00:00:37.872 Iterations: 3783



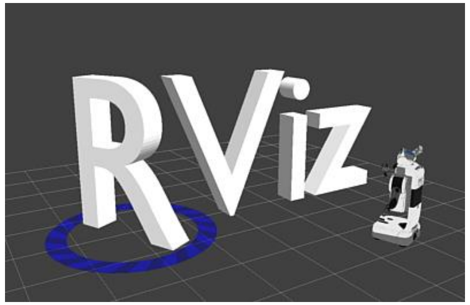
Global Options	
Fixed Frame	odom
Background Color	□ 255; 255; 255
Frame Rate	30
✓ Global Status: Ok	
Grid	☑
Axes	☑
RobotModel	☑
✓ Status: Ok	
Visual Enabled	☑
Collision Enabled	□
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
Odometry	☑
✓ Status: Ok	
Topic	/odom
Color	🟥 255; 25; 0
Position Tolerance	0.1
Angle Tolerance	0.1
Keep	100
Length	0.3
TF	□

Type: XYOrbit (rviz)

Current View	XYOrbit (rviz)
Near Clip ...	0.01
Target Fr...	<Fixed Frame>
Distance	0.403429
Focal Sha...	0.05
Focal Sha...	□
Yaw	0.785398
Pitch	0.785398
Focal Point	0.021407; 0.01...

ROS Time: 1496356477.25 ROS Elapsed: 15.55 Wall Time: 1496356477.29 Wall Elapsed: 15.49 □ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click::** Move Z. **Shift**: More options. 31 fps



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

Gazebo – TurtleBot3 Simulation

Installing TurtleBot3 simulation packages

Installing Gazebo

```
> sudo apt-get install ros-noetic-gazebo-ros-pkgs ros-noetic-gazebo-ros-control
```

Installing TurtleBot3 simulation packages

```
> cd ~/catkin_ws/src/
> git clone -b noetic-devel https://github.com/ROBOTIS-
GIT/turtlebot3_simulations.git
> cd ~/catkin_ws && catkin_make
```

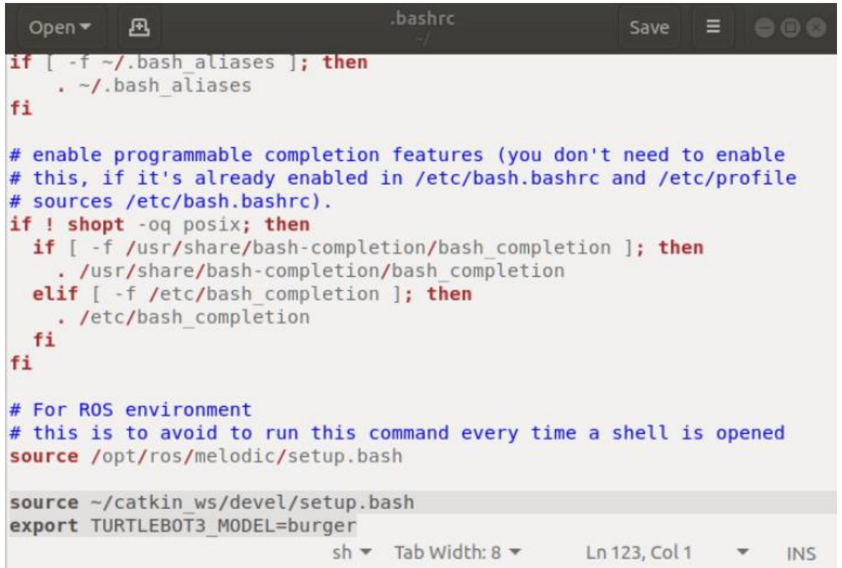
Modify bashrc file

```
> gedit ~/.bashrc
```

to add this 2 settings

```
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle  # setting the model
```

TurtleBot3 has three models, Burger, Waffle, and Waffle Pi, so you have to set which model you want to use



```
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

# For ROS environment
# this is to avoid to run this command every time a shell is opened
source /opt/ros/melodic/setup.bash

source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
```

Gazebo – TurtleBot3 Simulation

Installing TurtleBot3 simulation packages

Reload .bashrc

```
> source ~/.bashrc
```

Download the TurtleBot3 simulation files

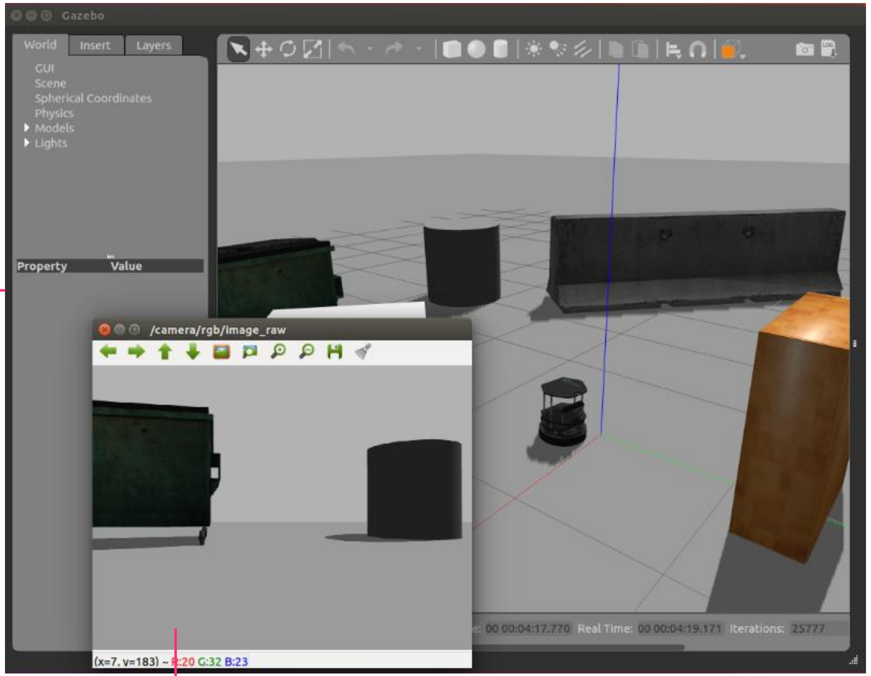
```
> cd ~/catkin_ws/src/
> git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
> cd ~/catkin_ws && catkin_make
```

Gazebo – TurtleBot3 Simulation

Launch Gazebo

```
> roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

(also try : turtlebot3_house.launch)

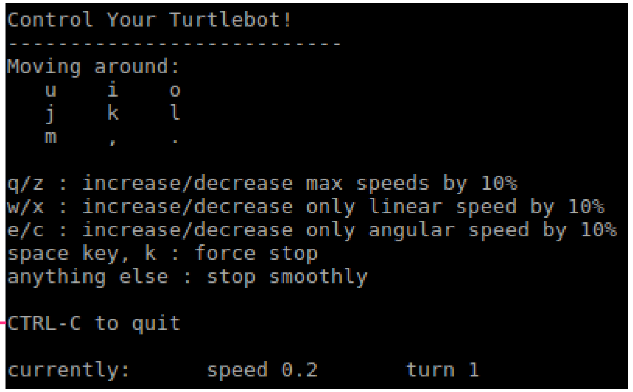


Display the image from the robot

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

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

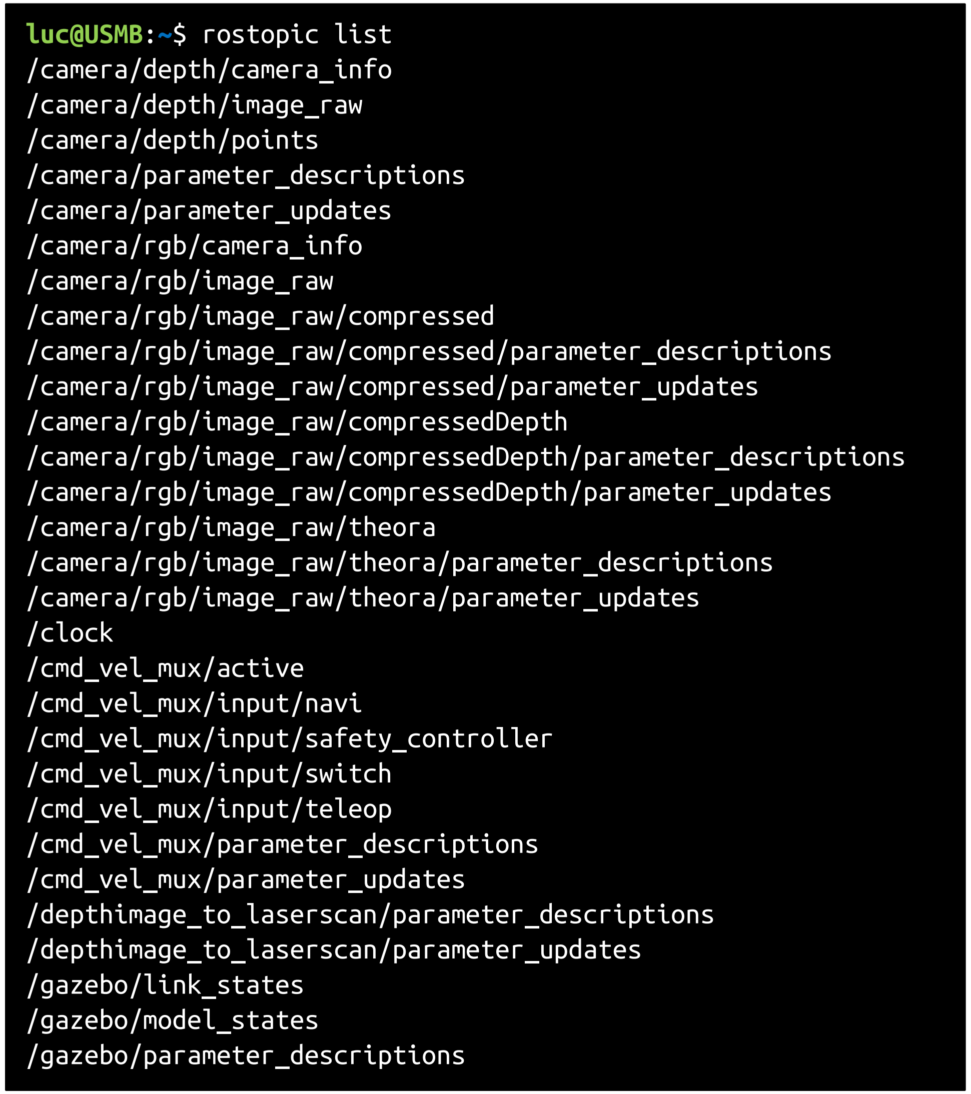
Move the robot with keyboard

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

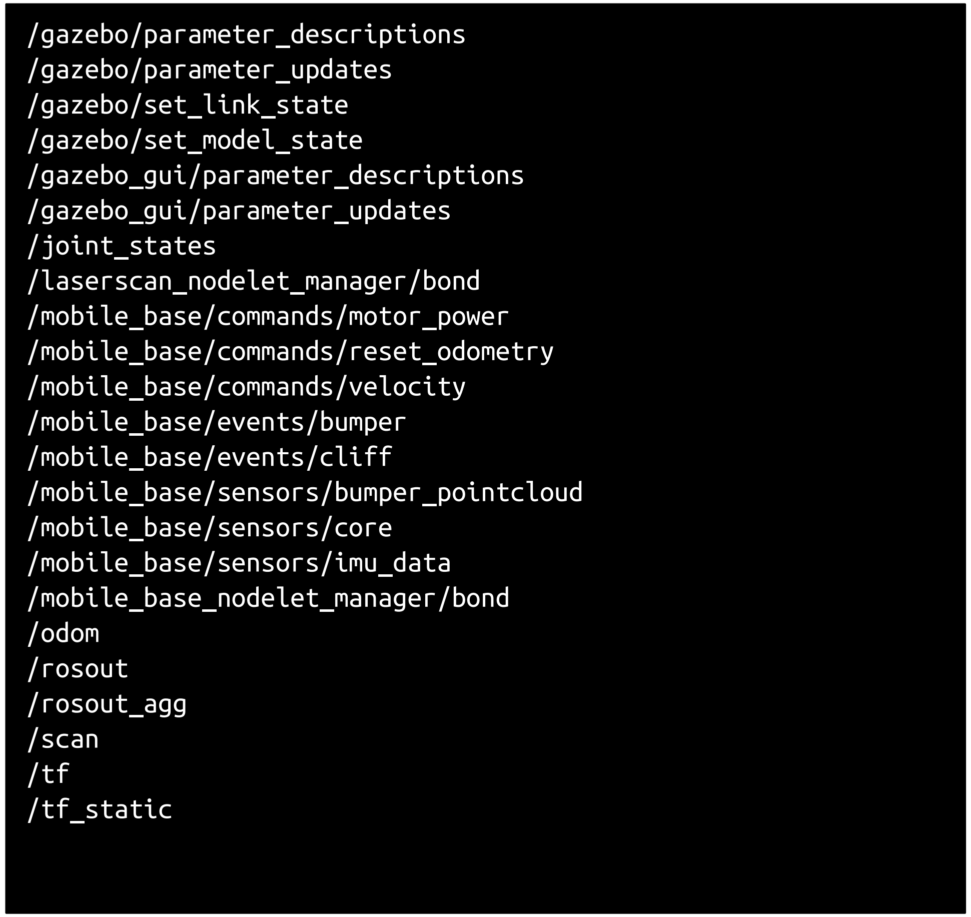


```
Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

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



```
luc@USMB:~$ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/switch
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthimage_to_laserscan/parameter_descriptions
/depthimage_to_laserscan/parameter_updates
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
```



```
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/joint_states
/laserscan_nodelet_manager/bond
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/velocity
/mobile_base/events/bumper
/mobile_base/events/cliff
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/imu_data
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/scan
/tf
/tf_static
```

Gazebo – TurtleBot3 Simulation

Odometry

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

  Twist messages to the topic ***/cmd_vel_mux/input/teleop***

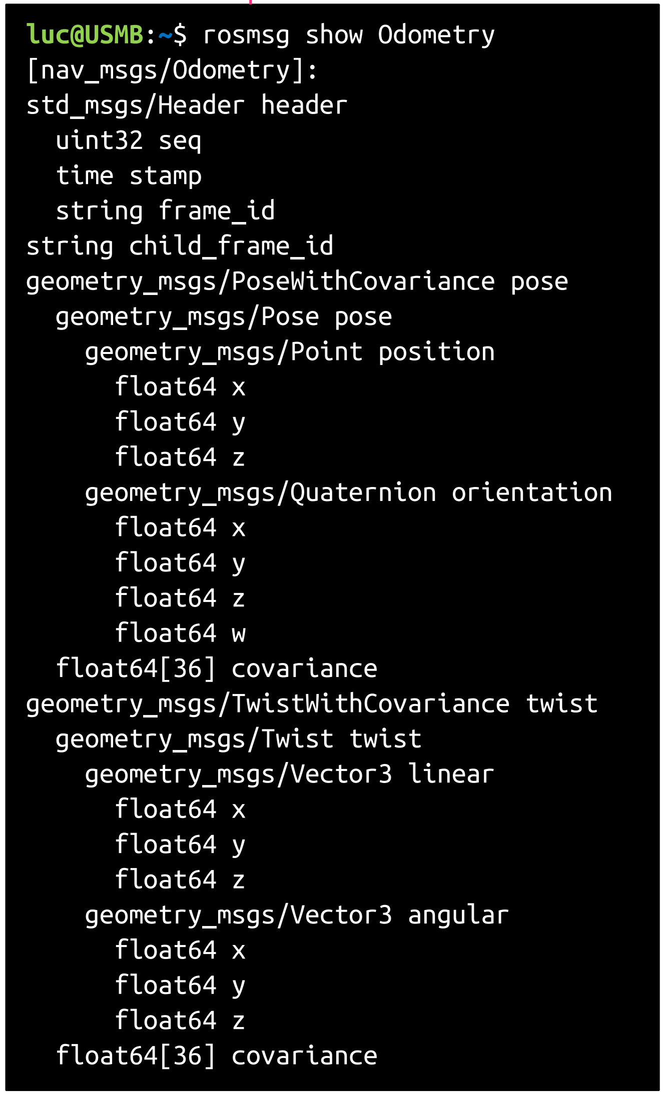
- To get a TurtleBot position and orientation in ROS we need to subscribe:

  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)

```
> rosmsg show Odometry
```



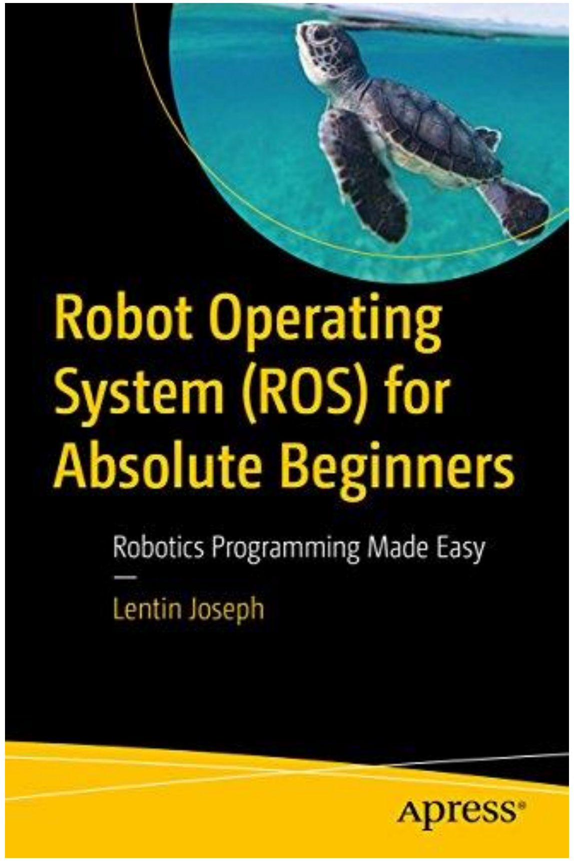
```
luc@USMB:~$ rosmsg show Odometry
[nav_msgs/Odometry]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion 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
```

Further References

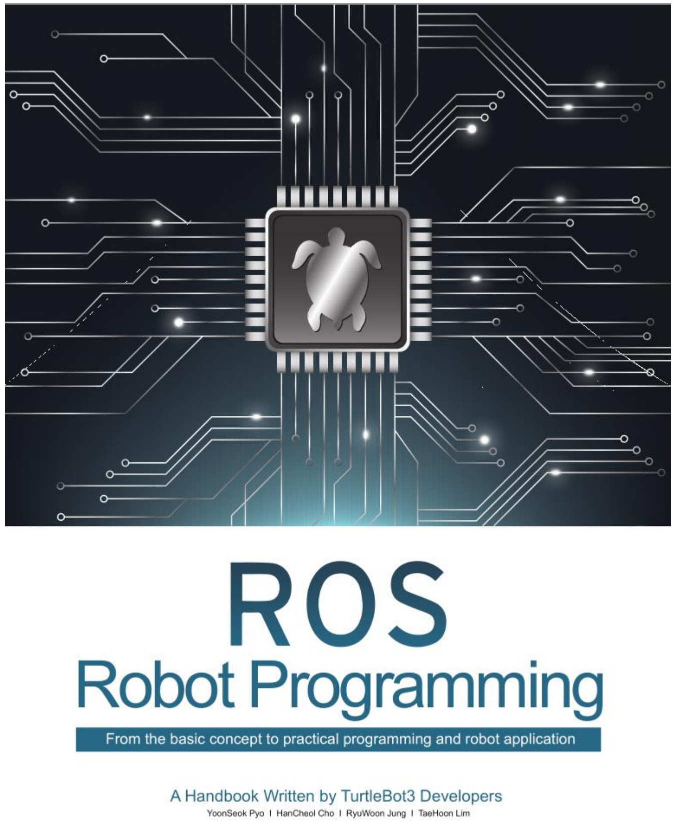
- ROS Turtlesim tutorials
  - wiki.ros.org/turtlesim/Tutorials/

- ROS Cheat Sheet
  - https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
  - https://kapeli.com/cheat_sheets/ROS.docset/

Relevant books and sources


Robot Operating System (ROS) for Absolute Beginners
Robotics Programming Made Easy
Lentin Joseph
Apress

Chapter 5


ROS
Robot Programming
From the basic concept to practical programming and robot application
A Handbook Written by TurtleBot3 Developers
YoonSeok Pyo I HanCheol Cho I RyuWoon Jung I TaeHoon Lim

Chapter 10

Contact Information

Université Savoie Mont Blanc

Polytech' Annecy Chambery
Chemin de Bellevue
74940 Annecy
France

https://www.polytech.univ-savoie.fr

Lecturer

Luc Marechal (luc.marechal@univ-smb.fr)
Polytech Annecy Chambéry
SYMME Lab (Systems and Materials for Mechatronics)