CS 789 Autonomous Racing:
Lab 1 Emergency Braking

Section TBD, Fall 2023

---

1 Lab Description

In this lab, groups will (1) create a ROS environment to (2) run the F1TENTH simulator that performs (3) emergency braking and (4) verify the behavior on a vehicle.

2 Learning Outcomes

Outcome 1 A functioning ROS 2 environment running within a docker container

Outcome 2 A functioning F1TENTH simulator running in the docker container

Outcome 3 An understanding of emergency braking and implementation within the simulator

Outcome 4 Practical demonstration of emergency braking on a vehicle

3 Rubric

In Simulator Demonstration	40
On Vehicle Demonstration	40
Package Deliverable	20
Total	100

4 Submission Instructions

A completed project is first demonstrated in the simulator and then upon the vehicle. Once demonstrated, the `safety_node` package is submitted via Canvas.

5 Laboratory Description

The first portion of the laboratory description provides instructions for connecting and preparing a ROS environment on the machine provided for this course. The ROS (and therefore) F1TENTH environment works **exclusively** on Ubuntu 20.04. While groups are encouraged to prepare their own environment on a personal machine, support for doing so will not be provided, and any faults with a personal environment will not be a contributing factor for extensions or leniency.

The second portion is an exercise for the groups to implement the automatic emergency braking node within the F1TENTH Gym Simulator and then deploy it to the vehicle.

5.1 Background

Similar to the vehicle students will be writing applications for, the software system to support those applications is built upon several distinct technologies. Students must develop an understanding of the relationship between those supporting technologies and the vehicle to succeed.

The vehicle software runs atop the robotic operating system (ROS). Although ROS has operating system embedded within the acronym, it is not an operating system. ROS is an application framework that runs atop another operating system Linux, specifically Ubuntu 20.04. The vehicle runs Ubuntu 20.04, ROS, and the F1TENTH packages necessary for operation.

To aid development, ROS and F1TENTH packages provide docker containers to allow testing via simulation without a vehicle. The complexity of running a docker container for Ubuntu 20.04 upon a host with a differing distribution (or version) is an additional task avoided in this course. Students may attempt to do so, however the instructions and support from the instructor and teaching assistant will be limited to Ubuntu 20.04 running docker containers without a GPU.

5.2 Virtual Machine

Each group has been provided a virtual machine with accounts for them to log into. Every student should have already received their credentials and machine name. If they have not, please contact the instructor immediately.

The instructions use the example username `gordon` and virtual machine `a301-f1sim.cs.unlv.edu`. Using SSH connect to the machine with the given credentials. Be aware, this virtual machine is only available on the campus network. If you are connecting to the campus network via eduroam you will need the Forticlient VPN available from `https://www.it.unlv.edu/vpn`.

```
$ ssh gordon@a301-f1sim.cs.unlv.edu
gordon@a301-f1sim.cs.unlv.edu's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-69-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

2 updates can be applied immediately.
2 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

54 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '22.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Wed Apr 12 12:28:11 2023 from 131.216.17.116
gordon@f1sim:~$
```

Immediately following your first login, change your password.

```
gordon@f1sim:~$ passwd
Changing password for gordon.
Current password:

New password:

```

```
Retype new password:

passwd: password updated successfully
```

You may choose to use VNC to connect to the machine. However, you will need to start a userspace VNC server and identify the port assigned to your instance. Before connecting via VNC, you will need to configure the userspace server with the following commands:

Step 1 – Set a VNC password (which will also create the necessary directories).

```
gordon@f1sim:~$ vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? n
```

Set a VNC password

Step 2 – Create an X11 session (a desktop manager)

```
gordon@f1sim:~$ cat > .vnc/xstartup
#!/bin/bash

xrdb $HOME/.Xresources
startxfce4
# Hit CTRL-D to end
gordon@f1sim:~$ chmod +x .vnc/xstartup
```

Creating an X11 Session

Step 3 – Start a vncserver (please limit yourself to one)

```
gordon@f1sim:~$ vncserver -localhost no

New 'f1sim:2 (gordon)' desktop at :2 on machine f1sim

Starting applications specified in /home/gordon/.vnc/xstartup
Log file is /home/gordon/.vnc/f1sim:2.log

Use xtigervncviewer -SecurityTypes VncAuth,TLSVnc -passwd
/home/gordon/.vnc/passwd f1sim:2 to connect to the VNC server.
```

Starting a VNC server

Note the ":2" is highlighted in the vncserver output, this informs the user of the port the vncserver is running on. Add the number following the colon to 5900 to get the port of the vncserver, e.g. 5902. To connect to the vncserver, use your favorite client to connect to `MACHINE:PORT`.

If you have not installed a VNC client locally, the suggested client is TigerVNC. To install a client on Ubuntu use `apt install tigervnc-viewer`. For windows, download the client from https://tigervnc.org/

```
  # Using vncviewer from gordon's personal laptop
  $ vncviewer a301-f1sim.cs.unlv.edu:5902
```

After providing the password set by vncpasswd, a desktop should be visible similar to the one shown in Figure 5.2.

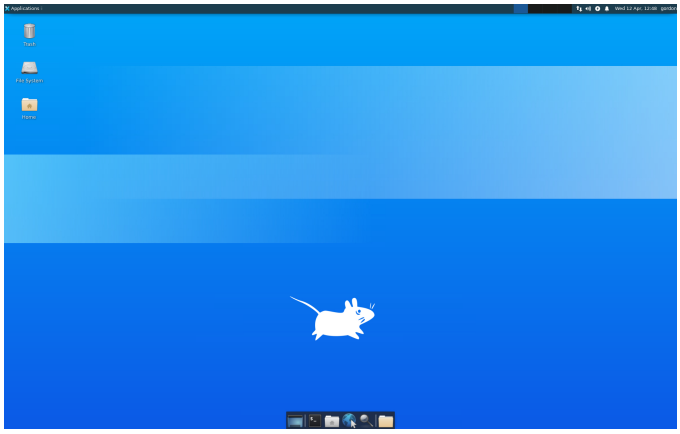

Figure 1: VNC Console for gordon

Please limit yourself to a single vncserver, to see how many servers you have running issue the following command:

```
gordon@f1sim:~$ vncserver -list

TigerVNC server sessions:

X DISPLAY #     RFB PORT #      PROCESS ID
:2              5902            51072
:3              5903            51299
:4              5904            51373
```

The output for `gordon` shows that they are running three, rather than one server. To stop the other two issue the vncserver -kill command with the display number.

```
gordon@f1sim:~$ vncserver -kill :3
Killing Xtigervnc process ID 51299... success!
gordon@f1sim:~$ vncserver -kill :4
Killing Xtigervnc process ID 51373... success!
```

Afterwards, there is only one server running:

```
gordon@f1sim:~$ vncserver -list

TigerVNC server sessions:

X DISPLAY #     RFB PORT #      PROCESS ID
:2              5902            51072
```

At this point, you should be able to log into the machine provided to your group and if you have chosen to use VNC, be able to interact with the desktop. If you are unable to log in stop, immediately, and troubleshoot the issue to obtain connectivity.

5.3 F1TENTH Gym and Simulator

With a machine running a docker server, either the provided virtual machine or an Ubuntu 20.04 of your own, the next step is to install the F1TENTH and simulator within a docker container.

To begin, clone the FT1TENTH gym in a new working directory.

```
    gordon@f1sim:~$ mkdir -p ws/gym-one
    gordon@f1sim:~$ cd ws/gym-one/
    gordon@f1sim:~/ws/gym-one$ git clone https://github.com/f1tenth/f1tenth_gym_ros
```

The next steps are a concert of operations, using docker compose two containers will be started. One of the containers allows the simulator to be started, the other container exports the display. To clarify, the

container exporting the display is visible only through a novnc web connection. Figure 5.3 illustrates how `gordon` may interact with the display after starting both containers.

Given the co-running desktop environments (Wayland and XFCE) the Terminal icon may not open a terminal automatically. If clicking on the terminal icon at the bottom of the screen does not work, navigate to `Applications -> System -> Xfce Terminal`.

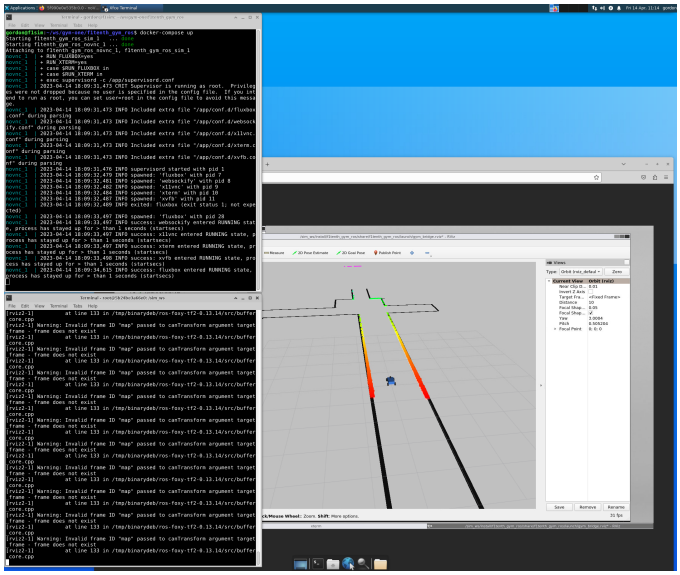

Figure 2: VNC Console for gordon

The first operation to get the simulator display running is to start the containers using `docker-compose`. In Figure 5.3, the upper left terminal shows the output of the command before any other steps are taken.

```
# starting the containers
gordon@f1sim:~$ cd ws/gym-one/f1tenth_gym_ros/
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros$ docker-compose up
```

Starting the Containers

With the containers running, the next step is to enter the container where the simulator may be started.

```
# First command enters the container
gordon@f1sim:~$ docker exec -it f1tenth_gym_ros-sim-1 /bin/bash
# The following commands configure the environment
root@5b24be3a66e0:/sim_ws# source /opt/ros/foxy/setup.bash
root@5b24be3a66e0:/sim_ws# source install/local_setup.bash
```

Entering the Simulator Container

The two `source` commands are required every time `gordon` enters the container. If they are not provided each and every time the ros commands will not be available.

```
# rebuild the gym (just in case)
root@5b24be3a66e0:/sim_ws# colcon build
# Start the rviz simulator
root@5b24be3a66e0:/sim_ws# ros2 launch f1tenth_gym_ros gym_bridge_launch.py
```

Starting the Simulator

The terminal in the bottom right of Figure 5.3 shows the output of entering the container and starting the simulator.

The last step is to connect to the display of the simulator. This is done through the NOVNC web interface. In Figure 5.3, `gordon` is using VNC to connect to `a301-f1sim.cs.unlv.edu`, within the VNC session `gordon` opens firefox to the url `http://localhost:8080/vnc.html`.

Alternatively, instead of VNC, you could point your local web browser to the assigned machine. For `gordon` that would be `http://a301-f1sim.cs.unlv.edu:8080/vnc.html`. Figure 5.3 is an example connection using a local web browser.

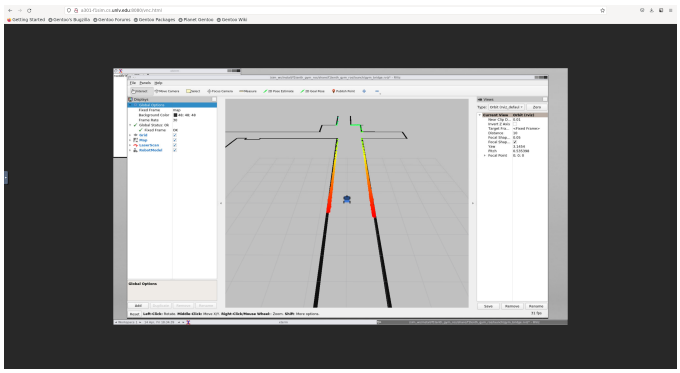

Figure 3: Display Container from a Local Web Browser

At this stage, you should be able to click on the rendered window, then drag the mouse to see the vehicle and environment from different perspectives. If you are unable to do so, remove the workspace from your directory and start anew from the beginning of this Subsection 5.3.

5.4 ROS Concepts

This subsection is based upon the first F1TENTH Lab on ROS:

`https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab1`.

As such, similar paths and names will be used to allow you to follow either set of instructions. Example interactions will use the username `gordon`, as a member of group 3, running of `a301-f1sim.cs.unlv.edu`. Within the instructions, your username should be replaced with `gordon`, and your group number should replace 3. The product of this subsection will be a package with a publisher and subscriber exchanging messages running within a new docker container.

The ros:foxy docker container assumes the docker user has root access on their local machine. If you are using the virtual machine prepared for you, you will not have root (or sudo) access outside of the container. Therefore, if you use the provided virtual machine file permissions must be managed manually.

5.4.1 ROS Workspaces

A prerequisite for a ROS package is a workspace. Summarily, a workspace is a directory which houses a set of related packages. A tutorial on workspaces is available at: `https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html`.

A workspace will be created at `/home/gordon/ws/ros_ws`. For each project a separate workspace will be needed. Within the workspace there is a `src` directory where the packages are stored. Given the relationship between workspaces and packages it may come as a surprise that the workspace is created after the package directory when following the instructions. This out-of-order creation is necessary due to the creation of the container without access to the ROS build tools. The ROS build tools, which initialize the workspace, are available only within the container.

Take care when following the instructions to note where a command is being issued, either within the docker container, or upon the localhost (f1sim).

5.4.2 Creating the Container

```
# On the localhost (f1sim)
gordon@f1sim:~/ws$ docker run -it -v /home/gordon/ws/ros_ws/src:/ros_ws/src \
  --name group3_ros ros:foxy
# Now running in the container
root@95fe0ebe6494:/#
root@95fe0ebe6494:/# exit # CTRL-D to exit the container
gordon@f1sim:~$
```

Creating an Running a Container

Take care to note the name of the current user `gordon`, paths within their home directory, and the name of the container which uses `group3` to differentiate the container from all others on the system.

As part of creating and running the container, a new directory to store the packages has been created `/home/gordon/ws/ros_ws/src` which is mounted within the container at `/ros_ws/src`. Within the container, the `/ros_ws` directory will be the ROS workspace, this directory will *not* be accessible from outside of the container. The user `gordon` does not have the rights to modify the `/home/gordon/ws/ros_ws/src` directory outside of the container.

```
# On the localhost (f1sim)
gordon@f1sim:~$ ls ws/ros_ws/ -alth1
total 12K
drwxr-xr-x 3 root   root   4.0K Apr 23 09:47 .
drwxrwxr-x 4 gordon gordon 4.0K Apr 23 09:47 ..
drwxr-xr-x 2 root   root   4.0K Apr 23 09:47 src
```

To remedy this problem, identify the user and group for `gordon` via the following commands outside of the container.

```
# On the localhost (f1sim)
gordon@f1sim:~$ id -u # get the user id for gordon
1001
gordon@f1sim:~$ id -g # get the group id for gordon
1002
```

With the user and group ids, 1001 and 1002 respectfully, attach to the docker container and modify the permissions of the `/ros_ws/src` to allow the local user to modify the directory and its contents.

```
# On the localhost
gordon@f1sim:~/ws$ docker start group3_ros
group3_ros
gordon@f1sim:~/ws$ docker attach group3_ros
# Now within the container
root@29629eeaf74a:/# cd /ros_ws/src/
root@29629eeaf74a:/ros_ws/src# ls -alth1
total 8.0K
drwxr-xr-x 3 root root 4.0K Apr 23 16:47 ..
drwxr-xr-x 2 root root 4.0K Apr 23 16:47 .
# Note, the . directory is owned by the user:group of root:root
# The user:group must match gordon's user:group, change the permissions
root@29629eeaf74a:/ros_ws/src# chown 1001:1002 .
root@29629eeaf74a:/ros_ws/src# ls -alth1
total 8.0K
drwxr-xr-x 3 root root 4.0K Apr 23 16:47 ..
drwxr-xr-x 2 1001 1002 4.0K Apr 23 16:47 .
root@29629eeaf74a:/ros_ws/src#
```

`gordon` then verifies the ability to create and modify files within the `/home/gordon/ws/ros_ws` directory outside of the container and then verify their contents from within the container.

```
# On the localhost
gordon@f1sim:~/ws$ cd
gordon@f1sim:~$ cd ws/ros_ws/src/
gordon@f1sim:~/ws/ros_ws/src$ echo "My name is Freeman" > secret.txt
gordon@f1sim:~/ws/ros_ws/src$ docker start group3_ros
group3_ros
gordon@f1sim:~/ws/ros_ws/src$ docker attach group3_ros
# Now within the container
```

```
root@29629eeaf74a:/# cd /ros_ws/src/
root@29629eeaf74a:/ros_ws/src# ls -alth1
total 12K
drwxr-xr-x 2 1001 1002 4.0K Apr 23 16:57 .
-rw-rw-r-- 1 1001 1002   16 Apr 23 16:57 secret.txt
drwxr-xr-x 3 root root 4.0K Apr 23 16:47 ..
root@29629eeaf74a:/ros_ws/src# cat secret.txt
My name is Freeman
root@29629eeaf74a:/ros_ws/src#
```

If there are problems modifying or creating files within the `/home/gordon/ws/ros_ws` directory outside of the container, the permissions of all files may be updated by entering the container and recursively changing the ownership of all files.

```
  gordon@f1sim:~/ws/ros_ws$ docker attach group3_ros
  root@626fa5bf19b0:/# cd /ros_ws/src
  root@626fa5bf19b0:/ros_ws/src# chown -R 1001:1002 *
```

If the container was created incorrectly, the container may be removed, only after being stopped. The following commands must be run *outside* of the container.

```
# Stop
gordon@f1sim:~$ docker stop group3_ros
group3_ros
# Remove
gordon@f1sim:~$ docker rm group3_ros
group3_ros
```

Stopping and Removing the Container

If the container has been stopped, it may be re-entered via the `docker start` and `docker attach` commands.

```
gordon@f1sim:~$ docker start group3_ros
group3_ros
gordon@f1sim:~$ docker attach group3_ros
root@626fa5bf19b0:/#
```

Attaching to a Container

Note: The command `docker start -a` attempts to start and attach to a container. Occasionally, starting a container using this command causes all commands within the container to hang. It is unclear why this occurs, but happens less frequently using the separate `docker start` and `docker attach` commands.

5.4.3 ROS Workspace

Begin by creating the ROS workspace from within the container. As a habit, source the ROS setup file every time you enter a container. This may not be necessary under all circumstances but it is a good habit to have to avoid problems in the future.

```
gordon@f1sim:~$ docker start group3_ros
group3_ros
gordon@f1sim:~$ docker attach group3_ros
# Good habits die hard
root@29629eeaf74a:/# source /opt/ros/foxy/setup.bash
root@29629eeaf74a:/# cd /ros_ws/
# The following command creates a workspace
root@29629eeaf74a:/ros_ws# colcon build

Summary: 0 packages finished [0.13s]
# To verify the workspace has been created list the contents of the
# directory and verify the build install and log directories have
# been created
root@29629eeaf74a:/ros_ws# ls
build  install  log  src
root@29629eeaf74a:/ros_ws#
```

Creating the Workspace

5.4.4 ROS Package

The following steps are taken from the ROS 2 Foxy Package Tutorial and adapted to our purposes, the original tutorial may be found here:

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

It is **strongly** encouraged that students follow the ROS workspace and package tutorials to understand their purpose and operation. The ROS group estimates this will take roughly an hour to complete. After completing the tutorials, return to this section where the instructions will be easier to follow.

The following commands will create a package named `ros_msg_pkg`. Upon entering the container, the first `source` command enters the ROS environment and the second `source` command enters the workspace, perform these two steps every time before operating on a workspace or a package within a workspace.

```
gordon@f1sim:~$ docker start group3_ros
group3_ros
gordon@f1sim:~$ docker attach group3_ros
root@29629eeaf74a:/# source /opt/ros/foxy/setup.bash
root@29629eeaf74a:/# source /ros_ws/install/local_setup.bash
root@29629eeaf74a:~# cd /ros_ws/src/
root@29629eeaf74a:/ros_ws/src# ros2 pkg create --build-type ament_python ros_msg_pkg
going to create a new package

>>> SNIP <<<

root@29629eeaf74a:/ros_ws/src# cd ..
# The following will build any package that is out of date
root@29629eeaf74a:/ros_ws# colcon build
Starting >>> ros_msg_pkg
Finished <<< ros_msg_pkg [0.76s]

Summary: 1 package finished [0.89s]
root@29629eeaf74a:/ros_ws#
```

Creating a ROS 2 Package

Then verify the package has been properly created and included. The package will not included until the workspace has been re-sourced.

```
root@29629eeaf74a:/ros_ws# ros2 pkg list | grep ros_msg
root@29629eeaf74a:/ros_ws#
# Not listed
root@29629eeaf74a:/ros_ws# source install/local_setup.bash
root@29629eeaf74a:/ros_ws# ros2 pkg list | grep ros_msg
ros_msg_pkg
# Now listed
```

Verifying ros_msg_pkg

Include the dependencies the package requires by editing `package.xml` within the `ros_msg_pkg`. Permissions may need to be updated from within the container to then update the file outside of the container on the localhost.

```
# From within the container.
root@29629eeaf74a:~# cd /ros_ws/src/
root@29629eeaf74a:/ros_ws/src#
root@29629eeaf74a:/ros_ws/src# chown -R 1001:1002 ros_msg_pkg
```

Updating Permissions

```
# On the localhost (f1sim)
gordon@f1sim:~$ cd /home/gordon/ws/ros_ws/src/ros_msg_pkg
# Use an editor of your own choice, emacs and vi are available.
gordon@f1sim:~/ws/ros_ws/src/ros_msg_pkg$ emacs -nw package.xml
```

Editing package.xml

Note: Editing files locally is another option, but will need to be transferred via scp
Insert the following XML tags after the `<license>` tag.

```
  <license>TODO: License declaration</license>

  <!-- Begin new tags -->
  <depend>rclcpp</depend>
  <depend>rclpy</depend>
  <depend>ackermann_msgs</depend>
  <exec_depend>ros2launch</exec_depend>
  <!-- End new tags -->

  <test_depend>ament_copyright</test_depend>
```

Updating package.xml

Install the dependencies before proceeding.

```
root@29629eeaf74a:/ros_ws# apt-get update

>>> SNIP <<<

Reading package lists... Done
root@29629eeaf74a:/ros_ws# rosdep install --from-paths src --ignore-src -y
executing command [apt-get install -y ros-foxy-ackermann-msgs]

>>> SNIP <<<

#All required rosdeps installed successfully
root@29629eeaf74a:/ros_ws#
```

Installing Dependencies

Add the talker node by creating a talker_node.py in the package directory.

```
gordon@f1sim:~$ cd /home/gordon/ws/ros_ws/src/ros_msg_pkg/ros_msg_pkg
gordon@f1sim:~/ws/ros_ws/src/ros_msg_pkg/ros_msg_pkg$ emacs -nw talker_node.py
```

Talker Node

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from ackermann_msgs.msg import AckermannDriveStamped, AckermannDrive


class TalkerNode(Node):

    def __init__(self):
        super().__init__("talker_node")
        self.declare_parameter('v', 0.2)
        self.declare_parameter('d', 0.1)
        self.speed = self.get_parameter('v').value
        self.steering_angle = self.get_parameter('d').value

        #Create /drive publisher
        self.drive_pub= self.create_publisher(
            AckermannDriveStamped,
            'drive',
            10)
        timer_period = 0.1 # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0
        self.get_logger().info("initial speed=%s , initial steering_angle=%s"%(self.speed,self.steering_angle))

    def timer_callback(self):
        msg = AckermannDriveStamped()
        msg.drive.speed = self.speed
        msg.drive.steering_angle = self.steering_angle
        self.drive_pub.publish(msg)
        self.get_logger().info('Publishing to /drive[%s]: speed=%s , steering_angle=%s'%(self.i,msg.drive.speed,msg.drive.
            ↪ steering_angle))
        self.i += 1

def main(args=None):
    rclpy.init(args=args)
    node = TalkerNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

talker_node.py

Add the relay node by creating a relay_node.py in the package directory.

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from ackermann_msgs.msg import AckermannDriveStamped, AckermannDrive


class RelayNode(Node):
    def __init__(self):
        super().__init__("relay_node")
        self.speed = 0.0
        self.steering_angle = 0.0
        #Create /drive subscriber
        self.drive_sub = self.create_subscription(
            AckermannDriveStamped,
            'drive',
            self.drive_callback,
            10)
        #Create /relay_drive publisher
        self.relay_drive_pub= self.create_publisher(
            AckermannDriveStamped,
            'relay_drive',
            10)
        timer_period = 0.1 # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def drive_callback(self,drive_msg):
        self.speed = 3 * drive_msg.drive.speed
        self.steering_angle = 3 * drive_msg.drive.steering_angle

    def timer_callback(self):
        msg = AckermannDriveStamped()
        msg.drive.speed = self.speed
        msg.drive.steering_angle = self.steering_angle
        self.relay_drive_pub.publish(msg)
        self.get_logger().info('Publishing to /relay_drive[%s}]: speed=%s , steering_angle=%s'%(self.i,msg.drive.speed,msg.drive.
            ↪ steering_angle))
        self.i += 1

def main(args=None):
    rclpy.init(args=args)
    node = RelayNode()
    rclpy.spin(node)
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

relay_node.py

To setup.py in the package directory, update the `console_scripts` variable to include the new talker and relay nodes.

```
      entry_points={
        'console_scripts': [
            'talker_node = ros_msg_pkg.talker_node:main',  # Added
            'relay_node = ros_msg_pkg.relay_node:main'     # Added
        ],
    },
```

The package is now ready to be built:

```
root@29629eeaf74a:~# cd /ros_ws/
root@29629eeaf74a:/ros_ws# colcon build
Starting >>> ros_msg_pkg
Finished <<< ros_msg_pkg [0.76s]

Summary: 1 package finished [0.89s]
```

Verify the executables have been correctly installed

```
root@29629eeaf74a:/ros_ws# ros2 pkg executables ros_msg_pkg
ros_msg_pkg relay_node
ros_msg_pkg talker_node
```

The executables are ready to run. To do so, two shells are needed within the container. There are two options for doing so, one method is to use `docker attach` and `docker exec`. The other method is to install `tmux` within the container. The use of `tmux` is recommended and students are encouraged to follow the instructions from the ROS and F1TENTH tutorials for installing and using it.

These instructions will use `docker exec` for simplicity. The following will start the talker and relay nodes within the container by using separate terminals.

```
gordon@f1sim:~$ docker start group3_ros
group3_ros
gordon@f1sim:~$ docker attach group3_ros
root@29629eeaf74a:/# source /opt/ros/foxy/setup.bash
root@29629eeaf74a:/# source /ros_ws/install/local_setup.bash
root@29629eeaf74a:/# ros2 run ros_msg_pkg talker_node
[INFO] [1682289030.905649223] [talker_node]: Publishing to /drive[1676}]: speed=0.2, steering_angle=0.1
>>> SNIP <<<
```

Terminal 1

```
# Must be run after docker attach in terminal 1
gordon@f1sim:~$ docker exec -it group3_ros bash
root@29629eeaf74a:/# source /opt/ros/foxy/setup.bash
root@29629eeaf74a:/# source /ros_ws/install/local_setup.bash
root@29629eeaf74a:/# ros2 run ros_msg_pkg relay_node
[INFO] [1682289025.882210772] [relay_node]: Publishing to
/relay_drive[1167}]: speed0.6000000089406967 ,  steering_angle=0.30000000447034836
>>> SNIP <<<
```

Terminal 2

The talker node publishes to /drive the speed and steering angle parameters which the relay node multiplies by three before publishing as /relay_drive parameters.

5.5 Automatic Emergency Braking

This subsection is based upon the F1TENTH Lab on Automatic Emergency Braking: https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab2. Begin by cloning the f1tenth Open Repository outside of the f1tenth Gym container directory created earlier.

```
# On the localhost
gordon@f1sim:~$ mkdir git
gordon@f1sim:~$ cd git/
gordon@f1sim:~/git$ git clone https://github.com/f1tenth/f1tenth_labs_openrepo
```

Within the open repository is a skeleton package that will be the basis of your submission. Copy the skeleton into the same directory containing the gym workspace.

```
gordon@f1sim:~$ cd ws/gym-one/
gordon@f1sim:~/ws/gym-one$ cp -r ~/git/f1tenth_labs_openrepo/f1tenth_lab2/safety_node .
gordon@f1sim:~/ws/gym-one$ ls
f1tenth_gym_ros  safety_node
```

After stopping the docker composition, edit the `docker-compose.yaml` file of the composition, include a volume for the newly copied `safety_node` directory.

```
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros$ emacs -nw docker-compose.yml
>>> SNIP <<<
    volumes:
      - .:/sim_ws/src/f1tenth_gym_ros
      # Add the next entry
      - /home/gordon/ws/gym-one/safety_node/:/sim_ws/src/safety_node
>>> SNIP <<<
```

Start the composition

```
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros$ docker compose up
```

In a separate terminal, enter the simulator container and attempt to build the `safety_node` package

```
 gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros$ docker exec -it f1tenth_gym_ros-sim-1 /bin/bash
root@445b65e71801:/sim_ws# source /opt/ros/foxy/setup.bash
root@445b65e71801:/sim_ws# source install/local_setup.bash
# The following may fail the build process
root@445b65e71801:/sim_ws# colcon build --packages-select safety_node
Starting >>> safety_node
Finished <<< safety_node [0.35s]

Summary: 1 package finished [0.49s]
```

If the following error is encountered, the `safety_node` has not been properly included in the container. Retry adding it as a mount point and restarting the composition.

```
root@445b65e71801:/sim_ws# colcon build --packages-select safety_node
[0.362s] WARNING:colcon.colcon_core.package_selection:ignoring unknown package 'safety_node' in --packages-select
```

5.5.1 Safety Node Template

Once the package is correctly installed, the safety node implementation may be completed. The source for the safety node is within the package at `safety_node/scripts/safety_node.py`.

5.5.2 Subscriptions

The Safety Node must subscribe to two topics, one subscription for the odomoeter and another for laser scan. The odometer publishes to `/ego_racecar/odom` and the laser scanner publishes to `/scan`. Within the constructor `__init__`, subscribe to the two topics using the `odom_callback` for odometer messages, and `scan_callback` for laser scan messages.

As part of the assignment, you will need to determine the contents of messages and methods to access them using the ROS2 tools provided. Begin by launching the gym

```
root@445b65e71801:/sim_ws# ros2 launch f1tenth_gym_ros gym_bridge_launch.py
```

In a new terminal, enter the simulator container where you may now utilize the ROS commands.

`ros2 topic list` Will enumerate the available topics that can be subscribed to.

```
root@445b65e71801:/sim_ws# ros2 topic list | grep scan
/scan
```

`ros2 topic type <topic name>` Will provide the message type for a topic

```
root@445b65e71801:/sim_ws# ros2 topic type /scan
sensor_msgs/msg/LaserScan
```

`ros2 topic echo <topic name>` Will provide the data from a type in a given topic

```
root@445b65e71801:/sim_ws# ros2 topic echo /scan
>>> SNIP <<<
```

`ros2 interface show <msg name>` Will describe the contents of a message type

```
root@445b65e71801:/sim_ws# ros2 interface show sensor_msgs/msg/LaserScan
>>> SNIP <<<
```

It is the output from `ros2 interface show` that will guide the implementation.

5.5.3 Publications

The `safety_node` is responsible for publishing an AckermannDriveStamped message to stop the vehicle when it detects the time to collision (TTC) falls below the acceptable threshold.

5.5.4 Topics

Note the following topic names for your publishers and subscribers:

`LaserScan: /scan`

`Odometry: /ego_racecar/odom`, specifically, the longitudinal velocity of the vehicle: `twist.twist.linear.x`

`AckermannDriveStamped: /drive`

5.5.5 Callback Responsibilities

`odom_callback`: It is the responsibility of `odom_callback` to update the current velocity of the vehicle using the information published to `/ego_racecar/odom`. The velocity is updated for every message processed by the callback.

`scan_callback`: It is the responsibility of the `scan_callback` to calculate the the TTC given the velocity information calculated by `odom_callback` and the information published to `/scan`. Further, if the TTC falls below the user defined threshold, the callback must publish an AckermannDriveStamped message with a velocity of 0.0 which will instruct the vehicle to stop. The TTC calculation and (potentially) publication of the stop message takes place for every message processed by the callback.

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node

import numpy as np
# TODO: include needed ROS msg type headers and libraries
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
from ackermann_msgs.msg import AckermannDriveStamped, AckermannDrive


class SafetyNode(Node):
    """
    The class that handles emergency braking.
    """
    def __init__(self):
        super().__init__('safety_node')
        """
        One publisher should publish to the /drive topic with a AckermannDriveStamped drive message.

        You should also subscribe to the /scan topic to get the LaserScan messages and
        the /ego_racecar/odom topic to get the current speed of the vehicle.

        The subscribers should use the provided odom_callback and scan_callback as callback methods

        NOTE that the x component of the linear velocity in odom is the speed
        """
        self.speed = 0.
        # TODO: create ROS subscribers and publishers.

    def odom_callback(self, odom_msg):
        # TODO: update current speed
        self.speed = 0.

    def scan_callback(self, scan_msg):
        # TODO: calculate TTC

        # TODO: publish command to brake
        pass

def main(args=None):
    rclpy.init(args=args)
```

```
    safety_node = SafetyNode()
    rclpy.spin(safety_node)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    safety_node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

scripts/safety_node.py

5.6 In Simulator Demonstration

To test the `safety_node` in the F1TENTH simulator, the ROS packages must be rebuilt and lanched. Refer to Section 5.3 for building and launching the simulator.

With the simulator running, open a new terminal and:

1. Enter the container.
2. Source the global and local setup
3. Rebuild the ROS packages

Once rebuilt, launch the keyboard control of the vehicle by issuing the following command:

```
root@16efddc49186:/sim_ws# ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

The keys corresponding to vehicle controls are displayed in the terminal. Leave this terminal open.

With the simulator running, the keyboard control running, open a new terminal and:

1. Enter the container.
2. Source the global and local setup
3. Enter the directory where the `safety_node` has been implemented.

Begin executing the safety node:

```
root@16efddc49186:/sim_ws# cd safety_node/scripts
root@16efddc49186:scripts# python3 ./safety_node.py
```

With `safety_node`, the simulation, and the keyboard teleop running, direct the vehicle towards a wall and verify the AEB system prevents the vehicle from colliding with it.

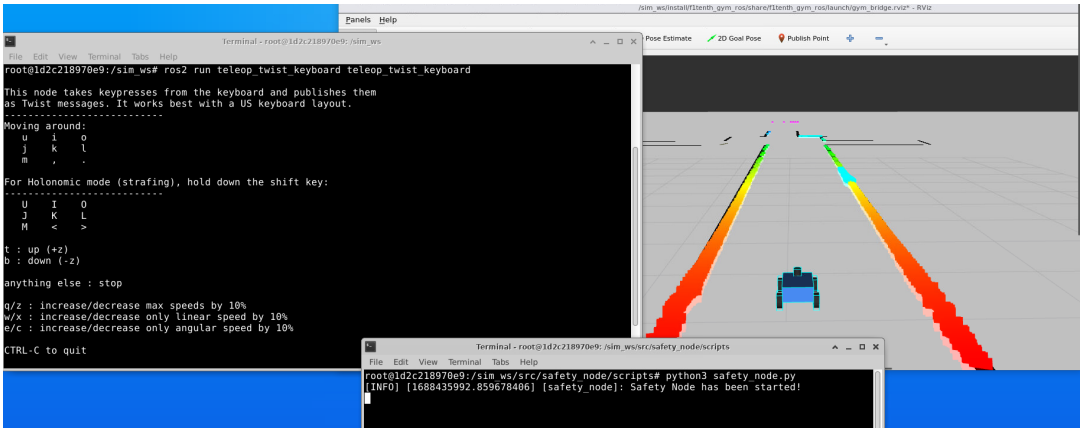

Figure 4: Simulator, Teleop and `safety_node.py`

A successful in simulator demonstration with the `safety_node` running will include:

1. (10 points) The vehicle remains stationary (indefinitely) without keyboard input.
2. (10 points) With keyboard input the vehicle will drive (indefinitely) if no wall or obstacle is within 1 meter.
3. (10 points) Driving straight towards a wall with keyboard input the vehicle will halt before colliding with any wall and maintain a minimum distance of .5 meters.
4. (10 points) Turning into a wall with keyboard input the vehicle will halt before colliding with any wall and maintain a minimum distance of .5 meters.

The evaluation uses distances rather than the implementation which considers time. This is for measurement purposes only, a value within a "reasonable" range of these distance measurements will receive full points. Focus more carefully upon the concept than the parameters of evaluation.

5.7 On Vehicle Demonstration

A submission which fails to be successfully demonstrated on the simulator will be prohibited from demonstration on the vehicle, receiving zero points for the on vehicle demonstration portion. Given the maximum speed of 60 miles per hour, the **maximum permissible TTC value is 1.5 seconds**. Before deployment on the vehicle the TTC used in the implementation must be presented.

The simulator differs from the physical vehicle in both form and ROS services. Component mounting on the vehicle has produced occlusion of the LiDAR's sensor range. The LiDAR has a 270 degree field of view. With the center of the LiDAR oriented to the front of the vehicle, the LiDAR scans counter clockwise from its "minimum angle" to its "maximum angle". Between the minimum and maximum angle there are 1080 points separated by the "angle increment". Refer to the "LaserScan Documentation" for more details.

Data returned from the LiDAR is an array of `ranges` where the first element (`range[0]`) is the distance to the point at the minimum angle and the final element (`range[1079]`) is the distance to the point at the maximum angle. The intermediate values are the distances to the points corresponding to the angle of their index within the `ranges` array. Figure 5.7 illustrates the LiDAR scan data with respect to the vehicle's orientation as well as the source of occlusion.

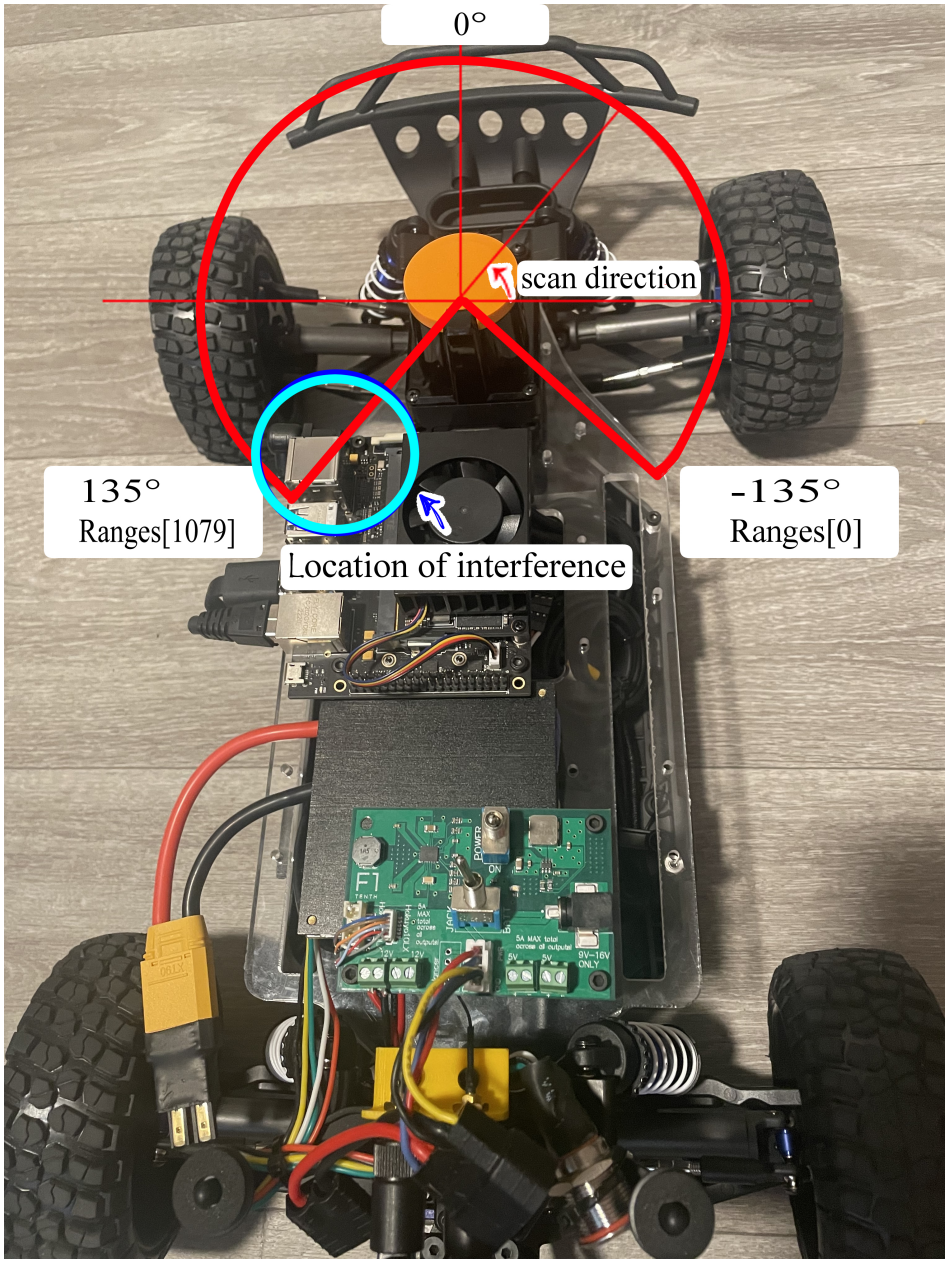

Figure 5: LiDAR Orientation and Interference

Another component on the vehicle occludes a portion of its scan data, returning constant values. A careful implementation will not be affected due to the location of the occlusion – consider the cos of the angle to one of the points. An incorrect implementation may behave erratically due to this occlusion. Please take care.

Within the simulator, the odometry topic is published to `/ego_racecar/odom`, on the vehicle the topic is published to `/odom`. Before deploying to the vehicle, subscriptions must be updated.

In place of keyboard control, the vehicle is operated with a PlayStation controller. The controller serves two purposes, the first is to allow the user to pilot the vehicle manually. To do so, the left bumper (LB) must be held down. The second purpose is to enable autonomous control of the vehicle e.g. emergency braking, navigation, etc. To enable autonomous signals to control the vehicle, the right bumper (RB) must be held down. ROS treats the controllers RB as a dead man's switch. When RB is held down, autonomous control signals are accepted. When RB is released, autonomous control signals are ignored. By default, manual control supersedes autonomous control.

This presents a conundrum for the Lab requirements. Manual control will be used to direct the vehicle toward a wall, autonomous control will attempt to stop the vehicle. Since manual control supersedes autonomous control, the car will collide with the wall. To address the problem, for this lab (and this lab only) autonomous control will be given higher priority than manual control.

The vehicle's priorities will be pre-configured. For reference, the mechanism for setting priorities is `mux.yaml`. The default values for the mux are:

```
ackermann_mux:
  ros__parameters:
    topics:
      navigation:
        topic   : drive
        timeout : 0.2
        priority: 10
      joystick:
        topic   : teleop
        timeout : 0.2
        priority: 100
```

Listing 1: Default `/home/f1/racecar_ws/src/f1tenth_system/f1tenth_stack/config/mux.yaml`

The navigation class covers autonomous control, joystick manual control. By setting the priority of navigation above that of the joystick, autonomous control will override manual control for its timeout period; increasing the timeout period gives the user time to respond and cease manual control. Both changes have been combined in yaml file below:

```
ackermann_mux:
  ros__parameters:
    topics:
      navigation:
        topic   : drive
        timeout : 5.0
        priority: 150
      joystick:
        topic   : teleop
        timeout : 0.2
        priority: 100
```

Listing 2: Modified `/home/f1/racecar_ws/src/f1tenth_system/f1tenth_stack/config/mux.yaml`

Updating the `mux.yaml` file alone will not begin enforcing the priorities.

```
  $ cd /home/f1/racecar_ws
  racecar_ws$ colcon buil
```

The following steps are required to deploy the `safety_node` package to the vehicle:

1. Connect to the wireless network the vehicle is also connected to. The SSID is `NETGEAR47` and the WPA key is `vanillacanoe001`.

2. The vehicle Verify your team's subdirectory has been created. The following example is for Team Galactic.

```
gordon@localhost:~$ ssh f1@rc0
# The password is 'password' without quotes
f1@rc0:~$ ls team-galactic
f1@rc0:~$
# No directory exists, create it.
f1@rc0:~$ mkdir -p team-galactic/gym-one
f1@rc0:~$ ls team-galactic
team-galactic
f1@rc0:~$
```

3. SCP the contents of the groups gym to the vehicle. Be **very** careful to transfer files to your own team's subdirectory. If a mistake is made, the vehicle may become inoperable for all of the teams.

```
gordon@localhost$ cd ws/gym-one
gordon@localhost:~/ws/gym-one$ scp -r f1tenth_gym_ros f1@rc0:team-galactic/gym-one/.
gordon@localhost:~/ws/gym-one$ scp -r safety_node f1@rc0:team-galactic/gym-one/.
```

4. Connect to the vehicle in at least three terminals, the first terminal will be referred to as TA, the second terminal TB, the third TC.

```
gordon@localhost:~$ ssh f1@rc0
f1@rc0:~$ echo ''This is terminal a!''
This is terminal a!
```

5. In TB, start the ROS system

```
# Terminal B
f1@rc0:~$ ./drive
# Leave this running
```

6. In TC, start the wireless controller input driver

```
# Terminal C
f1@rc0:~$ sudo ./ds4drv
# Leave this running
```

7. In TA, start the team's safety node (assuming galactic)

```
# Terminal A
f1@rc0:~$ cd team-galactic/gym-one/safety_node
fc@rc0:~/team-galactic/gym-one/safety_node$ python3 ./safety_node.py
# Woohoo!
```

A successful on vehicle demonstration of the `safety_node` will include:

1. (10 points) The vehicle remains stationary (indefinitely) without controller input.
2. (10 points) With manual controller input the vehicle will drive (indefinitely) if no wall or obstacle is within 1 meter.
3. (10 points) Driving straight towards an object with manual controller input the vehicle will halt before colliding with the object at a minimum distance of .5 meters.
4. (10 points) Turning towards an object with manual controller input the vehicle will halt before colliding with the object at a minimum distance of .5 meters.

5.8 Package Deliverable

Submit to canvas two `safety_node` packages, one for the simulator and another for the vehicle. Include the full contents of the package for `gordon` that would be the contents of the `/home/gordon/ws/gym-on/safety_node` directory. The packages should be tar'd and gzip'd, e.g.:

```
gordon@f1sim:~/ws/gym-one/$ tar -cvzf safety_node_sim.tar.gz safety_node
# Modifications of safety_node (or a separate package, hint, hint)
gordon@f1sim:~/ws/gym-one/$ tar -cvzf safety_node_vehicle.tar.gz safety_node_vehicle
```

Lab 1 Resources

Forticlient VPN The necessary VPN client for connecting to `a301-f1sim.cs.unlv.edu`. Download URL: https://www.it.unlv.edu/vpn

TigerVNC A client to connect to the virtual desktop server. Download URL: https://tigervnc.org

F1TENTH Gym The F1TENTH environment setup instructions are a streamlined form of the instructions found at: https://github.com/f1tenth/f1tenth_gym_ros

These instructions expand upon the "Without an NVIDIA gpu:" section. A personal machine with an NVIDIA GPU will perform far better than `a301-f1sim.cs.unlv.edu`

ROS 2.0 A native ROS development environment may be used. **However**, the instructions for the labs assume a docker container for the ROS installation. Instructions for ROS installation may be found at: https://docs.ros.org/en/foxy/Installation/Alternatives/Ubuntu-Development-Setup.html

Be careful to (1) have a base ubuntu 20.04 installation (2) install ROS 2 Foxy. Any deviation from (1) or (2) will result in failure.

Ubuntu 20.04 The **only** supported version of Ubuntu 20.04 for ROS 2.0 and F1TENTH. Download URL: https://releases.ubuntu.com/20.04.6/

F1TENTH Lab 1: ROS A "getting started" lab for setting up a ROS 2.0 environment. Instruction URL: https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab1

ROS 2.0 Workspace Tutorial Understanding and creating workspaces, a resource that is helpful before starting the Emergency Braking portion of the lab. Instruction URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

ROS 2.0 Packages Understanding and creating ROS 2.0 packages, a helpful resource to review before starting the Emergency Braking portion of the lab. Instruction URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

F1TENTH Lab 2: Emergency Braking The F1TENTH group's description of Lab 2: Emergency braking. Instruction URL: https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab2

ROS 2.0 Python Publisher/Subscriber Tutorial A brief introduction to writing an abstract publisher and subscriber. This may help to clarify some of the description of Automatic Emergency Braking. Instruction URL: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)

ROS 2 Tutorial Playlist- ROS2 Humble for Beginners An insightful introduction to ROS2 concepts, python packages, ROS2 topics, publishers and subscribers. Playlist URL https://youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy

LaserScan Documentation Documentation regarding the LaserScan and its messages. Documentation URL: https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html