LAB 2: Report

Team 1

Clementine Mitchell, Martina Stadler, Nick Villanueva, Samir Wadhwania, Jake Liguori, Jose Gomez

Overview - Martina

The objective of this lab was to familiarize participants with the MIT Racecar setup in preparation for future labs. Each student gained access to a Linux platform, downloaded and installed existing racecar software (either onto their Virtual Machine or natively onto their Linux operating system) and ran short tests to confirm the functionality of their setup. The team also created GitHub accounts and pages to enable future collaboration, both with their fellow students and the ROS community at large.

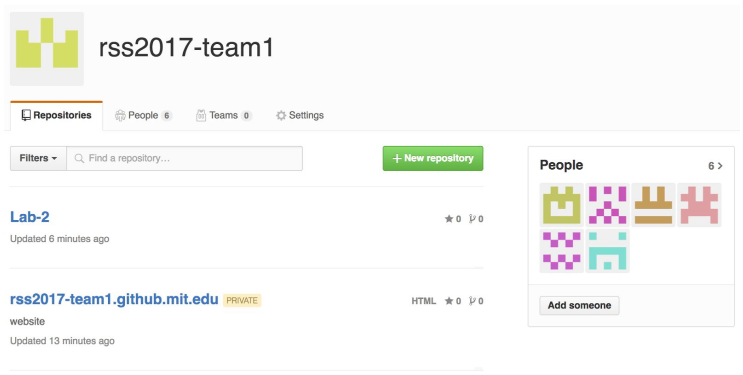
After the initial setup phase, the team received its race car, which was already fitted with sensors, and tested its ability to read and record the car's sensor data.

Approaches:

In this lab, students were not responsible for writing their own programs. Rather, they used existing modules to ensure that they had a complete setup, enabling future development.

Computer & GitHub Setup - Samir

The first part of the lab consisted of setting up our environment within the VM to properly interact with the robot and maintain consistency throughout the labs. The first step was to create an access token and configure our GitHub accounts for use in the terminal. The next step was to create an organization - **rss2017-team1** - to store our repositories for our website and our labs. Finally, I created a repository for this lab - **Lab 2** - to upload our code and pertinent files.



Lab-2
Updated 6 minutes ago
★ 0 0

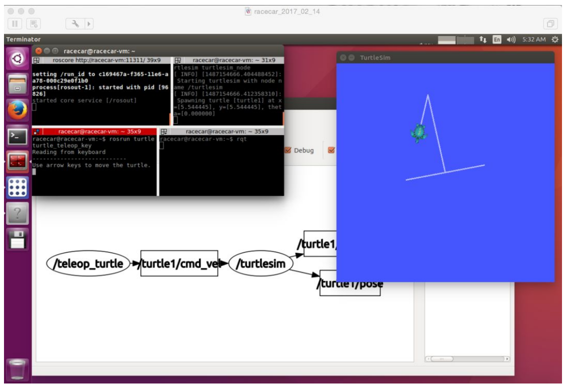
rss2017-team1.github.mit.edu PRIVATE
website
Updated 13 minutes ago
HTML ★ 0 0

Figure 1: Team Organization in GitHub

Website - Jose
A website to document our progress and the development of our software was also developed as part of this lab. The website was built using an open source bootstrap template and is published through GitHub. Currently the website has three main pages (the Homepage, an About page, and a page for Lab 2) that can be accessed through the navigation bar links on the top of the page. The most relevant pages for this particular lab are the about page, which contains team bios and introduction, and the Lab 2 page, which contains the Lab 2 report. From this lab forward the website will be updated weekly as the team progresses through the class. The website can be found in url below.

Website url: https://github.mit.edu/pages/rss2017-team1/

Running the Simulation - Jake
In the first section of this lab we learned the basics of Linux and how to interact with the ROS software from the command line. This was done using a simple robot simulator called the TurtleSim Node. We were able to control the turtle in this simulation using both the arrow keys and message commands issued at the command line. To visualize the computation graph, we ran the *rqt* graphical user interface (Figure 1). From here we were able to gain experience analyzing the nodes, topics and messages within this specific simulation, though all node graphs generated by ROS will have a similar format. These analysis tools were necessary when running simulations in the Racecar simulator, which we ran in this lab and will run in future labs as well.



```
setting /run_id to c169467a-f365-11e6-a
a78-000c29e0f1b0
process[rosout-1]: started with pid [96
826]
started core service [/rosout]
```

```
rtlesim turtlesim_node
[ INFO] [1487154666.404488452]:
 Starting turtlesim with node n
ame /turtlesim
[ INFO] [1487154666.412358310]:
 Spawning turtle [turtle1] at x
=[5.544445], y=[5.544445], thet
a=[0.000000]
```

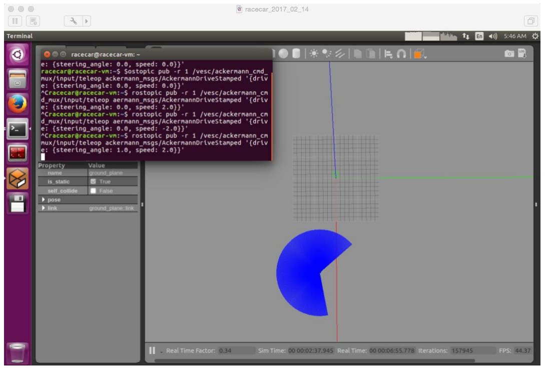
```
racecar@racecar-vm:~$ rosrun turtle
turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

```
racecar@racecar-vm:~$ rqt
```

Figure 2: TurtleSim Practice

a. **Open-Loop Control**

Each member of the team set up his or her computer to use the racecar simulator by forking and cloning the simulator, as well as building the team workspace, as explained above. Each computer now has the capacity to separately run the simulator and send commands to the racecar and watch it move about a three dimensional surface (though it always remains in the same plane). Commands are sent using *rostopic pub,* in the same way that we did in the TurtleSim node. We were able to alter both steering angle and speed to have the simulated car follow different paths, as seen in Figure 2.



```
e: {steering_angle: 0.0, speed: 0.0}}'
racecar@racecar-vm:~$ $ostopic pub -r 1 /vesc/ackermann_cmd_
mux/input/teleop ackermann_msgs/AckermannDriveStamped '{driv
e: {steering_angle: 0.0, speed: 0.0}}'
^Cracecar@racecar-vm:~$ rostopic pub -r 1 /vesc/ackermann_cm
d_mux/input/teleop aermann_msgs/AckermannDriveStamped '{driv
e: {steering_angle: 0.0, speed: 2.0}}'
^Cracecar@racecar-vm:~$ rostopic pub -r 1 /vesc/ackermann_cm
d_mux/input/teleop aermann_msgs/AckermannDriveStamped '{driv
e: {steering_angle: 0.0, speed: -2.0}}'
^Cracecar@racecar-vm:~$ rostopic pub -r 1 /vesc/ackermann_cm
mux/input/teleop ackermann_msgs/AckermannDriveStamped '{driv
e: {steering_angle: 1.0, speed: 2.0}}'
```

Figure 3: Open-Loop Control of Racecar Model

b. Open-Loop Control with Joypad - Clementine

The team initiated a new instance of the racecar simulation, which represented MIT's tunnel system, in order to control it with the joypad. The joypad receiver was connected to one team member's Virtual Machine via the USB port on their laptop. We wrote a launch file to remap the topic `/ackermann_cmd_mux/input/teleop` to the topic `/vesc/ackermann_cmd_mux/input/teleop` , which allowed the racecar to move when the joypad was in use. The code we used for the remapping of the joypad looked as follows:

```
<remap from="/ackermann_cmd_mux/input/teleop"  to="/vesc/ackermann_cmd_mux/input/teleop" />
<include file="$(find racecar_control)/launch/teleop.launch" />
```

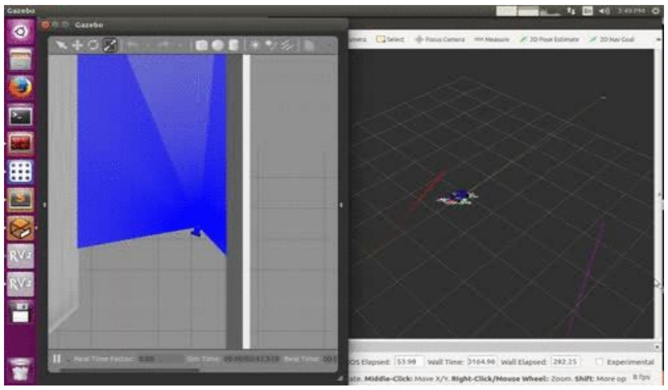


Figure 4: Simulation when the car was being controlled with joypad

Once the team confirmed that joypad use corresponded with the motion of the racecar in the simulator, we tested it out on the actual racecar hardware by inserting the joypad receiver into the back of the racecar instead of the VM. We were able to see the movement of the wheels of the racecar when we used the joypad, thus confirming the functionality of this component of our system.

Collecting and Storing Sensor Data - Nick
Once we became familiar with the racecar simulations, we moved on to collecting data from the actual car. After booting up the router and the onboard Jetson computer on the racecar, we were able test our connection to the car and then ssh into it. We were then able to visually stream data from both the Zed camera and laserscanner. The racecar streamed the images from the forward facing Zed camera to our laptops. Figure 3 shows an image captured by the camera.

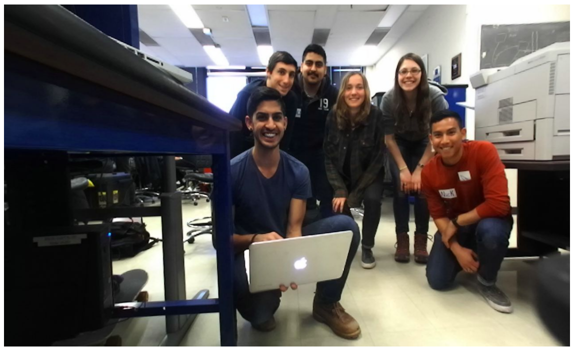
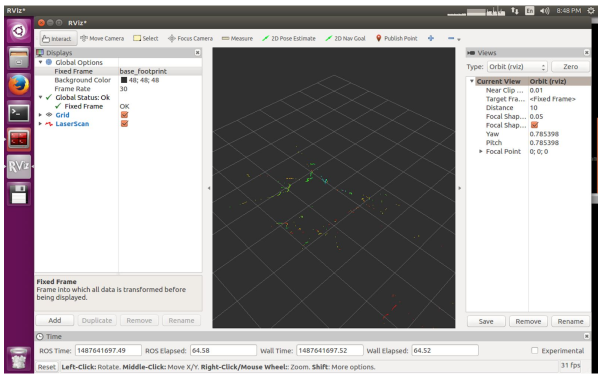


Figure 5: Zed Image

We were also able to stream data from the laserscanner mounted on the racecar. The laserscanner shows objects in the vicinity of the car represented as dots. However, it only detects objects in a single plane. Figure 4 shows a snapshot of the data sent back from the scanner.



Global Options
Fixed Frame: base_footprint
Background Color: 48; 48; 48
Frame Rate: 30
Global Status: Ok
Fixed Frame: OK
Grid: ✓
LaserScan: ✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0.01
Target Fra...: <Fixed Frame>
Distance: 10
Focal Shap...: 0.05
Focal Shap...: ✓
Yaw: 0.785398
Pitch: 0.785398
Focal Point: 0; 0; 0

ROS Time: 1487641697.49 ROS Elapsed: 64.58 Wall Time: 1487641697.52 Wall Elapsed: 64.52 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options. 31 fps

Figure 6: Laserscanner snapshot

In addition to being able to stream data from the racecar sensors, we were also able to use the wireless joystick to drive the car. Lastly, we downloaded the collected sensor data from the racecar.

Important Parts of our Code:
The important parts of our existing code are for:

i. Running the simulation
ii. Creating a launch file
iii. Streaming Data
iv. Collecting Data

For source code, see the team GitHub.

Lessons Learned:
Description about what we learned about working as a team:

Samir: Ensuring that everyone is able to participate can be difficult. We can definitely work together to make sure everyone understands how tasks are being divided, what is currently being done, and how everything is working together.

Martina: Finding a time where everyone can meet is challenging. Clarifying tasks and defining specific goals maximizes productivity when the team cannot get together.

Jake: Working in parallel can maximize productivity. Too many people working on the same piece of an assignment at once can lead to repetitiveness.

Jose: Clear work distribution or overall communication is quite important and something our team will need to work on.

Nick: Communication is key. Early in the project we set up a slack account which has helped us stay in contact with each other. Since then we have been able to keep each other up to date on where we were on setting up our computers and when we were available to plan meetings.

Clementine: Once we had clearly defined individual tasks, the work got done much more quickly. As we grew closer as a team, we also became more confident in asking each other for help, which allowed for greater efficiency.