AUTONOMOUS TURTLE

Assignment 2

F1/10 Autonomous Racing – CS 4501/SYS4582 – Spring 2019

Due Wed, 27rd February, at 2:00pm, before the lecture

Madhur Behl
(madhur.behl@virginia.edu)

In this assignment, you will create an ~~self driving~~ ~~self swimming~~ autonomous turtle using the ROS turtlesim simulator.

The goal of the assignment is to understand:

- ROS nodes
- ROS topics
- ROS messages
- ROS parameters and services
- ROS launch files

And get familiar with:

- ROS workspace
- The catkin build system
- Creating your own ROS nodes.

To get started:

- Git pull to obtain the the autoturtle ROS package from https://github.com/linklab-uva/f1tenth-course-labs
- Link the autoturtle package to the catkin workspace
  - ln -s ~/github/f1tenth-course-labs/autoturtle/ ~/catkin_ws/src/
  - Run catkin_make in the catkin_ws folder. Source your environment.
- The node turtlesim_move.py has been provided for reference.
- Make sure you are able to run the provided node successfully before attempting the remaining assignment:

rosrun autoturtle turtlesim_move.py

[should move the turtle towards the x direction until it hits the wall]

Problem 1: Turtle Swim School

Problem 1a: swim_school.py

[20 points]

1. Using the turtlesim_move.py as reference, create a new ROS node called swim_school.py
2. Modify the code to demo the following:

[1a demo:] Upon running the command rosrun autoturtle swim_school.py

- User can input a linear (x) velocity
- User can input an angular (z) velocity
- The turtle then swims in a figure 8 shape, using the velocity and angular rate specified by the user.

Problem 1b

[20 points]

random_swim_school.py

1. Create a new ROS node random_swim_school.py

[1b demo:]

Upon running the command rosrun autoturtle random_swim_school.py

- A random figure 8 is drawn in the ocean:
  - Turtle moves with a random linear velocity and a random angular velocitywhich is chosen every time the node is run.
  - The figure 8 is drawn at a random position during each run. Its fine if the figure 8 hits the tsim boundary.
- The center position (x,y) of the random figure 8 is displayed on the terminal.

Problem 1c: back_to_square_one.py

[30 points]

1. Using the turtlesim_move.py as reference, create a new ROS node called back_to_square_one.py
2. Modify the code to demo the following:

1b demo: Upon running the command

rosrun autoturtle back_to_square_one.py

- User inputs the length of the side of the square (number between 1 and 5, any number including floating points is allowed.)
- The ocean turns **red**
- A square is drawn with the length of the side of the square equal to the user input.
  - Note the square must be drawn as the turle swims…do not use teleport absolute to draw the square.
- The bottom left corner of the square is always at (x,y) = (1,1), theta could be anything you want.

Problem 2: Swim to goal

[30 points]

Problem 2: swim_to_goal.py

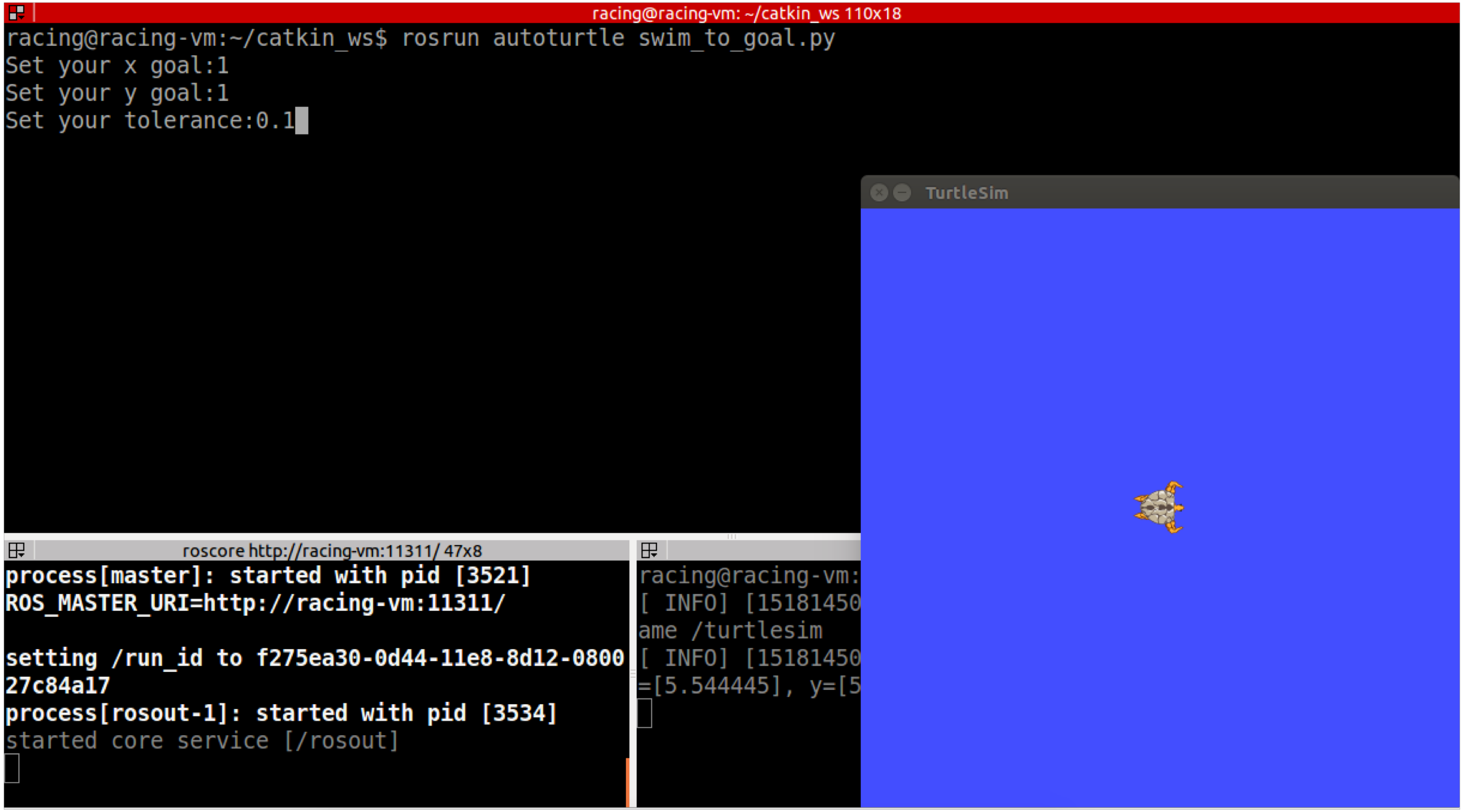
1. Create a new ROS node called swim_to_goal.py

Demo the following:

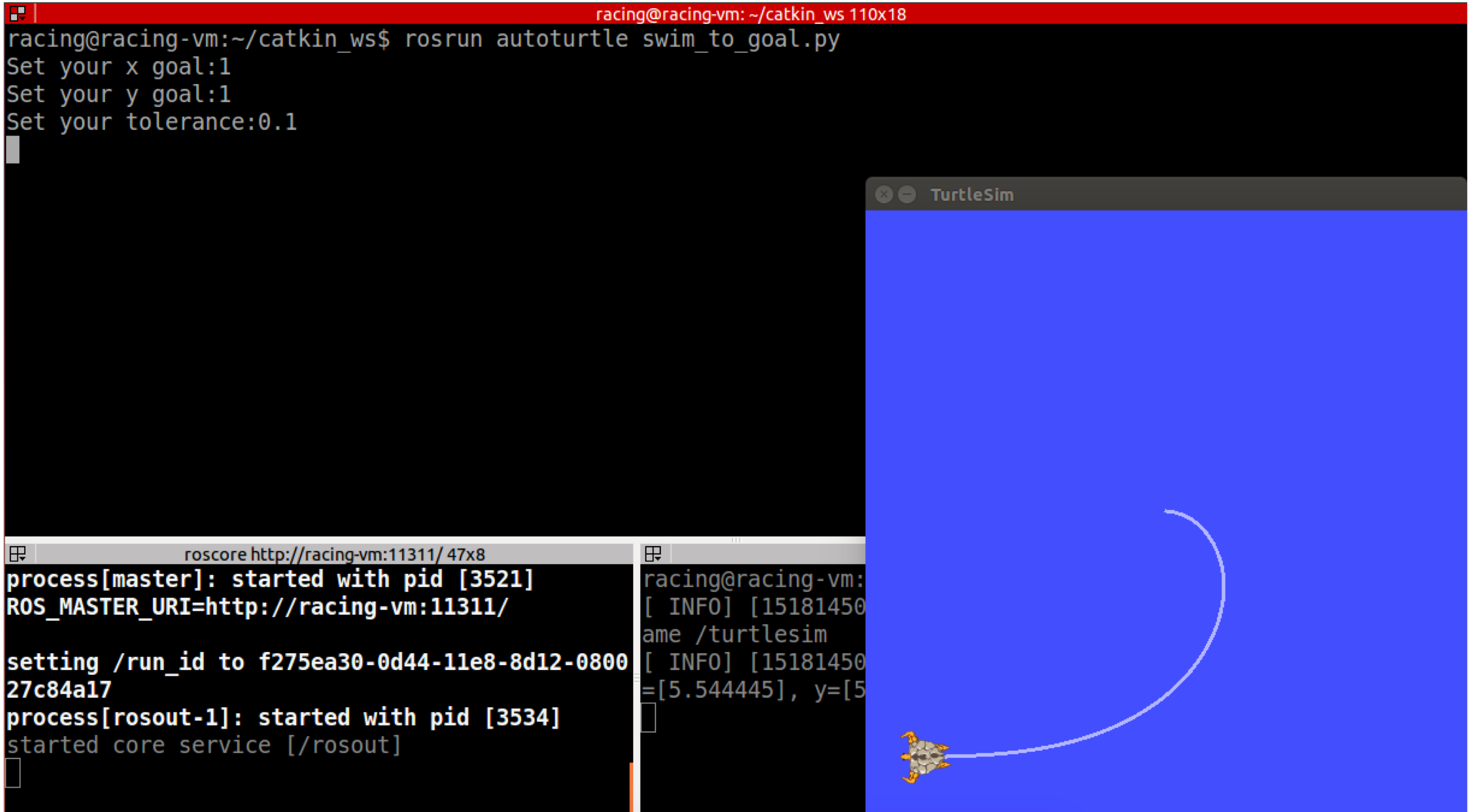
- User inputs the x coordinate for the goal: (e.g. 1)
- User inputs the y coordinate for the goal: (e.g. 1)
- User inputs a ‘tolerance’ value (defined in subsequent slides): (e.g. 0.1)
- The turtle swims (not teleports) to the goal with a velocity and angular rate, proportional to the distance between the current position and the goal.
- Display and explain the rqt_graph for this system

Problem 2: What should node execution look like ?

User enters x, y, goal, and a tolerance value.



Problem 2: What should the output look like ?



Problem 2: swim_to_goal.py - Hints

What does proportional velocity and angular command mean ?

Proportional speed and turning:

Velocity x = **1.5 *** (Euclidean distance between position at any time and the goal)

Angular z = **4 *** (atan2 (Euclidean distance between position at any time and the goal))

The 1.5 and 4 are constants but the velocity and angular rate will change as the turtle moves towards the goal, since the distance between the position of the turtle and the goal is always decreasing. **You should use 1.5 and 4 as the constants in your code**.

Tolerance is a constant which dictates how close does the turtle need to be to the (x,y) goal before it is considered that it has reached the goal.

The tolerance is needed since the pose messages that turtlesim echoes is usually a floating point value.

A good value for the tolerance is between 0.1 and 1.

A useful command in python to use for dealing with pose data is:

round(number[, ndigits])

Useful declarations for this problem are:

```
import rospy
from geometry_msgs.msg  import Twist
from turtlesim.msg import Pose
from math import pow, atan2,sqrt
```

You will demo your code in the lab, but in addition to that you need to submit all your code by the assignment deadline.

What do I have to submit ?

- All the ROS nodes you create should be inside the /autoturtle/scripts directory. (The same as the turtlesim_move.py script).
- Compress the entire /autoturtle folder.
- **Rename the compressed file to <your_computing_ID_ROS_distributionName>.zip and submit on Collab.**
  - **Only submit a single zip file.**
- Ensure, the code you are submitting does not throw any errors during a catkin_make.

- We will un-compress your submitted folder inside the src folder of our catkin workspace and run catkin_make.

- We will then test your code with the following commands:
  - rosrun autoturtle swim_school.py
  - rosrun autoturtle random_swim_school.py
  - rosrun autoturtle back_to_square_one.py
  - rosrun autoturtle swim_to_goal.py

Extra Credit >> Upto 40 points

There are 4 problems:

For each problem, you can earn 10 points for Extra Credit, if you can create and use a launch file to demonstrate that problem (i.e. you create the appropriate node, but use a launch file for each problem to launch the relevant nodes for that problem).