

## TP2:

### Controlling a Turtle (bis)

All the ROS environment is already setup into a Docker container. To start it, enter the following command line:

```
./run.sh
```

Run the roscore and turtlesim node (see TP1).

#### Forward then turn

In this exercise you have to make the turtle do the following movements:

- go forward for 1 meter
- turn to reach  $\pi/2$  heading
- go forward for 2 meters
- turn to reach  $\pi$  heading
- go forward for 2 meters
- go forward for 2 meters
- turn to reach 0 heading
- go forward for 1 meter

The turtle current pose (x, y, heading) is saved in the variable `turtle_pose`. The idea here is to get familiar with publisher, so we do not require that the turtle follow a perfect square.

To reset the turtle position (if required) you can use the rosservice `/reset`:

```
rosservice call /reset
```

1. Define a subscriber that will get the pose of the turtle and update the global variable
2. Publish the correct cmd message to control the turtle (you can use multiple while loop)

```
import rospy
from turtlesim.msg import Pose  # import the turtlesim Pose message type
from geometry_msgs.msg import Twist  # import the Twist message type
import math

turtle_pose = Pose()
def pose_callback(data):
    """
    a callback to save the current turtle pose into the global variable turtle_pose
    """
    global turtle_pose
```

```

turtle_pose =      # <-- COMPLETE HERE

rospy.init_node( COMPLETE HERE )

#
turtle_pose_subscriber = rospy.Subscriber( COMPLETE_HERE )

cmd_publisher = rospy.Publisher(  COMPLETE HERE )
current_velocity_cmd = Twist()

rospy.sleep(0.2)
rospy.loginfo(turtle_pose)
init_turtle_pose = turtle_pose

# First movement
while turtle_pose.x  # <-- COMPLETE HERE
    # COMPLETE HERE (compute new command based on t)

# Second movement

# Third movements

# etc..

```

## Go To (x,y)

Now you have to move the turtle to a position (X, Y).

### Look then Move To

The easiest way to go to a position is to - rotate the robot to look to the target point - move to the point following a straight line

**Define a function that will turn the robot until a specific angle is reached**

You can use the following code:

```

def turn_to(robot_theta, target_theta, cmd_publisher):
    cmd_vel = Twist()
    # COMPLETE HERE : compute the right command

```

```

# send the command to the robot
cmd_publisher.publish( COMPLETE_HERE )

```

Define a function that will move the robot on a straight line based on a distance

```

def move_to(distance, cmd_publisher):
    cmd_vel = Twist()
    # COMPLETE HERE : compute the right velocity command

    # send the command to the robot
    cmd_publisher.publish( COMPLETE_HERE )

```

Use the two functions

Complete the following code to use the two previous functions.

```

import rospy
from turtlesim.msg import Pose # import the turtlesim Pose message type
from geometry_msgs.msg import Twist # import the Twist message type
import math

turtle_pose = Pose()
robot_current_orientation = 0

def pose_callback(data):
    """
    a callback to save the current turtle pose into the global variable turtle_pose
    """
    global turtle_pose
    # COMPLETE HERE

rospy.init_node("TurtleEx4")
turtle_pose_subscriber = rospy.Subscriber( COMPLETE_HERE )
cmd_publisher = rospy.Publisher( COMPLETE_HERE )

t = Twist()

target_X = #<--COMPLETE HERE
target_Y = # <-- Complte Here
rospy.sleep(0.2)
rospy.loginfo(turtle_pose)
init_turtle_pose = turtle_pose

target_theta = # Complete here (simple trygonometry)

```

```

while not ( COMPLETE HERE ): # <-- you need to check that the robot has not yet reached the
    turn_to( COMPLETE HERE)

while not (COMPLETE HERE): # <-- you need to check that the robot has not yet reached the
    move_to( COMPLETE HERE)

rospy.loginfo("destination reached !!!! YEAH !!")

```

### Turn and move at the same time

Now you have to write a go to function that allows the turtle to go to a destination (X,Y) but doing the move and rotation at the same time.

You can use the following control law:

$$v = k_v * \sqrt{(X - cur_x)^2 + (Y - cur_y)^2}$$

$$\theta_{target} = \tan^{-1}(y - cur_y)/(x - cur_x)$$

$$\theta_{velocity} = k_t * (\theta_{target} - \theta_{robot})$$

You can use different value for kv and kt, e.g kv=0.1, kt=1. You could also limit the min velocity that you send to the turtle, for instance a velocity bellow 0.1 meter/sec is maybe too slow.

For more information see : <https://www.youtube.com/watch?v=Qh15Nol5htM&feature=youtu.be&list=PLSzYQG CXRW1HLWHdJ7ehZPA-nn7R9UKPa>

```

import math

#####
#COMPLETE HERE

#####

move_to(1, 1)

```

## ROS TOOLS (RVIZ and Gazebo)

Launch the turtle bot gazebo with corridor world:

```

TURTLEBOT_GAZEBO_WORLD_FILE=/opt/ros/kinetic/share/turtlebot_gazebo/worlds/corridor.world
roslaunch turtlebot_gazebo turtlebot_world.launch

```

Start rviz and visualize the robot and it's sensors.