



Lecture 4

2021

INFO 802

Master Advanced Mechatronics

Luc Marechal



ROS

**Robot control
Turtlesim**

Course 4 : Robot control

Outline

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

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

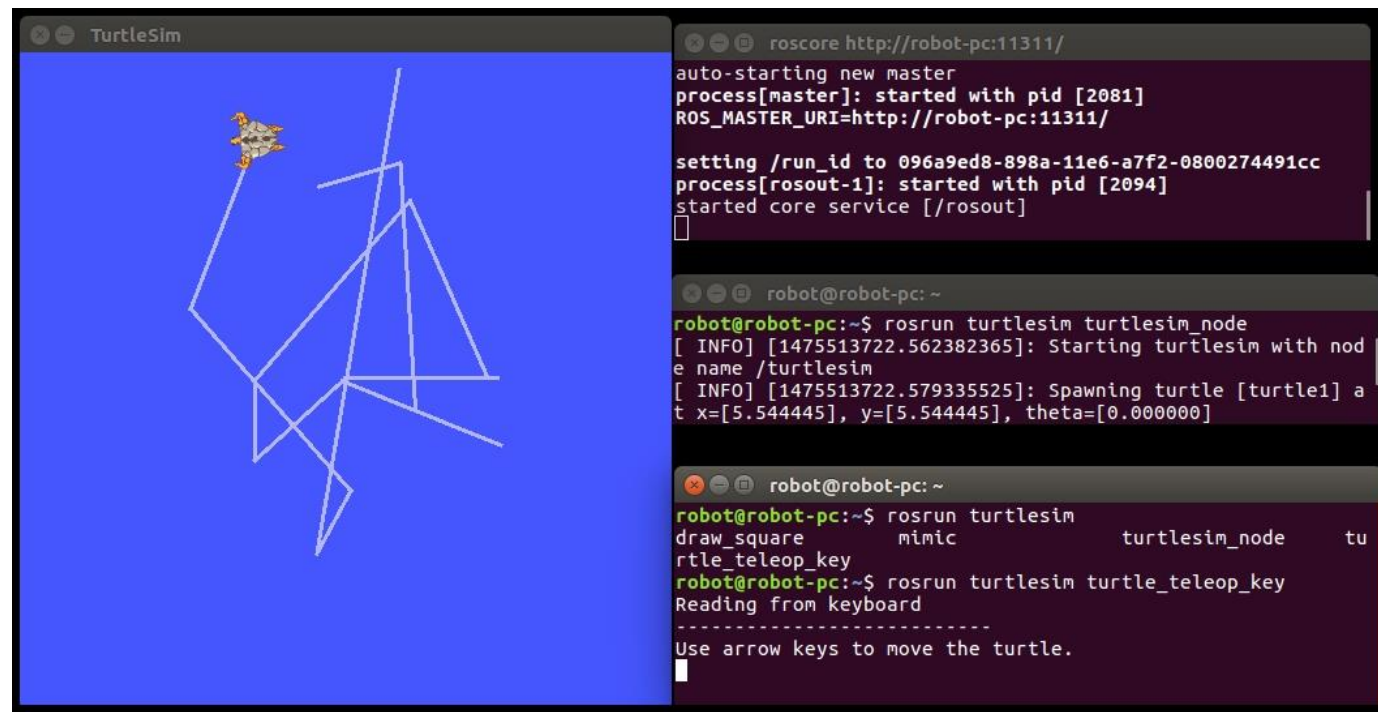
Turtlesim

Recall: Open a terminal for each command

```
> roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```



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?

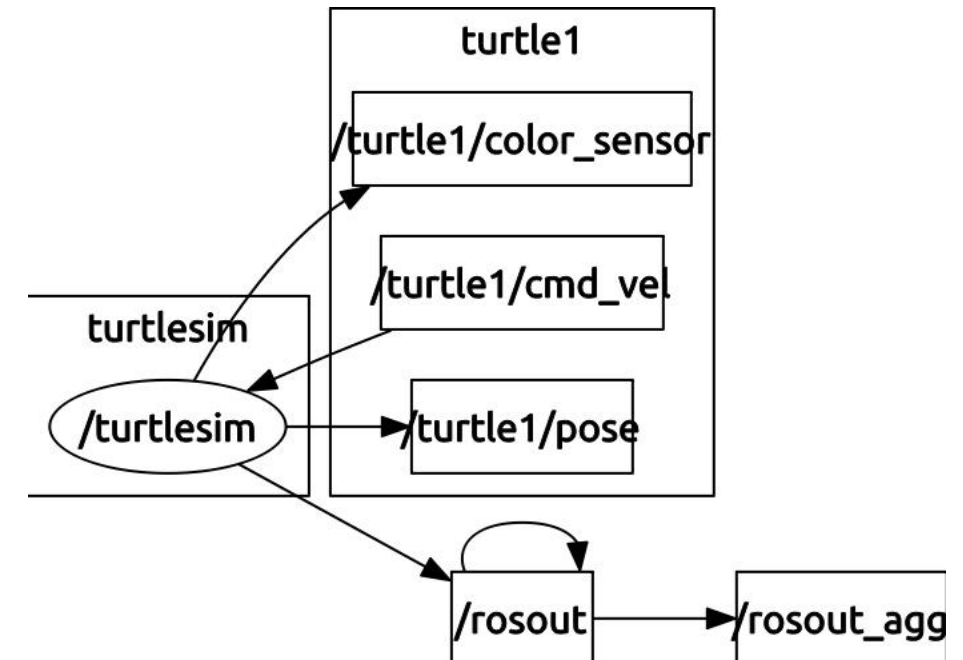
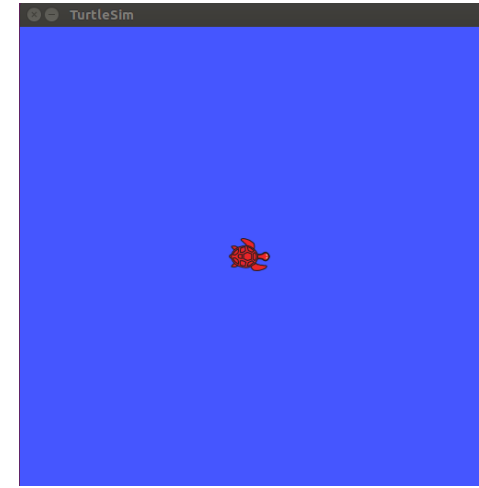
Which message packages are they from?

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

Visualize node and topic

```
> rqt_graph
```

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



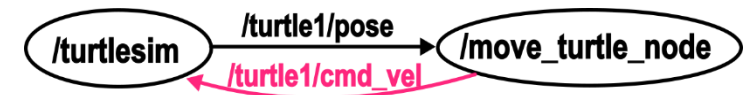
Turtlesim

Twist

- 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)



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

```
luc@USMB:~$ rosmmsg 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
```

Example of use

```
create a Twist object — vel = Twist()  
set the linear velocity along x — vel.linear.x = 1.0  
set the angular rate about z — 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

- 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)



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

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

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
```

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]'
```

topic

message

linear.x linear.y linear.z angular.x angular.y angular.z



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

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

```
luc@USMB: ~  
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  
XXXXXXXXXXXXX # import Twist message  
  
def move_turtle():  
    # Initialize node  
    XXXXXXXXXXXXX  
  
    # Create a publisher to "talk" to Turtlesim  
    pub = XXXXXXXXXXXXX  
  
    # 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):  
        XXXXXXXXXXXXX # publish velocity command to Turtlesim  
        rate.sleep()  
  
if __name__ == '__main__':  
    try:  
        move_turtle()  
    except rospy.ROSInterruptException:  
        pass
```



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



move_turtle_linear_node (Python)

Run the Node

start roscore

run the turtlesim_node

run your node !

```
luc@USMB: ~/catkin_ws
roscore http://USMB:11311/80x17

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
* /roscdistro: melodic
* /rosversion: 1.14.3

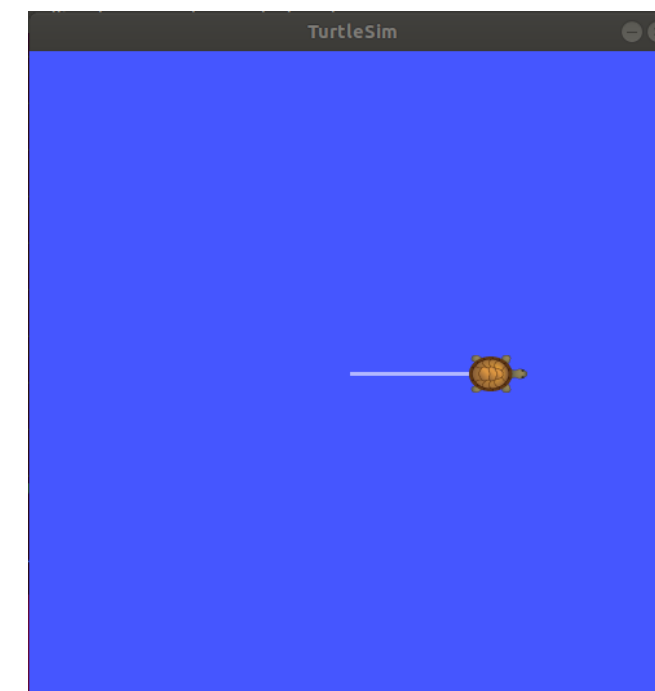
luc@USMB: ~ 80x11

luc@USMB:~$ roslaunch 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=11,096445, y=5,544445])
[ WARN] [1584915522.373602824]: Oh no! I hit the wall! (Clamping from [x=11,104889, y=5,544445])
[ WARN] [1584915522.388809895]: Oh no! I hit the wall! (Clamping from [x=11,104889, y=5,544445])
[ WARN] [1584915522.405520633]: Oh no! I hit the wall! (Clamping from [x=11,104889, y=5,544445])

luc@USMB: ~/catkin_ws 80x9

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

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



move_turtle_printout_node (Python)

Adding the turtle position print out

move_turtle_printout_node.py

```
#!/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):
    XXXXXXXX # 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
    XXXXXXXXXXXXXXXXXXXX

    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]))
```



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]))
```



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")

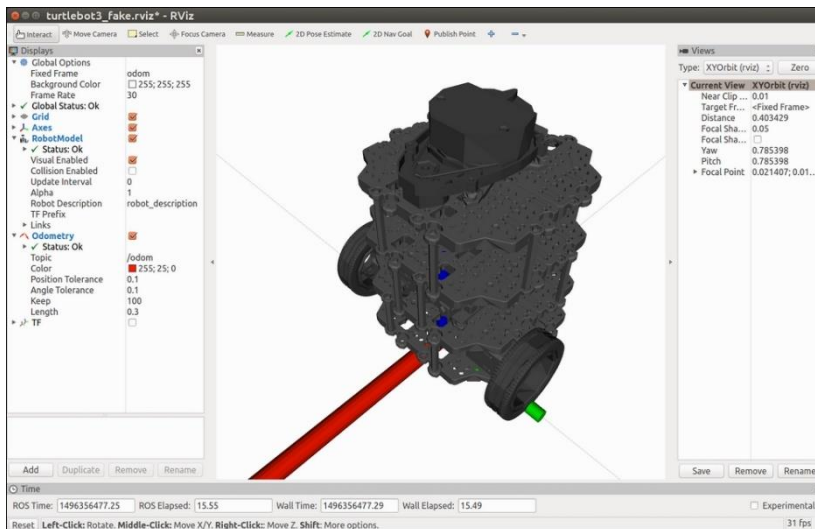
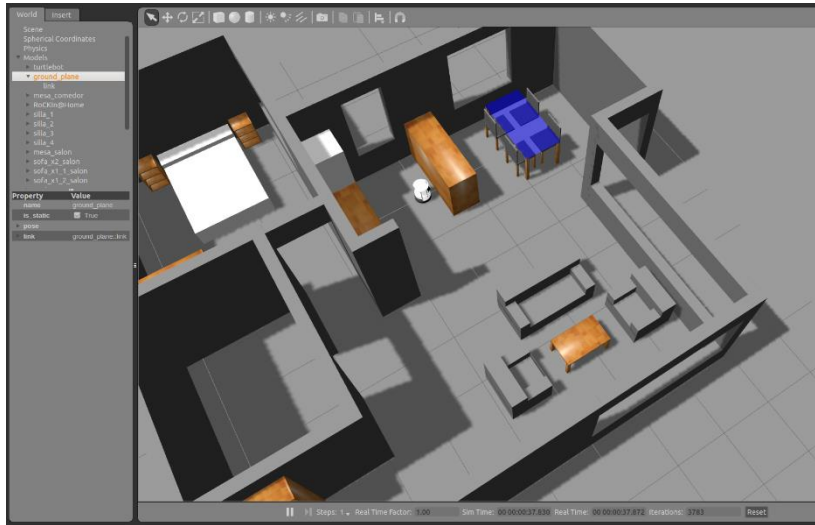
        break

    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]))
```



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>

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



```
Open ▾  .bashrc  Save  ≡  ⌵  ⌵  ⌵  
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  
sh ▾ Tab Width: 8 ▾ Ln 123, Col 1 ▾ INS
```

Gazebo – TurtleBot3 Simulation

Installing TurtleBot3 simulation packages



Reload .bashrc

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

Download the TurtleBot3 simulation files

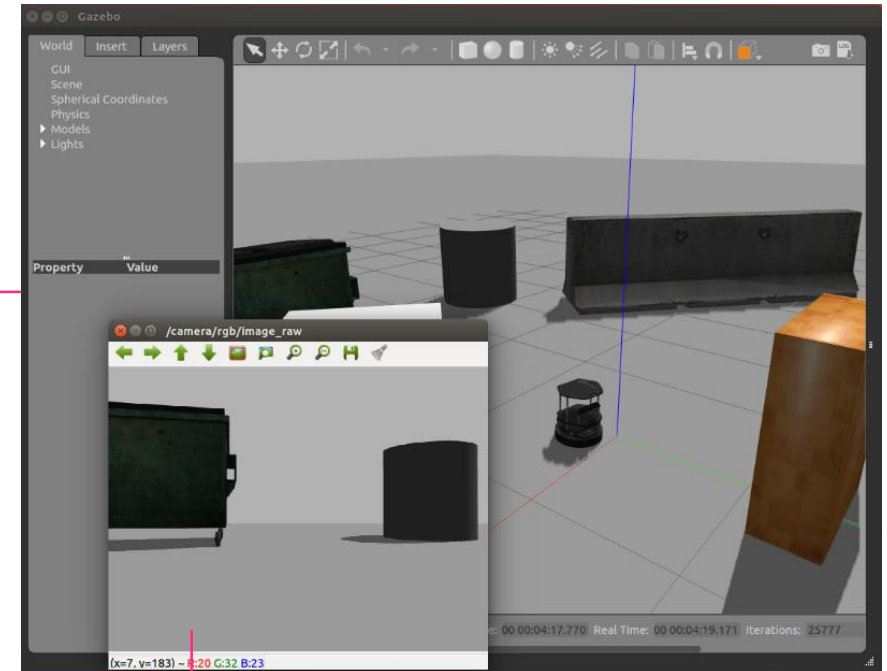
```
> cd ~/catkin_ws/src/  
> git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git  
> cd ~/catkin_ws && catkin_make
```

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/inu_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)

```
> 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
```

move_turtlebot (Python)

Add distance

move_turtlebot.py

```
#!/usr/bin/python3

import rospy
from geometry_msgs.msg import Twist
import sys

def move_turtlebot():

    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__':
    move_turtlebot(float(sys.argv[1]),float(sys.argv[2]))
```

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]

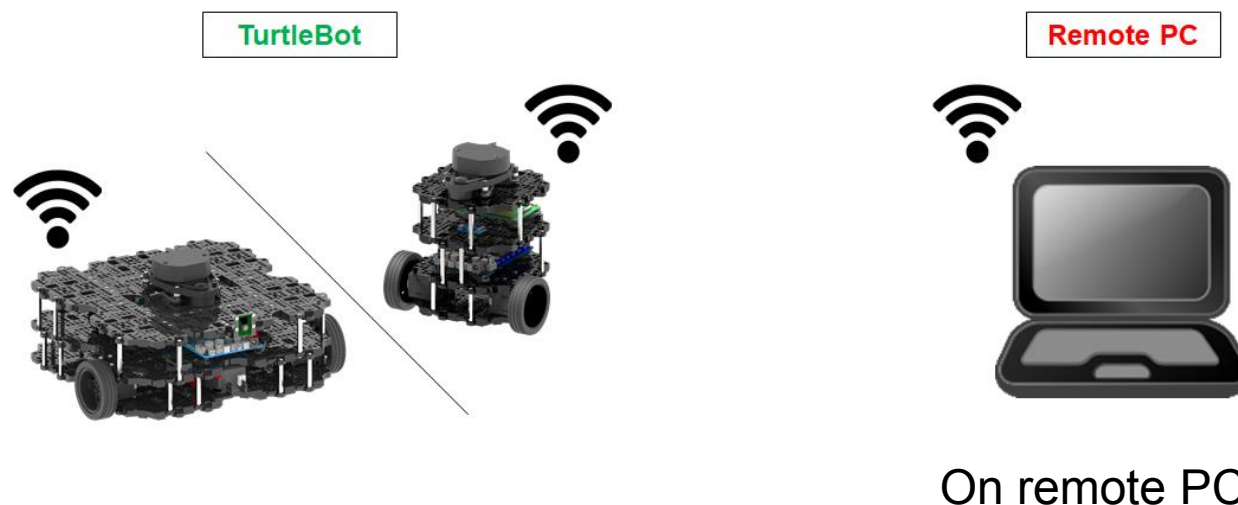
float32 time_increment # time between measurements [seconds] - if your scanner
# is moving, this will be used in interpolating position of 3d points
float32 scan_time # time between scans [seconds]

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 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

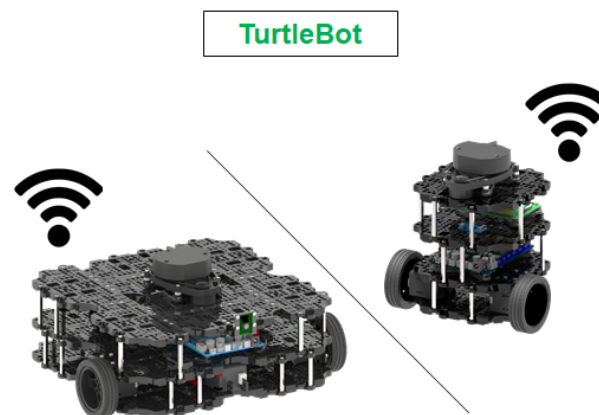
> source ~/.bashrc
```

On TurtleBot

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

> source ~/.bashrc
```

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



```
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

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)

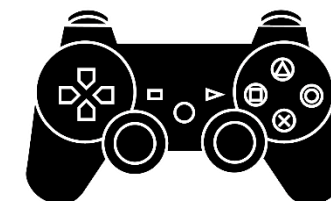
```
> roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch --screen
```



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
```

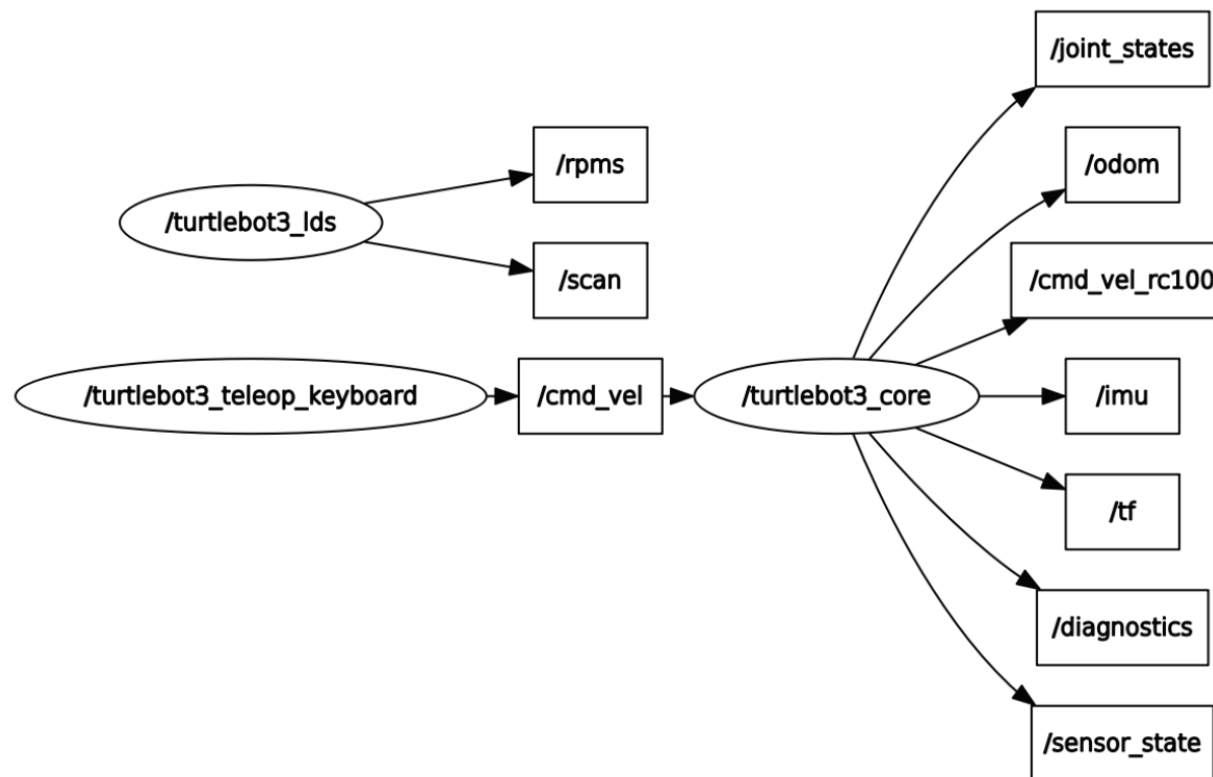


TurtleBot3

Node and topic

Visualize

```
> rqt_graph
```



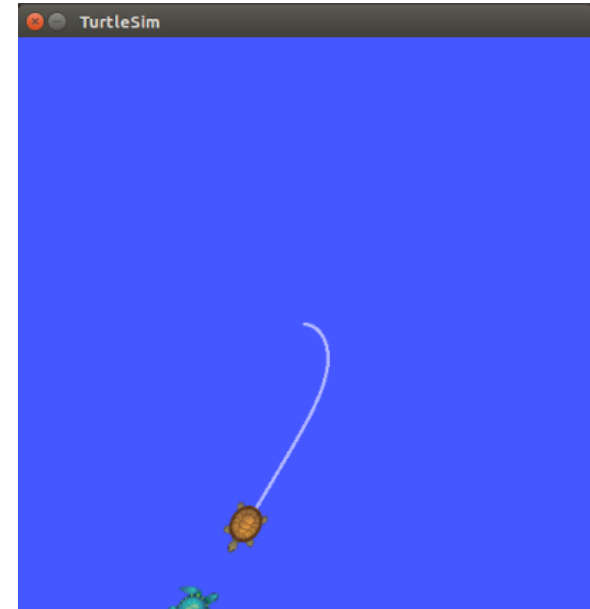
SUMMARY

- Twist
- Pose
- Odometry

Assignement

turtle_to_target (Python)

- Due to Wednesday April 8, 2020. (6pm max. deadline)
- Send me by email the answers to questions in the further slides
- Attach your *turtle_to_target.py* code and launch file
- luc.marechal@univ-smb.fr



Assignement

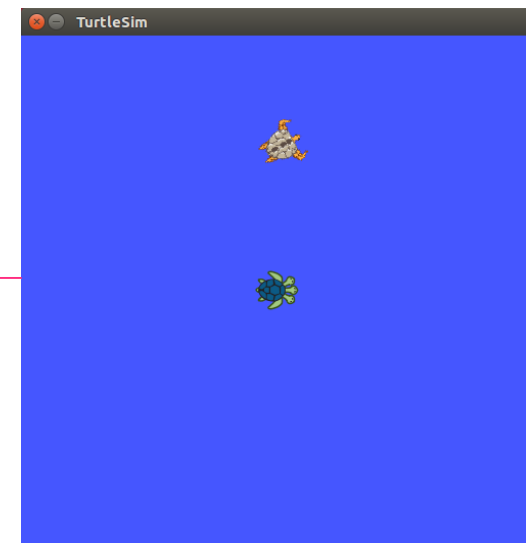
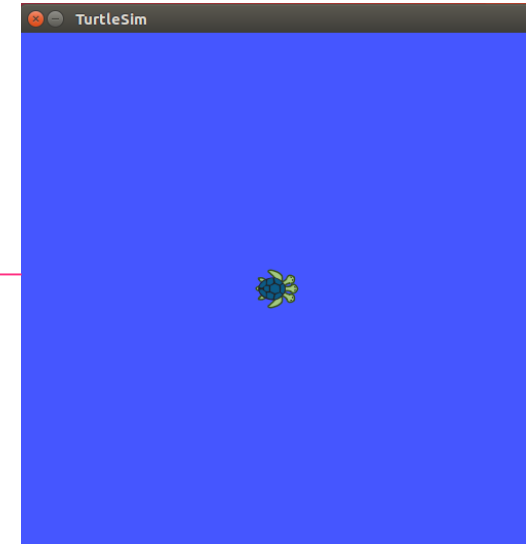
turtle_to_target (Python)

Run the **turtlesim_node** and the **spawn_turtle** node
Find which topics are running on the ROS system

```
luc@USMB:~$ roscore
```

```
luc@USMB:~$ rosrn turtlesim turtlesim_node
```

```
luc@USMB:~$ rosrn turtlesim_tutorials spawn_turtle.py
```



Assignement

turtle_to_target (Python)

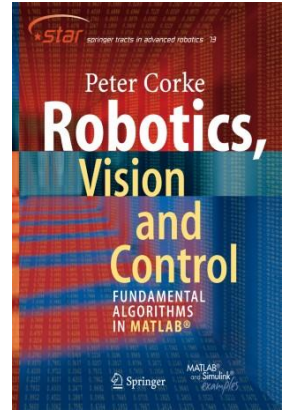
- Create a node called *turtle_to_target* that automatically move the turtle1 to the turtle2. For that you will use linear velocity and angular velocity to control the turtle1.
- The forward velocity is defined as a constant gain of multiplied by the distance between the target turtle2 and the turtle1. 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 turtle 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

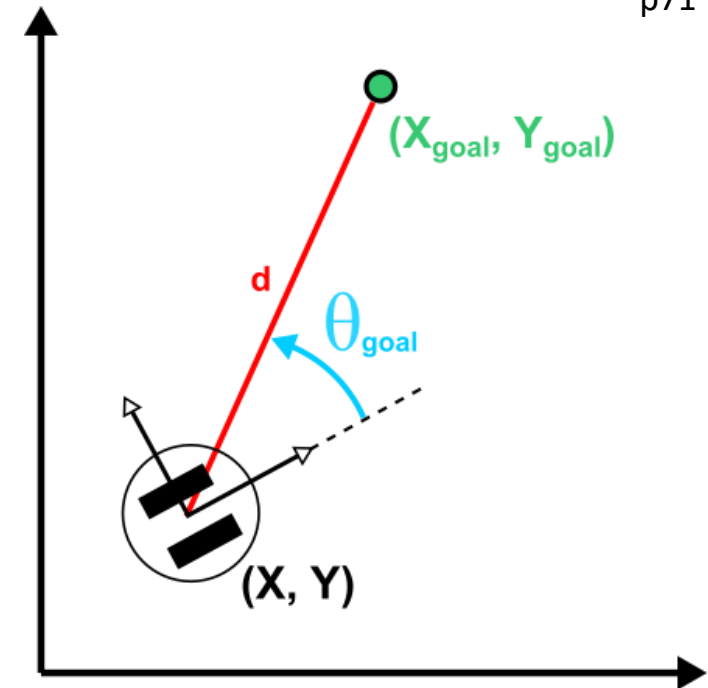
Moving to a Point (x,y) in the 2D plane

- Euclidean distance: $distance = \sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2}$
- Orientation: $\theta_{goal} = \text{atan2} \frac{(y_{goal} - y)}{(x_{goal} - x)}$
- Proportional Controller: Velocity $v = K_v \times distance$
 Steering angle $\gamma = K_h \times \theta_{goal}$

The robot's velocity is proportional to its distance to the goal



p71



Assignment

turtle_to_target (Python)

At least, answer the following questions:

- What are the turtlesim Pose and Twist message like?
- What should we import in the python header file to use Pose and Twist objects?
- 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 ?
- Explain what the `spawn_turtle` node is doing

Assignement

turtle_to_target (Python)

Download the files: *turtlesim_target.launch*
spawn_turtle.py
turtle_to_target.py



https://github.com/LucMarechal/ROS_Lectures/tree/master/Assignement_Lecture4

Place them in the right folders inside the *turtlesim_tutorials* package (give the locations)

Don't forget to source your catkin workspace !

Edit the *turtle_to_target.py* and fill the XXXX in the code

Edit the *turtlesim_target.launch* file so it launches :

- *turtlesim_node.py*
- *spawn_turtle.py*
- *turtlesim_target.py*

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

Further References

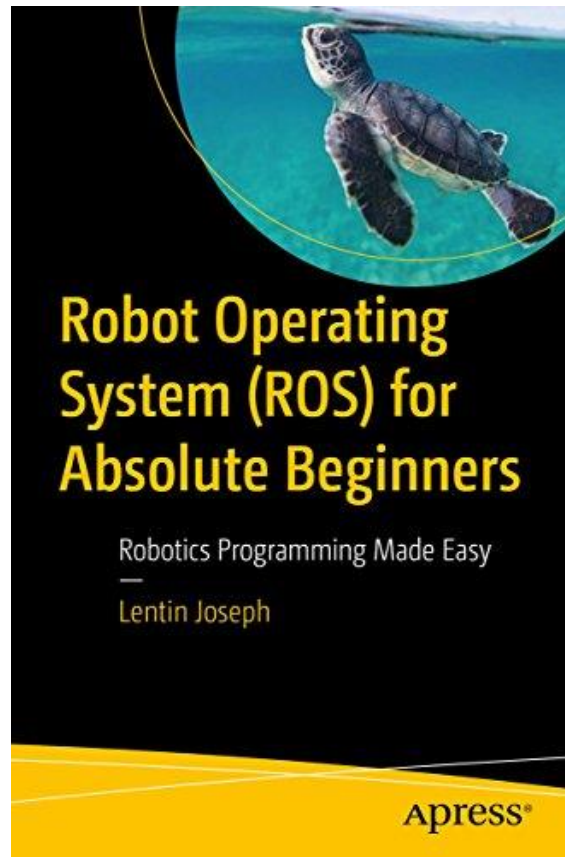
- **ROS Turtlesim tutorials**

- wiki.ros.org/turtlesim/Tutorials/

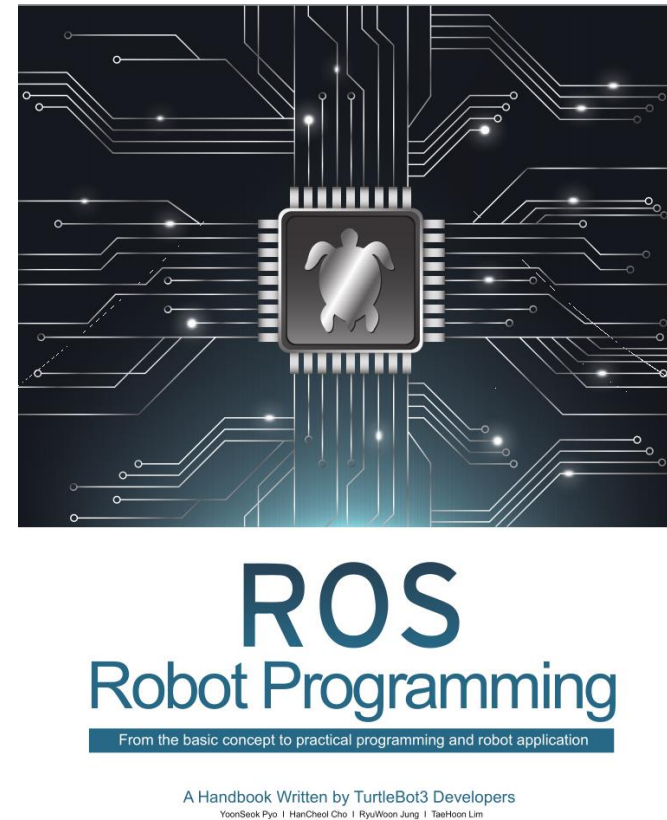
- **ROS Cheat Sheet**

- <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 Chambéry
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)



SYMME