2020

# INFO 802

## Master Advanced Mechatronics

Luc Marechal

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

ROS

# Course 4
# Robot control

# Course 5 : Robot control

## Outline

- Turtlesim topic, messages and commands
- Move Turtle
- Gazebo TurtleBot simulation
- TurtleBot3 real robot
- Assignement

# Course 5 : Robot control

## Objectives

- Being able to know witch topics are at stake in a node
- Being able to know what kind of message and its source package
- Write a publisher function
- Write a subscriber function and understand its callback

# Turtlesim

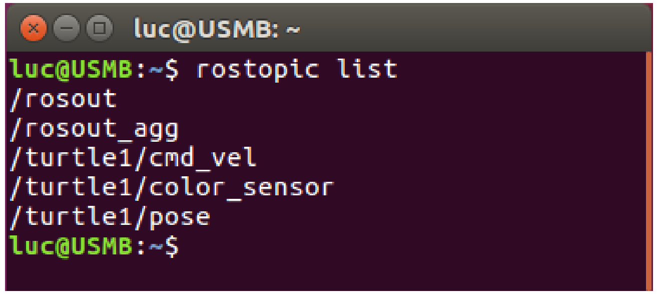
- Questions to answer:

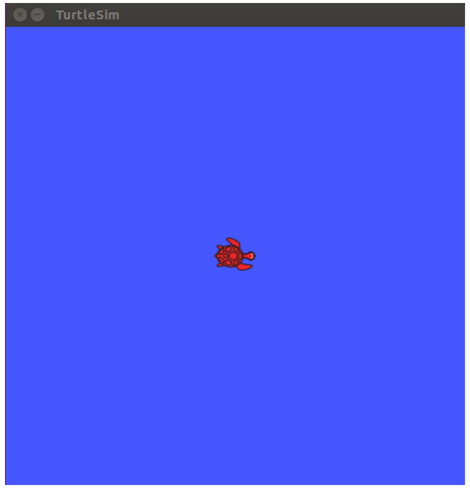
Which topic is the velocity command published to?

Which topic is position information available from?

What kind of messages are used?

Which message package are they from?

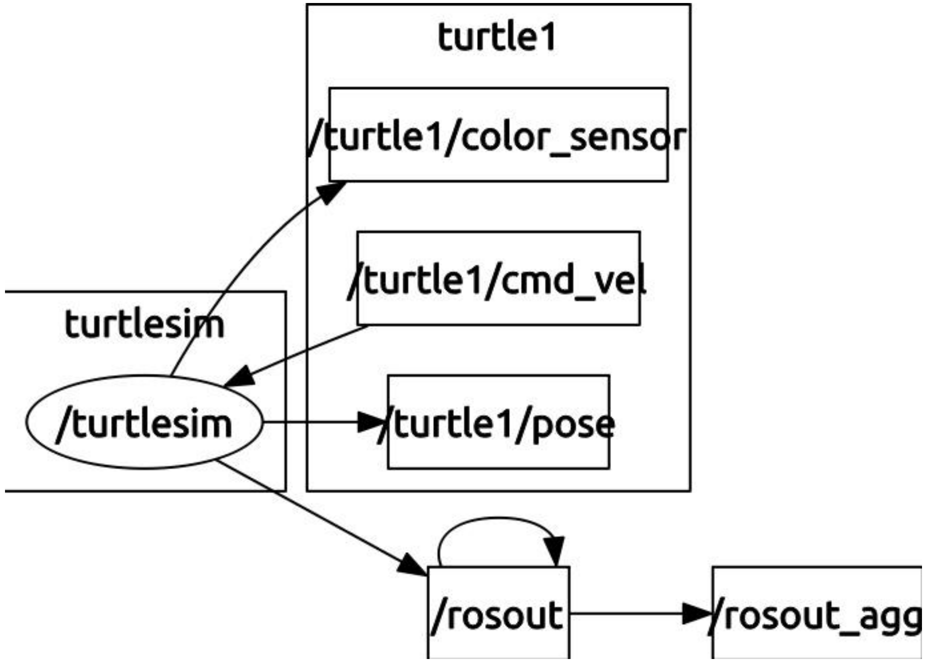



```
> rosrun turtlesim turtlesim_node
> rostopic list
> rosnode info turtlesim_node
```

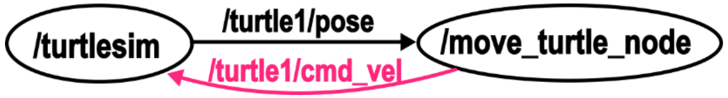
Visualize node and topic

```
> rqt_graph
```



# Turtlesim

## Twist




```
> 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_msg`* message package

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

Example of use

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

create a *Twist* object

set the linear velocity along x

set the angular rate about z

# Turtlesim

## Pose


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

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

```
> 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

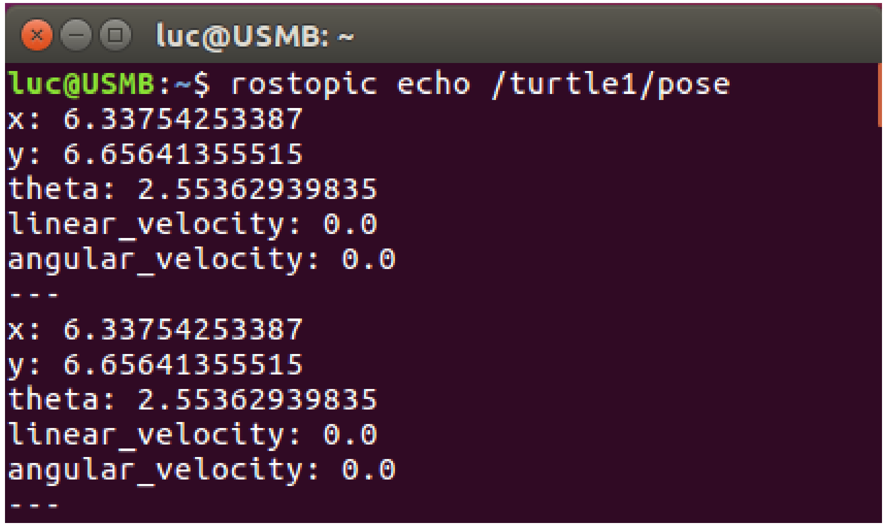
## Twist / Pose

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
```



# Turtlesim

## Turtle_teleop_key node

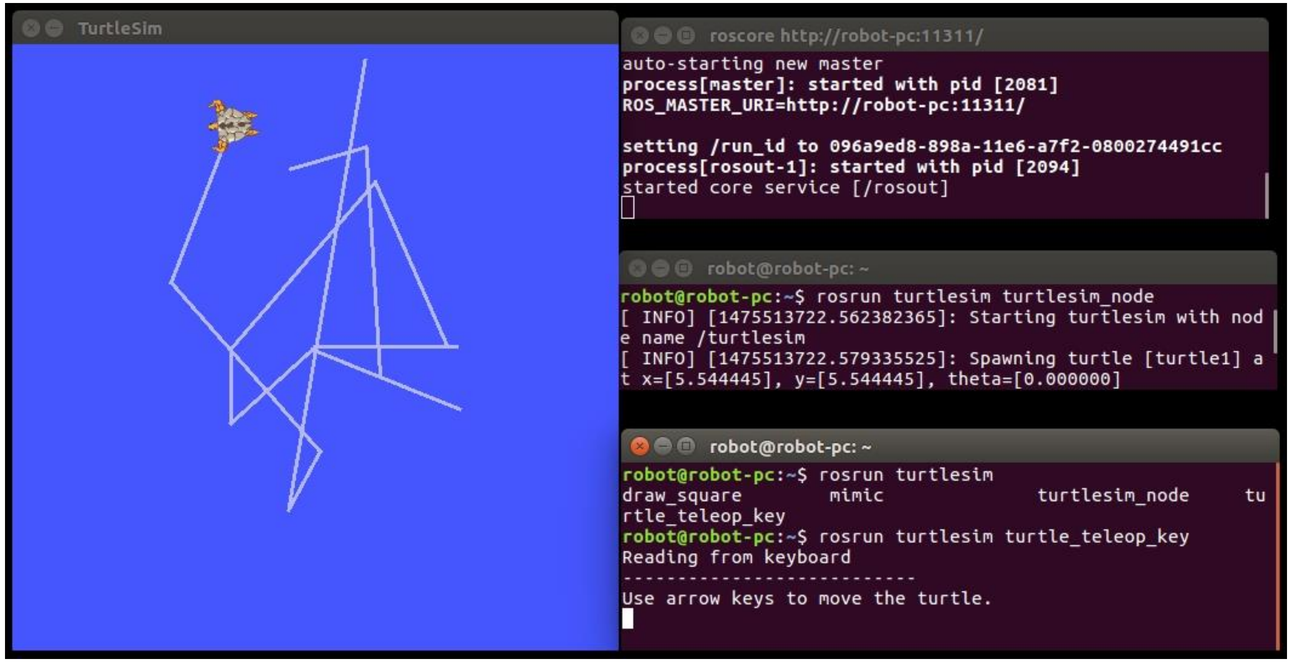
Test moving the turtle
(with the *turtle_teleop_key* node)

Recall: Open a terminal for each command

```
> Roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```




The terminal which *turtle_teleop_key* is running on MUST be active.
Run the keyboard teleoperation node. Change the turtle's position by pressing arrow keys on the keyboard.

# 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
> gedit 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 ./devel/setup.bash
```

*move_turtle_linear_node.py*

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

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__':
    try:
        move_turtle()
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/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
> gedit 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 ./devel/setup.bash
```

*move_turtle_linear_node.py*

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

import rospy
from geometry_msgs.msg import Twist

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__':
    try:
        move_turtle()
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/master/turtlesim_tutorials/move_turtle_linear_node.py

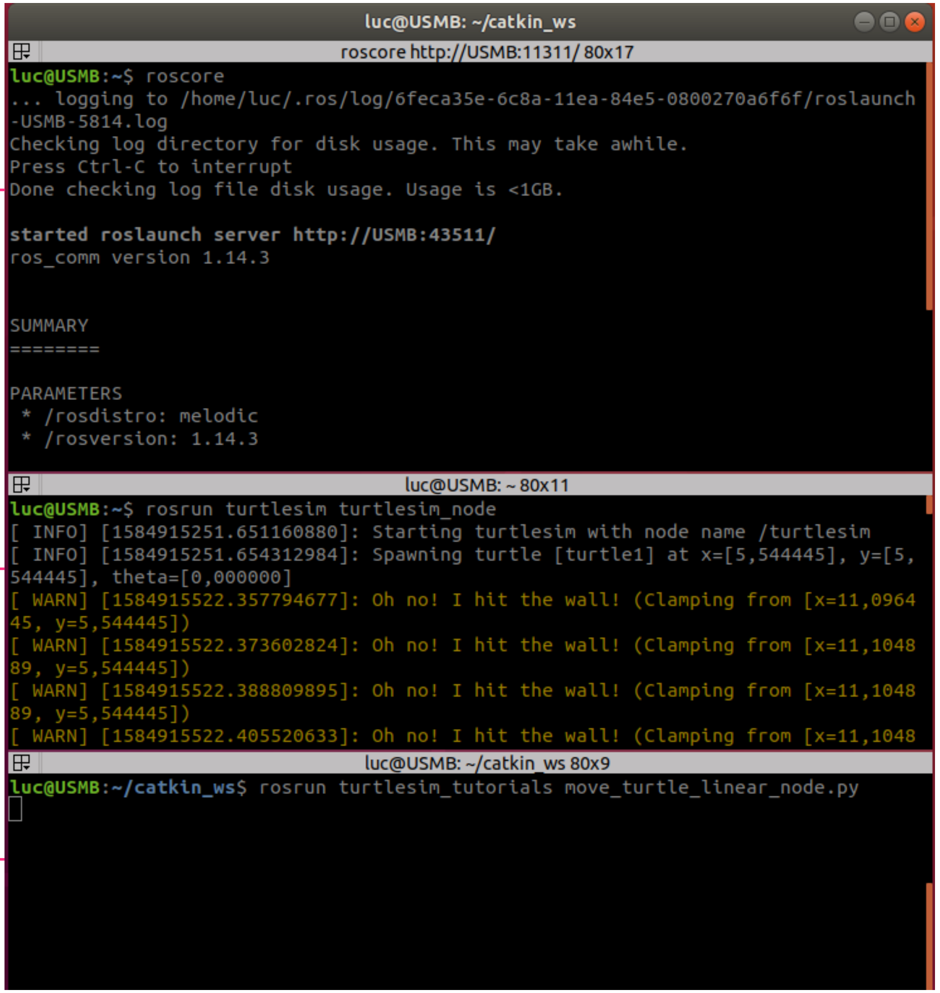
# move_turtle_linear_node (Python)

## Run the Node

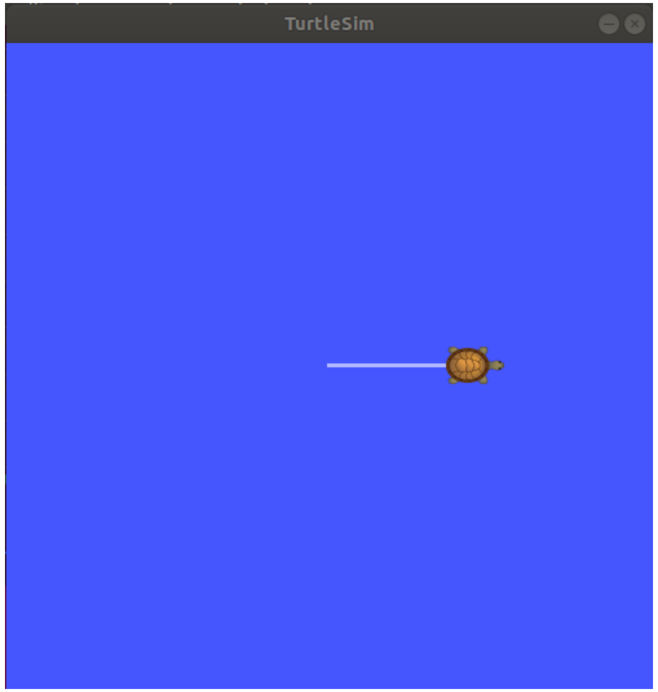
`start roscore`

`run the turtlesim_node`

`run your node !`




*move_turtle_linear_node.py*




https://github.com/LucMarechal/ROS_Lectures/blob/master/turtlesim_tutorials/move_turtle_linear_node.py

# move_turtle_command_node (Python)

## Adding command line arguments

*move_turtle_command_node.py*

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

import rospy
from geometry_msgs.msg import Twist

#Handling command line arguments
import sys

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

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

    vel = Twist()

    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

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

        #Publishing Twist message
        pub.publish(vel)

        rate.sleep()

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

### Run the node

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

arguments

https://github.com/LucMarechal/ROS_Lectures/blob/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 python

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

import sys

#/turtle1/Pose topic callback
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__':
    try:
    #Providing linear and angular velocity through command line
        move_turtle(float(sys.argv[1]),float(sys.argv[2]))
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/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 python

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

import sys

#/turtle1/Pose topic callback
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()
```

```
    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__':
    try:
    #Providing linear and angular velocity through command line
        move_turtle(float(sys.argv[1]),float(sys.argv[2]))
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/master/turtlesim_tutorials/move_turtle_printout_node.py

# move_turtle_printout_node (Python)

## Adding the turtle position print out

*move_turtle_printout_node.py*

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

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

import sys

#/turtle1/Pose topic callback
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()
```

```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__':
    try:
    #Providing linear and angular velocity through command line
        move_turtle(float(sys.argv[1]),float(sys.argv[2]))
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/master/turtlesim_tutorials/move_turtle_printout_node.py

# move_turtle_feedback_node (Python)

## Adding the position feedback

*move_turtle_feedback_node.py*

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

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

import sys

robot_x =0

#/turtle1/Pose topic callback
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()
```

```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

#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__':
    try:
    #Providing linear and angular velocity through command line
        move_turtle(float(sys.argv[1]),float(sys.argv[2]),
float(sys.argv[3]))
    except rospy.ROSInterruptException:
        pass
```

https://github.com/LucMarechal/ROS_Lectures/blob/master/turtlesim_tutorials/move_turtle_feedback_node.py

# Gazebo - TurtleBot 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)

GAZEBO

**More Info and tutorials**
http://gazebosim.org/tutorials

# Gazebo - TurtleBot Simulation

Installing TurtleBot simulation packages

Installing turtlebot simulation packages

```
> sudo apt-get update

> sudo apt-get install ros-kinetic-turtlebot-gazebo ros-kinetic-turtlebot-
simulator ros-kinetic-turtlebot-description ros-kinetic-turtlebot-teleop
```

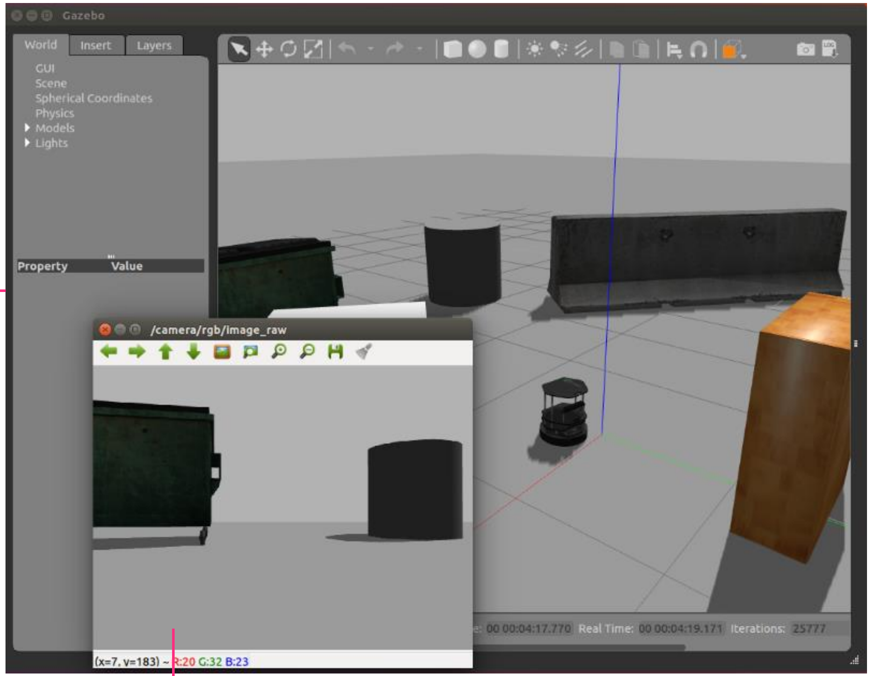
Configure environment parameter TURTLEBOT_GAZEBO_WORLD_FILE

```
> Export
TURTLEBOT_GAZEBO_WORLD_FILE=/opt/ros/kinetic/share/turtlebot_gazebo/worlds/play
ground.world
```

# Gazebo - TurtleBot Simulation

Launch Gazebo

```
> roslaunch turtlebot_gazebo turtlebot_world.launch
```

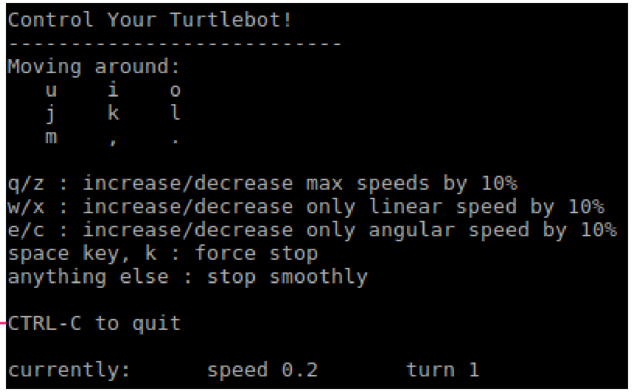



Display the image from the robot

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

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

Move the robot with keyboard

```
> roslaunch turtlebot_teleop keyboard_teleop.launch
```



# Gazebo - TurtleBot 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 - TurtleBot 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
```

# move_turtlebot_node (Python)

Add distance

move_turtlebot_node.py

```
#! /usr/bin/python

import rospy
from geometry_msgs.msg import Twist
import sys

def move_turtle():

    rospy.init_node('move_turtlebot', anonymous=False)
    pub = rospy.Publisher('/cmd_vel_mux/input/teleop', 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
```

https://github.com/LucMarechal/ROS_Lectures/blob/a3096bfa5d4fb130f4b455e7ecf89456e67458c5/turtlesim_tutorials/move_turtle_feedback_node.py

# LaserScan Message

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

```
# Single scan from a planar laser range-finder

Header header
# stamp: The acquisition time of the first ray in the scan.
# frame_id: The laser is assumed to spin around the positive Z axis
# (counterclockwise, if Z is up) with the zero angle 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.
```

# TurtleBot3

## Setup


On remote PC

On TurtleBot AND on remote PC

```
> sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy ros-kinetic-
teleop-twistkeyboard ros-kinetic-laser-proc ros-kinetic-rgbd-launch ros-kinetic-
depthimage-to-laserscan roskinetic-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
```

# TurtleBot3

## Setup


### On TurtleBot

```
> cd ~/catkin_ws/src
> git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
> git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
> git clone https://github.com/ROBOTIS-GIT/hls_lfcd_lds_driver.git
> cd ~/catkin_ws && catkin_make
```

### On remote PC

```
> cd ~/catkin_ws/src/
> git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
> git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
> git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
> cd ~/catkin_ws && catkin_make
```

# TurtleBot3

## Setup

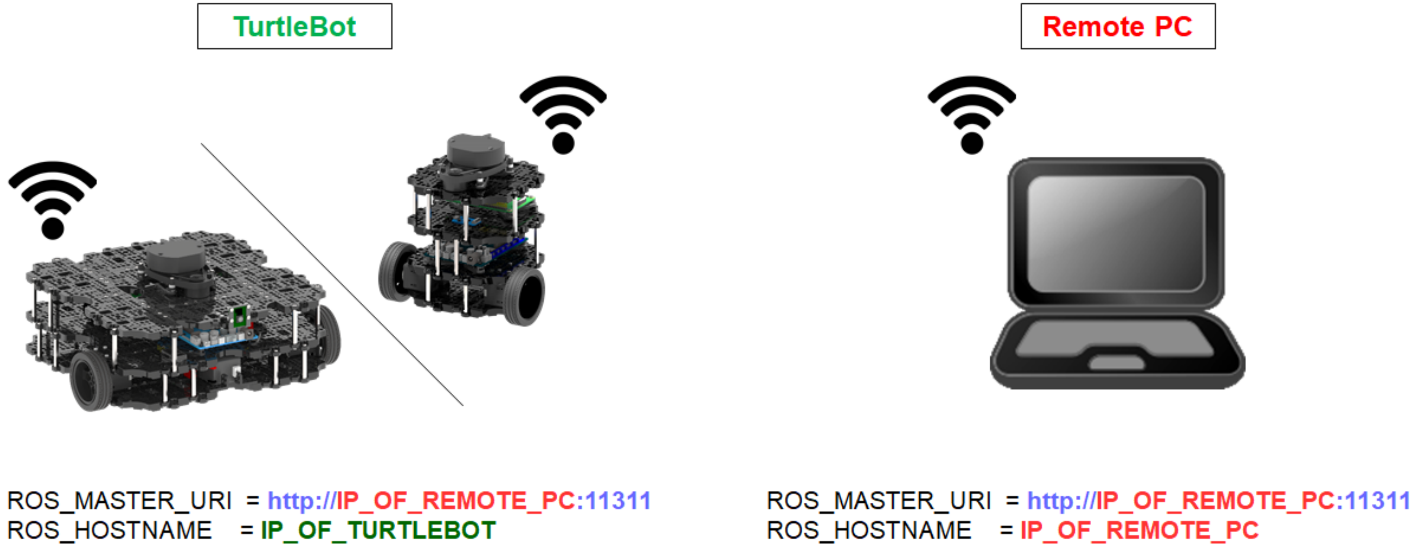
### On remote PC

```
> Sudo gedit ~/.bashrc
  export ROS_MASTER_URI =
  export ROS_HOSTNAME =
  export TURTLEBOT_MODEL = burger

> source ~/.bashrc
```

### On TurtleBot

```
> Sudo gedit ~/.bashrc
  export ROS_MASTER_URI =
  export ROS_HOSTNAME =

> source ~/.bashrc
```

```
> roslaunch turtlebot3_bringup turtlebot3_robot.launch
```




* Example when ROS Master is running on the Remote PC

**More Info**
http://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/

# TurtleBot3

## Setup

SSH (Secure Shell)

- cryptographic network protocol that allows to log in to another computer and run command on a remote system

Install SSH

```
> sudo apt-get install ssh
```

Connect to a remote computer

```
> ssh username on the remote PC  @ ip address of the remote PC

  Example: > ssh turtlebot@192.168.1.100
```

Enable and start SSH

```
> sudo systemctl enable ssh
> sudo systemctl start ssh
```

**More info**
https://www.raspberrypi.org/documentation/remote-access/ssh/

# TurtleBot3

## TurtleBot3 remote control

Start roscore (Execute on the remote PC only)

```
> roscore
```

Run TurtleBot3 (Execute on the TurtleBot)

```
> roslaunch turtlebot3_bringup turtlebot3_robot.launch --screen
```

Control TurtleBot with keyboard (Execute from the Remote PC)

```
> roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```

Control TurtleBot with PS3 joystick (Execute from the Remote PC)

```
> sudo apt-get install ros-kinetic-joy ros-kinetic-joystick-
drivers ros-kinetic-teleop-twist-joy
```

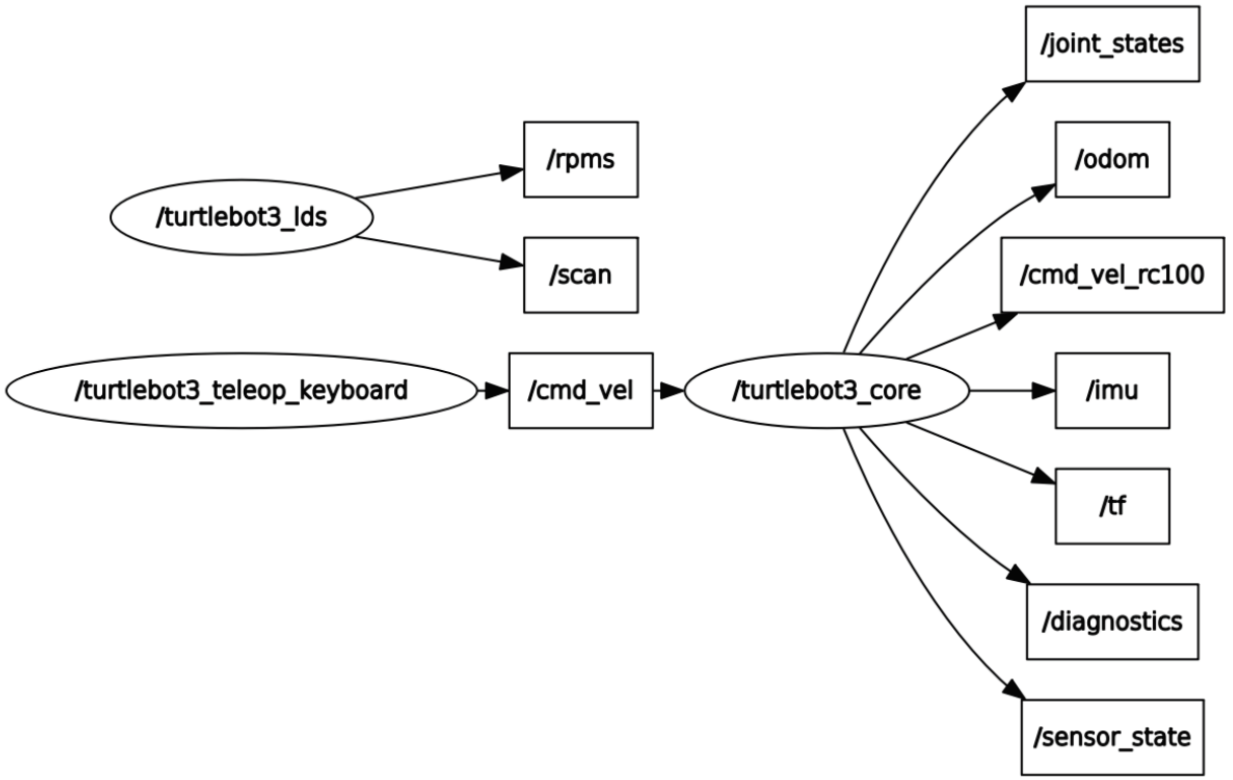
```
> roslaunch teleop_twist_joy teleop.launch --screen
```

# TurtleBot3

## Node and topic

Visualize

```
> rqt_graph
```



# SUMMARY

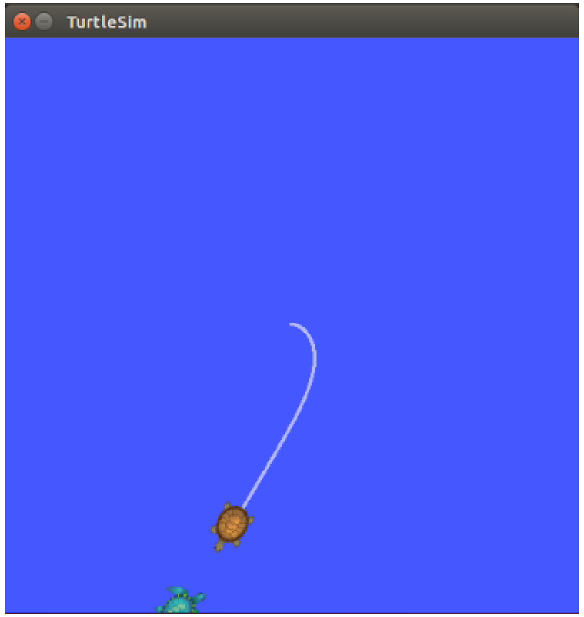
- Twist
- Pose
- Odometry

# Assignement

## move_turtle_to_target_node (Python)

- Due to Monday May 6, 2019. (6pm max. deadline)
- Send me by email the answers to questions in the further slides
- Attach your move_turtle_to_target_node.py code

- luc.marechal@univ-smb.fr



# Assignement

## move_turtle_to_target_node (Python)

- Create a node move_turtle_to_target_node.py that automatically move a turtle1 to a turtle2. For that you will use linear velocity and angular velocity to control the turtle1.

- The forward velocity is defined as a constant gain of multiplied by the distance between the target turtle2 and the turtle1. This means that the forward velocity is higher the further you are away from the target, and goes to zero as you approach the target.

- The angular velocity is calculated similarly with a gain multiplied by the difference in angle between the line that is directly connecting the robot and the goal position, and the angular pose of the robot itself (check the meaning of the atan2 in the steering angle computation). This causes the robot to adjust its own theta to eventually move in a straight line to the target.

# Assignement

## Moving to a Point (x,y) in the 2D plane

- Euclidean distance: $distance = K_v\sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2}$

- Orientation: $\theta_{goal} = \text{atan2}\frac{(y_{goal} - y)}{(x_{goal} - x)}$

- Proportional Controller: Velocity $v = K_v \times distance$

  Steering angle $\gamma = K_h \times \theta_{goal}$

- Moving to a Point (x*, y*) in the 2D plane
  - Linear velocity $v^* = K_v\sqrt{(x^* - x)^2 + (y^* - y)^2}$
  - Steering Angle $\theta^* = \tan^{-1}\frac{y^* - y}{x^* - x}$
  - Proportional Controller $\gamma = K_h(\theta^* \ominus \theta),\ K_h > 0$
    - turns the steering wheel toward the target

Shema

# Assignement

## move_turtle_to_target_node (Python)

Answer the following questions:

- What is the turtlesim Pose message like?
- What should we import in the python header file to use Pose object?

- How can we get the Pose of the turtle?
- How can we get the Pose of the target?

- How can we send velocity command to the turtle?
- What should we then import in the python header file ?

# Assignement

## move_turtle_to_target_node (Python)

Download the files: `turtlesim_tutorials.launch`
`spawn_turtle.py`
`move_turtle_to_target_node.py`

https://github.com/LucMarechal/ROS_Lectures/tree/a3096bfa5d4fb130f4b455e7ecf89456e67458c5/Assignement_Lecture5

Place them in the right folders inside the `move_turtle` package

Edit the `move_turtle_to_target_node.py` and fill the XXXX in the code

Launch file

```
<launch>
  <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" output="screen"/>
  <node name="spawn_turtle" pkg="move_turtle" type="spawn_turtle.py" output="screen"/>
  <node name="turtlesim_target_node" pkg="move_turtle" type="move_turtle_to_target_node.py" output="screen"/>
</launch>
```

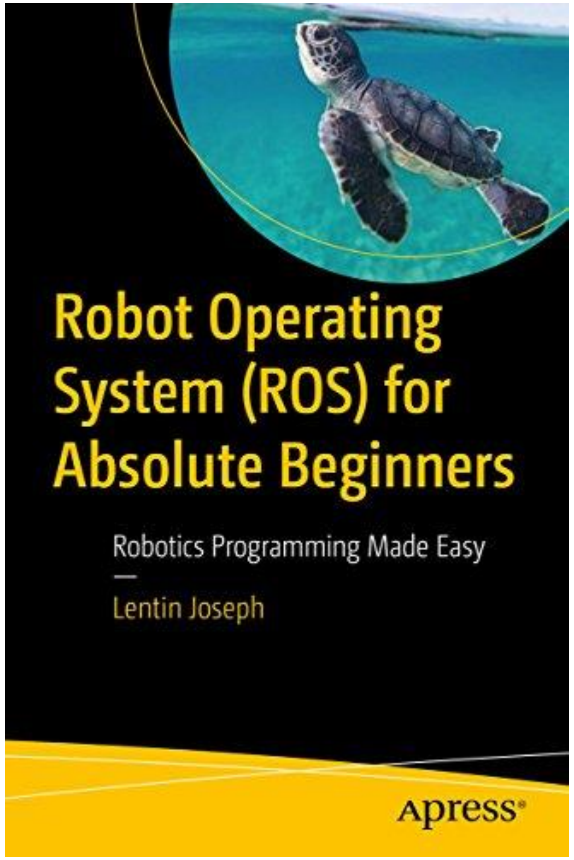
- `spawn_turtle.py : node that randomly spawn the target (i.e turtle2) in the turtlesim window`
- `move_turtle_to_target_node.py : the node that makes the turtle1 move to the turtle2`

# Further References

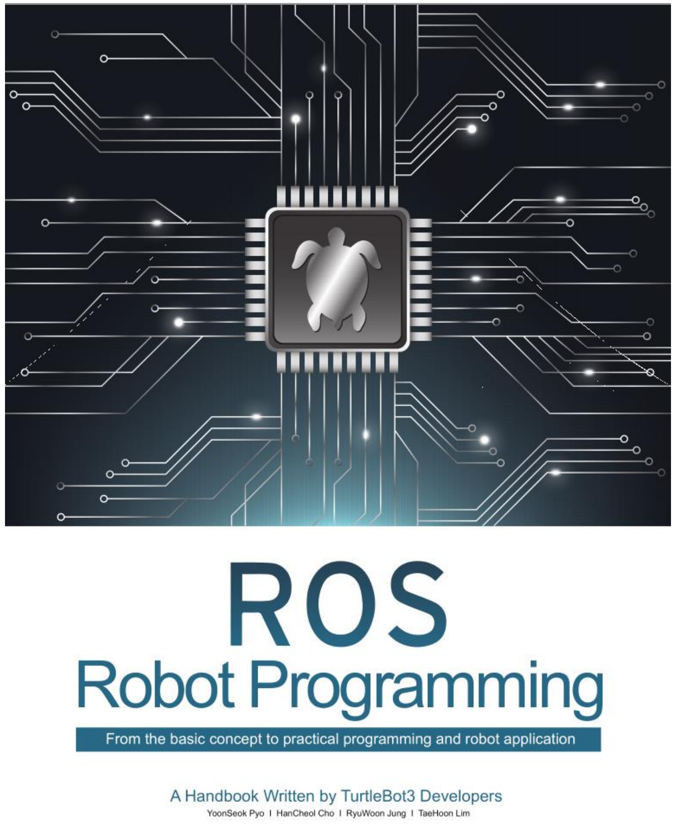
- ROS Turtlesim tutorials
  - wiki.ros.org/turtlesim/Tutorials/

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

# Relevant books and sources




Chapter 5




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)