ROS Assignment 1 Report

Scripts Breakdown

1. **moveturtle.sh**
   Steps:
   1. Kills the default turtle with the ‘kill’ service to reposition it, which ensures the first name initial is at the centre wrt the turtlesim window.
   2. Spawns new turtle at appropriate position & orientation utilising the ‘spawn’ service.
   3. Publishes consecutive stream of ‘geometry_msgs/Twist’ type messages to the ‘turtle1/cmd_vel’ topic that guide the turtle in manoeuvring the letter ‘R’. (The specific values of linear and angular velocities were decided upon after some trial & error.)

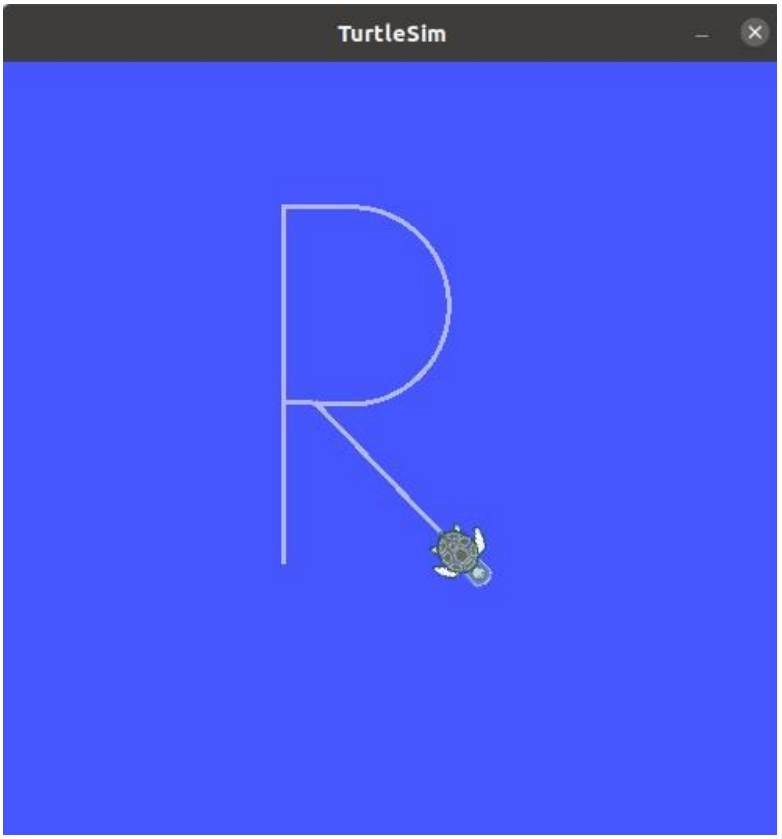
Topics	Services	Messages
`turtleX/cmd_vel`	`kill`	`geometry_msgs/Twist`
	`spawn`	

2. **movetwoturtles.sh**
   Steps:
   1. Kills the default turtle with the ‘kill’ service.
   2. Changes background colour of window to green utilising parameters- ‘/turtlesim/background_g’ & ‘/turtlesim/background_b’.
   3. Calls the ‘clear’ service for the background change to take effect.
   4. Spawns two new turtles at appropriate positions & orientations utilising the ‘spawn’ service.
   5. Calls the ‘/turtleX/set_pen’ service to change the colour of the turtle trails.
   6. Publishes consecutive stream of ‘geometry_msgs/Twist’ type messages to the ‘turtle1/cmd_vel’ topic that guide turtle1 in manoeuvring the letter ‘R’.
   7. Publishes consecutive stream of ‘geometry_msgs/Twist’ type messages to the ‘turtle2/cmd_vel’ topic that guide turtle2 in manoeuvring the letter ‘S’. (The specific values of linear and angular velocities were decided upon after some trial & error.)

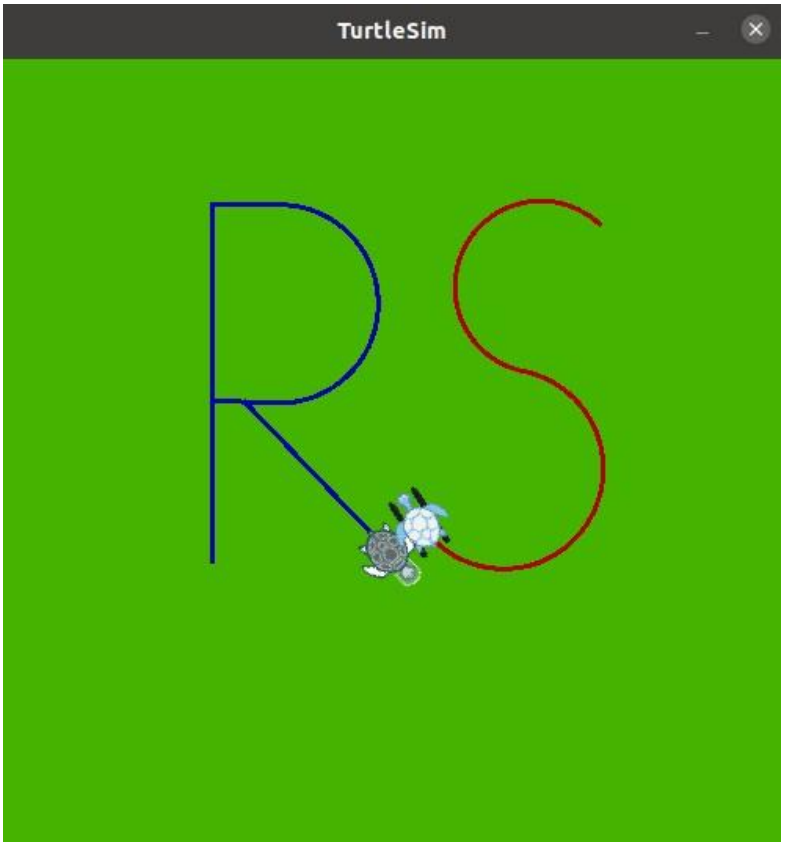
Topics	Services	Messages	Parameters
`turtleX/cmd_vel`	`kill`	`geometry_msgs/Twist`	`/turtlesim/background_g`
	`clear`		`/turtlesim/background_b`
	`spawn`		
	`turtleX/set_pen`		

Performance Testing

1. moveturtle.sh



2. movetwoturtles.sh



How can one implement the Publisher-Subscriber functionality of the hello world example using a Service and Client? What would be the structure of the .srv file?

The central idea of the ‘Publisher-Subscriber functionality’ is to segregate and modularize the process of data generation and consumption. So long as the publisher generates the data and makes it available on the topic, the subscriber can consume it.

The Service and Client framework can follow a similar workflow in essence. So long as the client calls the service(with or without request arguments), it should receive the appropriate response. It’s just that we no longer need a mediator for the transfer of data.

In the case of the ‘Hello World’ example, the Publisher publishes the ‘hello world’ string along with the current timestamp to the ‘chatter’ topic, which is subscribed by the Subscriber. The Publisher here has everything it needs to generate data to publish to the topic. So, to convert this to a Service-Client framework, we see that the .srv file that describes a service need not have the ‘request’ part but just the ‘response’ part, which is a string.

The .srv file in this case should be simply:

```
      # Get the return value as a string
      ---
      string return_str
```