Section 3

CS237A Fall 2025
Last updated Oct 10, 2025

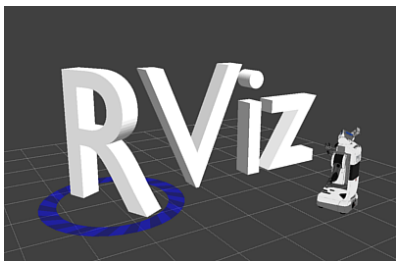
Overview

The goals for this section:

1. Learn to use RViz for visualizing information from your robot.
2. Learn to write launch files to launch multiple nodes.
3. Install nodes, launch files, and RViz configuration files in your ROS2 package.
4. Test out code in simulation

Information Visualization with RViz

RViz is a 3D visualization tool for ROS2, visualizing information such as maps and point clouds in a much more intuitive way compared to `ros2 topic echo`.



Prep

Task 0.1 — Fresh start

Close any open windows and terminals. Remove directory `~/autonomy_ws` if it exists. This might be leftover from a previous section.

Task 0.2 — GitHub Account

Use the following commands to logout of any existing GitHub accounts, and login to the account of one of your group members.

```shell
gh auth logout  # Log out of any accounts from previous sections.

gh auth login   # Login to the account your group is using.
# (1) press enter to select Github.com
# (2) select press enter to select HTTPS
# (3) type Y then hit enter
# (4) Take note of the supplied code, then press enter to open
#     the web browser. Follow the instructions in the browser to
#     login to your GH account.
# (5) Once you've logged in, return to the terminal and press enter
#     to finalize the authentication process.
```

Task 0.3 — Group Repo

→ If you haven't already, go to github.com and create a *private* repository for your group (make sure to initialize this repository with a `README` file), and add the other members of your group as collaborators (to do this go to Settings > Collaborators in your repo once you've created it).

Otherwise **git clone** your existing repository

Setup Workspace

Task 1.1 — **Directories.** Create the base directory structure for your group's ROS workspace, `~/autonomy_ws/src`.

Task 1.2 — **Clone and Branch.** Clone your group's GitHub repo to the `src/` directory, and create a new branch called `section3`.

Task 1.3 — **Create a Package.** Create an empty ROS package inside of your group's git repository with the name `s3_basic`.

Task 1.4 — **Examine and Customize.** Just like you did in the previous section, use the `cd` and `ls` commands to explore your group's newly created package.

> Optional: Use any editor to fill in the `description`, `maintainer`, and `license` fields of your newly created `package.xml` file. Use `Apache License 2.0` for the license field.

Task 1.5 — **Build.** Navigate back to the "root" of your workspace, `~/autonomy_ws/`, and build your ROS workspace using the command `colcon build --symlink-install` (do **not** run this command in any other directory). You should see something like the output below.

```
Shell
Starting >>> s2_basic          # package from last week's
section
Starting >>> s3_basic
Finished <<< s2_basic [0.87s]  # package from last week's
section
Finished <<< s3_basic [0.39s]

Summary: 2 packages finished [1.24s]
```

Task 1.6 — Add a node.
Create a **`scripts/`** directory inside **`s3_basic`**
copy over your **`constant_control.py`** file to this location from last week.

Edit it so that it makes the robot move in a straight line until we publish a kill message. Make this an executable, and edit your `CMakeLists.txt` and `package.xml` files. Build your workspace again from the root by running:

```shell
colcon build --symlink-install
```

Cheat Sheet:

```shell
ls                  # List files and folders in the current
directory
cd <directory>      # Change into directory
mkdir <directory>   # Make a directory with a given name
git clone <url>     # Clone a Git repository from a URL
git branch <name>   # Create a branch in a repository
git checkout <name> # Switch to a branch with a given name

# Create a new ROS package with the CMake build system.
ros2 pkg create --build-type ament_cmake <package_name>

# Run a ROS2 node
ros2 run <package_name> <script_name>

touch <file>     # create a file
chmod +x <file>  # make the file an executable

code <file>      # Open a file for editing with VSCode
```

Turtlebot Bring Up

Task 2.1 — Run in Simulation. First launch the simulator in a separate terminal by running the following command.

```shell
ros2 launch asl_tb3_sim root.launch.py
```

Troubleshooting: No visualization shows up? run “unset DISABLE_GUI”

Task 2.2 — Test the constant control node. In a new terminal, run the constant control node using `ros2 run`. This is the same as last week where you left off.

After running it for a while, publish a kill message with the following command (in a new terminal):

```shell
ros2 topic pub --once /kill std_msgs/msg/Bool '{data: true}'
```

Checkpoint — Call up a CA and demonstrate that your robot is moving in sim, and stops on a kill message!

Starting Up RViz

Task 3.1 — Launch RViz. Keep the simulator and constant control node running, and from another terminal, run:

```c/c++
rviz2
```

Now let's visualize some data!

Task 3.2 — Laser scans.

In RViz, to add a topic to the display, you need to click on the *Add* button on the bottom left. Once you've clicked this, a popup will come up. In the popup, click the ***By topic*** **tab** and scroll down until you see `LaserScan`. Double-click this and watch the scans appear. (If the scans are too faint, try changing their size from the *Displays* tab)

Task 3.3 — Slam Toolbox's Environment Map.

Next, add `Map` under the `/map` topic (again from the ***By topic tab***). This is how the mapping package delivers LiDAR data.

Task 3.4 — Slam Toolbox's Odometry.

Next, add `Odometry` under the `/odom` topic (again from the ***By topic tab***). The mapping package we use publishes the robot's odometry extracted from LiDAR data through this topic.

Task 3.5 — Coordinate Axes.

Next, add `Axes` from under the *By display type* tab (not *By topic*). This shows us where the world's origin point is and the orientation of the axes.

Task 3.6 — tf's Transform Tree.

Next, add `TF` from under the *By display type* tab. This shows us the axis origins and the orientations of any other frames we've defined. You can see we've defined quite a few for this robot (you may have to zoom in to read them).

Saving RViz's current configuration

That was quite a lot of topics to open! Imagine doing that every time you wanted to debug your ROS2 stack! Thankfully, there's a way we can save this current configuration to a file.

Task 4.1 — Save RViz Config.

Use *File -> Save Config As* to save the current configuration at `~/autonomy_ws/src/<group-repo>/s3_basic/rviz/section3.rviz` . Then close and reopen RViz, and load the configuration file you just created (using *File -> Open Config*) so you can see that it reloads all of your previously opened topics.

If you want to open RViz with your configuration file directly from the terminal, you can run:

```shell
rviz2 -d <path-to-rviz-config-file>
```

Checkpoint — Call a CA over to demonstrate that your RViz is properly set up!

Custom Launch Files

Phew! That was quite a number of terminals we just opened! It can be cumbersome to start all the nodes from scratch and set up RViz every time we want to run the stack.

Task 5.1 — Write a launch file.

To make this easier, create a launch file named `section3.launch.py` under a new `launch` directory in your `s3_basic` package which:

1. Starts the `constant_control.py` node.

   Hint: The cheat sheet at the end of this section provides instructions for setting this part up.

2. Opens RViz with the configuration file you saved. (This part is done for you.)

Task 5.2 — Register your launch and RViz configuration files with CMake. Add the following instructions to `s3_basic/CMakeLists.txt`.

```
Shell
install(DIRECTORY launch rviz DESTINATION
share/${PROJECT_NAME})
```

Task 5.3 — Test it out.

Build and source your workspace. Launch the simulator first, and then test your launch file by running:

```
Shell
ros2 launch <name of package> <name of launch file>
```

After a while, publish a kill message from a new terminal.

Checkpoint — Call a CA over to demonstrate that RViz, and the constant control node can be launched simultaneously using the launch file.

Cheat Sheet:

```
Python
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.substitutions import PathJoinSubstitution
from launch_ros.substitutions import FindPackageShare
```

```
def generate_launch_description():
    constant_control_node = # TODO

    # Launches RViz with your configuration file
    rviz_config =
PathJoinSubstitution([FindPackageShare("s3_basic"), "rviz",
"section3.rviz"])
    rviz_node = Node(
        package="rviz2",
        executable="rviz2",
        arguments=["-d", rviz_config],
        parameters=[{"use_sim_time": True}]
    )

    return LaunchDescription([
        constant_control_node,
        rviz_node
    ])
```

Clean Up

Clean up the workstation.

1. Push your code (**`add`**, **`commit`**, and **`push`**) to your **`section3`** branch!

Shell

```
git add s3_basic/              # add the 's3_basic/' folder
```

```
git commit -m "<message>"      # commit your changes to the
local branch

git push -u origin section3    # push for the first time
after creating a branch

git push                       # subsequent pushes on the
same branch
```

2. Create a PR either from the GitHub web page or by running **`gh pr create`** in your terminal.

3. Verify that you've pushed correctly by checking if the code you wrote is in the repository on GitHub.com

4. Delete the **`~/autonomy_ws/`** .

```
Shell
rm -rf ~/autonomy_ws
```

5. Log out of GitHub on the laptop.

```
Shell
gh auth logout
```

6. Close all terminals and windows!

Checkpoint — Call a CA over to sign you out!