

Autonomous Turtle

Assignment 1

F1/10 Autonomous Racing – CS 4501– Spring 2021

Due Thu, 25th March, by 11:59pm [Firm Deadline]

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 (or copy) the autoturtle package to the catkin workspace
 - In -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 can run the provided node successfully before attempting the remaining assignment:

`roslaunch autoturtle turtlesim_move.py`

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



Problem 1: Turtle Swim School

[20 points]

Problem 1a: swim_school.py

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 `roslaunch autoturtle swim_school.py`

- User can input a linear (x) velocity in the range [2,6] (floating point allowed)
- User can input an angular (z) velocity [1,3] (floating point allowed)
- 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:]

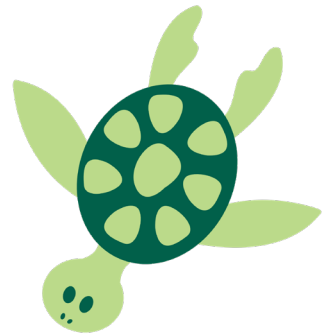
Upon running the command `roslaunch autoturtle random_swim_school.py`

- A random figure 8 is drawn in the ocean:
 - Turtle moves with a random linear velocity and a random angular velocity which is chosen every time the node is run. Use the same input range as problem 1a.
 - In addition, the figure 8 is drawn at a random position during each run. Its fine if the figure 8 hits the turtlesim boundary.
- The center position (x,y) of the random figure 8 is displayed on the terminal.

[30 points]

Problem 1c: back_to_square_one.py

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

`roslaunch 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 turtle 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`

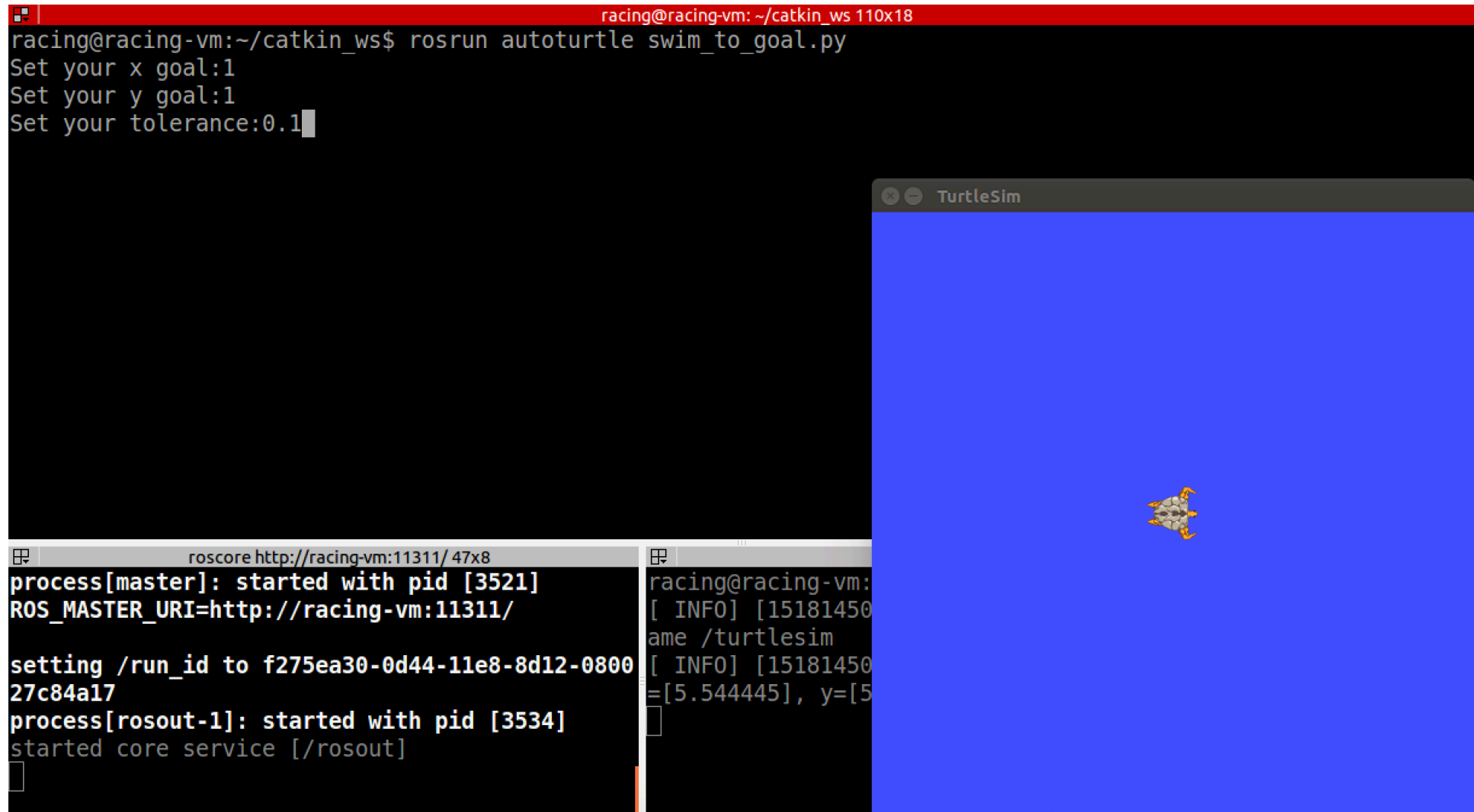
1. Create a new ROS node called `swim_to_goal.py`

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.



The screenshot displays a ROS environment with two windows. The top window is a terminal titled 'racing@racing-vm: ~/catkin_ws 110x18'. It shows the command 'roslaunch autoturtle swim_to_goal.py' being executed. The terminal prompts the user to 'Set your x goal:1', 'Set your y goal:1', and 'Set your tolerance:0.1'. The bottom window is a 'TurtleSim' window, which is a blue square representing a 2D environment. A small turtle icon is visible in the center of the blue area. The bottom-left corner of the screenshot shows a 'roscore' window with the following output: 'process[roscore]: started with pid [3521]', 'ROS_MASTER_URI=http://racing-vm:11311/', 'setting /run_id to f275ea30-0d44-11e8-8d12-080027c84a17', 'process[roscore-1]: started with pid [3534]', and 'started core service [/roscout]'. The bottom-right corner shows a terminal window with the following output: 'racing@racing-vm: [INFO] [15181450] name /turtlesim', '[INFO] [15181450] x=[5.544445], y=[5.544445]', and a cursor.

```
racing@racing-vm: ~/catkin_ws 110x18
racing@racing-vm:~/catkin_ws$ roslaunch autoturtle swim_to_goal.py
Set your x goal:1
Set your y goal:1
Set your tolerance:0.1

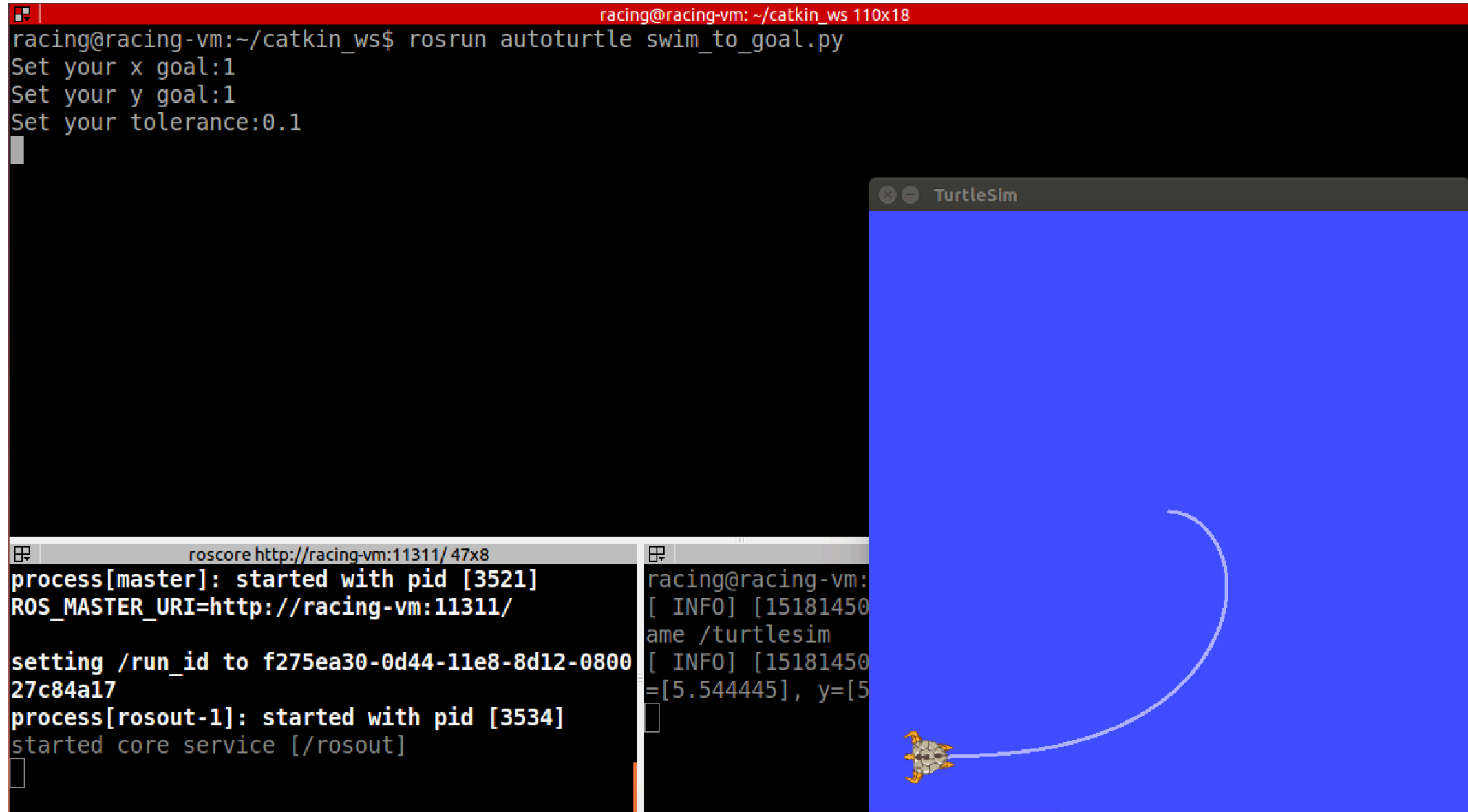
racing@racing-vm:~/catkin_ws$

roscore http://racing-vm:11311/ 47x8
process[roscore]: started with pid [3521]
ROS_MASTER_URI=http://racing-vm:11311/

setting /run_id to f275ea30-0d44-11e8-8d12-080027c84a17
process[roscore-1]: started with pid [3534]
started core service [/roscout]

racing@racing-vm: [ INFO ] [15181450] name /turtlesim
[ INFO ] [15181450] x=[5.544445], y=[5.544445]
```

Problem 2: What should the output look like ?



The screenshot displays a ROS environment with three windows. The top window is a terminal titled 'racing@racing-vm: ~/catkin_ws 110x18'. It shows the command 'roslaunch autoturtle swim_to_goal.py' being executed, followed by prompts for 'Set your x goal:1', 'Set your y goal:1', and 'Set your tolerance:0.1'. The bottom-left window is a terminal titled 'roscore http://racing-vm:11311/ 47x8', showing the master node starting with pid [3521] and the ROS_MASTER_URI set to 'http://racing-vm:11311/'. The bottom-right window is a terminal titled 'racing@racing-vm:', showing the command 'roslaunch turtlesim turtlesim' being executed, followed by log messages indicating the launch of the turtlesim package. The rightmost window is a 'TurtleSim' window with a blue background. It shows a small turtle icon at the bottom left, with a white curved line representing its trajectory moving towards the top right.

```
racing@racing-vm:~/catkin_ws 110x18
racing@racing-vm:~/catkin_ws$ roslaunch autoturtle swim_to_goal.py
Set your x goal:1
Set your y goal:1
Set your tolerance:0.1

roscore http://racing-vm:11311/ 47x8
process[master]: started with pid [3521]
ROS_MASTER_URI=http://racing-vm:11311/

setting /run_id to f275ea30-0d44-11e8-8d12-080027c84a17
process[roscout-1]: started with pid [3534]
started core service [/roscout]

racing@racing-vm:
[ INFO] [15181450]: started with pid [15181450]
ame /turtlesim
[ INFO] [15181450]: started with pid [15181450]
=[5.544445], y=[5.544445]
```

Problem 2: swim_to_goal.py - Hints

What does proportional velocity and angular command mean ?

Proportional speed and turning:

Velocity $x = 1.5 * (\text{Euclidean distance between position at any time and the goal})$

Angular $z = 4 * (\text{atan2 (ratio of y component to x component)})$

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 can 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
```

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>.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`.
- **Video demo submission:** In addition, you need to record the demo for each problem that you submit. Use any screen recording software (such as `recordmydesktop`, or `OBS`) to share a **single video** for the entire submission.
- **You should include a PDF file in the uploaded zip folder which contains the link where the instructors can view your video (e.g. it could be an unlisted video on YouTube or Gdrive etc.). In the PDF you can also include the `rqt_plot` for Problem 2 along with a brief explanation for it.**

- 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 and correct input ranges:
 - rosrun autoturtle swim_school.py
 - rosrun autoturtle random_swim_school.py
 - rosrun autoturtle back_to_square_one.py
 - rosrun autoturtle swim_to_goal.py

Launch file for each problem.

There are 4 problems in this assignment:

For each problem, you can earn 10 additional points, 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).

These files should be included in the autoturtle/launch subdirectory in your submission