op1319

Iuro Nascimento

Mar 23, 2025

CONTENTS:

This is the main landing page of the documentation.

CHAPTER

ONE

OPEN-RMF_GETTING_STARTED

This is a OP1319 repository that contains a getting started tutorial to use the Open-RMF and its tools.

CHAPTER

TWO

ROADMAP

There are a few options to follow:

1. Run the Open-RMF demos: install the Open-RMF from binaries and follow the tutorial until the *rmf_demos section*.
2. Create a custom map: install the Open-RMF from binaries and follow the tutorial until the *custom map section*.
3. If you wish to use the rmf_web: install from binaries and follow the tutorial until the *rmf_web section*
4. If you wish to have full control over the Open-RMF: install from source and do all the sections in this tutorial.

2.1 README sections

CHAPTER

THREE

ROS 2

One of the main tools used in the project is ROS 2, which is used to communicate with the robots. This is the first thing to be installed. Install ROS 2 by following the ROS 2 official install guide.

Important: In the ROS 2 official install guide, they recommend to install the ROS 2 desktop. However, we recommend everyone to install the ROS 2 full. By installing the ROS 2 full, the Gazebo Harmonic is installed automatically. Thus, you may skip the *Gazebo section*, you will only need the *Gazebo ROS 2 section*. To install the ROS 2 full, when you reach the install part of the ROS 2 install guide, run the command:

```
sudo apt install ros-jazzy-desktop-full
```

In the ROS 2 official install guide, they install only the ROS 2 desktop.

CHAPTER

FOUR

GAZEBO INSTALLATION

Another main tool used in the project is the Gazebo simulator. We are using the Gazebo Harmonic.

> If you installed the ROS 2 full in the *ROS 2 section*, there is no need to install the Gazebo Harmonic again and you may skip this section. Continue in the *Gazebo ROS 2 section*. For the sake of documentation, and for the case we want to install only the ROS 2 desktop in the final application, we will leave this section here.

We need to install both the Gazebo and the Gazebo ROS 2 packages. First, install Gazebo. **If only the Gazebo ROS 2 is installed, the demo examples may not run correctly.**

Follow the Gazebo Harmonic installation guide. Here follows a copy of the instructions.

Harmonic binaries are provided for Ubuntu Jammy (22.04) and Ubuntu Noble (24.04). The Harmonic binaries are hosted in the packages.osrfoundation.org repository. To install all of them, the metapackage `gz-harmonic` can be installed.

First install some necessary tools:

```
sudo apt-get update
sudo apt-get install curl lsb-release gnupg
```

Then install Gazebo Harmonic:

```
sudo curl https://packages.osrfoundation.org/gazebo.gpg --output /usr/share/keyrings/
↪pkgs-osrf-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/pkgs-osrf-
↪archive-keyring.gpg] http://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_
↪release -cs) main" | sudo tee /etc/apt/sources.list.d/gazebo-stable.list > /dev/null
sudo apt-get update
sudo apt-get install gz-harmonic
```

All libraries should be ready to use and the `gz sim` app ready to be executed.

If the gazebo window does not load correctly, check the *multicast issue*.

CHAPTER

FIVE

GAZEBO ROS 2 INSTALLATION

This Gazebo ROS 2 is required to run the Open-RMF demos. Thus, do not skip this section.

1. If you are using ROS 2, follow this tutorial to install the ROS 2 Gazebo Harmonic
2. Install it with : `sudo apt-get install ros-jazzy-ros-gz`

CHAPTER

SIX

OPEN-RMF

The Open-RMF is the other main tool used in the project. It is a robot fleet management system. It is ROS 2 based. To install it, you can follow the official install guide. Since this is the most important part of the project, the installation steps are copied here. For the sake of clarity, some comments were removed. See them below.

6.1 Preparation to install the Open-RMF

Instruction below are aimed at `Ubuntu 24.04` with `ROS 2 Jazzy`.

First please install ROS 2 via binary debians as mentioned in *the ROS 2 section*.

Install all non-ROS dependencies of Open-RMF packages,

```
sudo apt update && sudo apt install ros-dev-tools -y
```

The `rosdep` helps install dependencies for ROS packages across various distros and will be installed along with `ros-dev-tools`. However, it is important to update it.

```
# run if first time using rosdep.
sudo rosdep init
rosdep update
```

Update `colcon mixin` if you have not done so previously.

```
colcon mixin add default https://raw.githubusercontent.com/colcon/colcon-mixin-
↪repository/master/index.yaml
colcon mixin update default
```

6.2 Choose your Open-RMF installation type

If you plan to develop new tools for the Open-RMF install it from source using the *install from source section*. if you just want to test the Open-RMF demos and do not plan to modify the Open-RMF source code *install it from binaries*.

6.3 Binary installation

First follow the instruction in the *Preparation section*. Now, install Open-RMF debian packages:

```
sudo apt update && sudo apt install ros-jazzy-rmf-dev
```

Note: This will install all necessary debian packages to run Open-RMF except for those in rmf_demos for reasons described in this issue. To install the demos check *the rmf_demos section*.

6.4 Building Open-RMF from source

Follow the official guide, which is also presented below.

Install Gazebo before installing the Open-RMF from source. Follow the instructions in the *Gazebo installation section*

Do not try to install both ways (binary packages and from source).

First follow the instruction in the Setup section.

Due to newer changes in the source code, there might be conflicts and compilation errors with older header files installed by the binaries. Hence before proceeding, make sure to uninstall all Open-RMF binaries for the ROS 2 distro that will be sourced for the build.

```
sudo apt purge ros-<distro>-rmf* && sudo apt autoremove
```

6.4.1 Download the source code

Open-RMF is a collection of several repositories/packages. The rmf.repos provides a snapshot of these packages. This repository stores different versions of this file depending on the ROS 2 distribution and version of the release. Depending on the use case, you may choose to download this file from one of the following branches in this repository:

- `main` : Latest version of all packages which may or may not have a binary release yet. This is recommended for development and bug fixes.
- `<distro>`: The latest version of the packages for a specific ROS 2 distro. This may be different from `main` as new API/ABI breaking features merged into `main` will not be backported into `<distro>` branches. The packages downloaded from `<distro>` may have changes that are not yet available in binaries.
- `<distro>-release` : Where `<distro>` is a supported ROS 2 distribution (eg. `humble`). The version of packages here will correspond to those of the latest binaries available for this distro.
- `release-<distro>-YYMMDD` : A tag for a specific `<distro>` where the version of packages correspond to those of the binaries available on `YYMMDD`. See Releases for additional information for each release. This is useful if users would like to build packages from an older release.

Setup a new ROS 2 workspace and pull in the demo repositories using `vcs`. Replace `main` with the branch of your choice.

```
mkdir -p ~/rmf_ws/src
cd ~/rmf_ws
wget https://raw.githubusercontent.com/open-rmf/rmf/main/rmf.repos
vcs import src < rmf.repos
```

Update your rosdep definitions and install dependencies via `rosdep`.

```
cd ~/rmf_ws
sudo apt update
rosdep update
source /opt/ros/jazzy/setup.bash # replace jazzy with preferred ROS 2 distro.
rosdep install --from-paths src --ignore-src --rosdistro $ROS_DISTRO -y
```

NOTE: We strongly recommend compiling Open-RMF packages with `clang` as compiler and `lld` as linker.

```
sudo apt update
sudo apt install clang clang-tools lldb lld libstdc++-12-dev
```

Compile the workspace after sourcing the ROS 2 distro of choice.

```
cd ~/rmf_ws
export CXX=clang++
export CC=clang
colcon build --mixin release lld
```

The build process will require python cmake module to be installed. On Ubuntu 24.04 it is not possible to install it system wide. Thus, create a virtual environment with all the system packages and build within the package:

```
cd ~/rmf_ws
virtualenv --system-site-packages venv_build
. venv_build/bin/activate
pip install cmake
export CXX=clang++
export CC=clang
colcon build --mixin release lld
```

> NOTE: The first time the build occurs, many simulation models will be downloaded from Gazebo Fuel to populate the scene when the simulation is run. As a result, the first build can take a very long time depending on the server load and your internet connection (may be up to an hour). To build without downloading models append `--cmake-args -DNO_DOWNLOAD_MODELS=On` to the `colcon build`.

After the build is completed, go to the *installing the rmf_demos section*.

6.5 Installing the rmf_demos

> If you installed the Open-RMF from source following *the source install section*, you can skip the rmf_demos installation because they already come together with the source code.

To build `rmf_demos` from source, first determine the version of `rmf_demos` that corresponds to the latest binary release for your distro. In this project we will use the `jazzy` version.

Other versions of `rmf_demos` for this release can be found in the `rmf.repos` file in the `humble-release` branch of this repository, for example here.

Next create a ROS 2 workspace and build `rmf_demos` from source. Replace the `2.0.3` tag below with the version of `rmf_demos` for your distribution.

```
mkdir ~/rmf_ws/src -p
cd ~/rmf_ws/src
git clone https://github.com/open-rmf/rmf_demos.git
cd rmf_demos
git checkout jazzy
```

> If your computer does not allow installing python packages system wide, you will need to create a python virtual environment. Some of the Open-RMF packages will require additional python packages such as Python `cmake`.

The virtualenv is a self contained python environment separated from the system wide python installation. We need to create a virtualenv because Ubuntu 24.04 does not allow installing python packages system wide. 0. Install virtualenv: `sudo apt install virtualenv`

1. Go to the rmf_ws: `cd ~/rmf_ws`
2. Create the venv with the system wide python packages (check this link). We need to create the virtualenv with all the packages because of the ROS 2 packages: `virtualenv --system-site-packages venv_rmf_ros2`

3. Activate it: `. venv_rmf_ros2/bin/activate`
4. **Python cmake is required** : `pip install cmake`

Now, build the workspace after the virtual environment is activated:

```
cd ~/rmf_ws
colcon build
```

If you installed the open-rmf from the binary packages, you need to install some ROS 2 dependencies to run the demo examples correctly:

```
cd ~/rmf_ws
rosdep install -i --from-path src --rosdistro jazzy -y
colcon build
```

The command above should fix any missing dependency to run the demos.

6.6 Testing a demo

To run some demonstrations of Open-RMF in simulation, see README in rmf_demos. There you can find the command to run different ready demos from Open-RMF. **Do not forget to change to the `jazzy` branch on the rmf_demos repository.**

Try running the hotel demo:

```
cd ~/rmf_ws
source /opt/ros/jazzy/setup.bash
source ~/rmf_ws/install/setup.bash
ros2 launch rmf_demos_gz hotel.launch.xml
```

The demo should open with RViz and Gazebo.

If the gazebo window does not load correctly, check the *multicast issue*.

Next, run the following command in a new terminal to command the robots in the scene:

```
source /opt/ros/jazzy/setup.bash
source ~/rmf_ws/install/setup.bash
ros2 run rmf_demos_tasks dispatch_patrol -p restaurant  L3_master_suite -n 1 --use_sim_
↪time
ros2 run rmf_demos_tasks dispatch_clean -cs clean_lobby --use_sim_time
```

If everything works, the robots should start moving.

To run some demonstrations of Open-RMF in simulation, see README in the rmf_demos repository. **Always remember to change to the jazzy branch in the rmf_demos repository.**

CHAPTER

SEVEN

TRAFFIC EDITOR

This tutorial was copied from the original traffic editor repository.

7.1 Installation

This repository is structured as a collection of ROS 2 packages and can be built using `colcon`. For full installation of RMF, please refer to here.

The `rmf_building_map_tools` package requires the following Python 3 dependencies to generate worlds:

```
sudo apt install python3-shapely python3-yaml python3-requests
```

Clone the repository to your computer:

```
git clone https://github.com/open-rmf/rmf_traffic_editor.git
```

Go to the repository folder and source ROS 2 and the previous installed rmf workspace:

```
cd rmf_traffic_editor
source /opt/ros/jazzy/setup.bash
source ~/rmf_ws/install/setup.bash
```

You will need python cmake to build the ROS 2 rmf traffic editor packages. Thus, we need to create and activate a python virtual environment to install cmake python since Ubuntu 24.04 does not allow installing python packages system wide:

```
virtualenv --system-site-packages venv_rmf_traffic_editor
. venv_rmf_traffic_editor/bin/activate
```

After activating the virtualenv, install python cmake package:

```
pip install cmake
```

Now, you can build the packages form the rmf_traffic_editor inside the rmf_traffic_editor folder:

```
cd ~/rmf_traffic_editor
colcon build
```

To test if the traffic editor installation was successfull source the rmf_traffic_editor and run it:

```
source ~/rmf_traffic_editor/install/setup.bash
traffic-editor
```

7.2 Overview

Most things here were copied from the official open-rmf book.

The traffic-editor is composed of the GUI and other tools to auto-generate simulation worlds from GUI outputs. The GUI is a interface that create and annotate 2D floor plans with robot traffic along with building infrastructure information. Also, existing floor plans of the environment, such as architectural drawings can be opened, which simplify the task and provide a "reference" coordinate system for vendor-specific maps. For such cases, `traffic-editor` can import these types of "backgroud images" to serve as a canvas upon which to draw the intended robot traffic maps, and to make it easy to trace the important wall segments required for simulation.

The `traffic_editor` GUI projects are stored as `yaml` files with `.building.yaml` file extensions. Although the typical workflow uses the GUI and does not require hand-editing the `yaml` files directly, we have used a `yaml` file format to make it easy to parse using custom scripting if needed. Each `.building.yaml` file includes several attributes for each level in the site as annotated by the user. An empty `.building.yaml` file appears below. The GUI tries to make it easy to add and update content to these file.

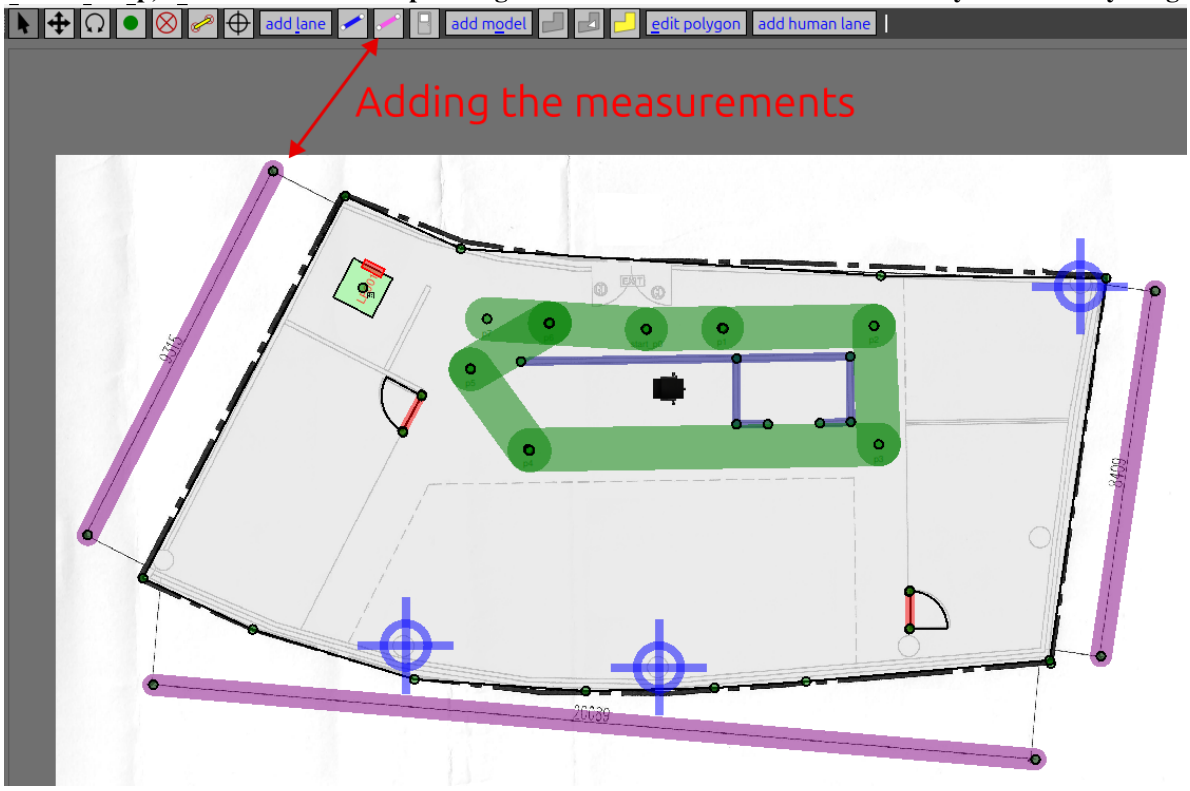
```
levels:
    L1: doors:
        - []
        - drawing:
        - filename:
        - fiducials:
        - elevation: 0
        - flattened_x_offset: 0
        - flattened_y_offset: 0
        - floors:
            - parameters: {}
            - vertices: []
        - lanes:
            - []
        - layers:
            {}
        - measurements:
            - []
        - models:
            - {}
        - vertices:
            {}
    walls:
        {}
lifts:
    {}
    name: building
```

At this point, we recommend you to read the official rmf book sections about the traffic-editor. Please read all the sections from GUI Layout until the Conclusion section. This tutorial will make you learn how to create a new scenario to be used with the open-rmf.

7.3 Tips for the rmf-traffic-editor tutorial

- Some ready map examples can be found in your rmf_ws folder where you installed the open-rf. They are under the folder: `/rmf_ws/src/demonstrations/rmf_demos/rmf_demos_maps/maps`

- We recommend to do the tutorial with the office map, which is the same from the official tutorial for the traffic-editor.
- To use run the traffic-editor tutorial, we highly recommend you to copy (duplicate) the office map to another folder and open it from there instead of working on the `rmf_demos_maps`.
- **The first thing you should do after you open the traffic-editor and load your map .png file is to add the measurements on all the known measurements in the map. After adding the measurements (the pink lines), save the map, close the editor and open it again. The traffic-editor will automatically rescale everything.**



- Adding floor polygon: first add the vertices (green dots) of the polygon and next connect them with the floog polygon tool.
- If for some reason the zooming of the map behaves strange or if everything in the map vanish for every opened map. Click on the **View** tab and reset the zoom. The map should appear again.
- **Adding feature points**: make sure you select the correct level when adding the feature point. Select the level map to add points on the level map and select the map_scan (the name you gave for the scan of the map) to add features to the scan. Connect the features from the level with the features of the scan using the **Add constraint** option. Last, click on `Edit > Optimize Layer Transform`. The scan and the map should align automatically.

CHAPTER

EIGHT

RUNNING A CUSTOM MAP

Follow the tutorial on the *custom map tutorial page*.

CHAPTER

NINE

ADDITIONAL TOOLS

There are some additional tools that may be usefull for some more advanced applications with Open-RMF.

9.1 Getting started with the rmf-web

This tutorial was copied from the official rmf-web repository

9.1.1 Prerequisites

We currently support Ubuntu 24.04, ROS 2 Jazzy and the latest Open-RMF release. Other distributions may work as well, but is not guaranteed.

Install pnpm and nodejs

```
curl -fsSL https://get.pnpm.io/install.sh | bash -
# after installing, source your bashrc file
source ~/.bashrc
pnpm env use --global lts
```

For Debian/Ubuntu systems,

```
sudo apt install python3-pip python3-venv
```

9.1.2 Installing Open-RMF Web

To run the Open-RMF web with the demos, you need to have followed the *rmf_demos section*.

Get the rmf-web

Clone the *rmf-web repository* .

```
git clone https://github.com/open-rmf/rmf-web.git
```

Install dependencies

Run the following commands to install the rmf-web things.

```
cd path/to/rmf-web
pnpm install
```

Launching for development

Source Open-RMF and launch the demo dashboard in development mode,

```
# For binary installation
source /opt/ros/jazzy/setup.bash

# For source build: source the rmf_ws you installed in the steps above
source /path/to/your/rmf/workspace/install/setup.bash

cd path/to/your/rmf-web
cd packages/rmf-dashboard-framework
pnpm start
```

This starts up the API server (by default at port 8000) which sets up endpoints to communicate with an Open-RMF deployment, as well as begin compilation of the demo dashboard. Once completed, it can be viewed at localhost:5173.

If presented with a login screen, use `user=admin password=admin`.

Ensure that the fleet adapters in the Open-RMF deployment is configured to use the endpoints of the API server. By default it is `http://localhost:8000/_internal`. Launching a simulation from `rmf_demos_gz` for example, the command would be,

```
ros2 launch rmf_demos_gz office.launch.xml server_uri:="http://localhost:8000/_internal"
```

Now you can dispatch tasks using the rmf-web interface clicking at localhost:5173.

Configuration

- See the api-server docs for API server run-time configurations.
- rmf-dashboard-framework allows you to easily build a dashboard.

9.2 Docker

Now, to use some open-rmf tools, it will be useful to install docker because some tools provide docker files ready to run (for example, rmf-web).

This instructions were copied from the official docker tutorial.

9.2.1 Prerequisites

To install Docker Desktop successfully, you must:

- Meet the general system requirements.
- Have an x86-64 system with Ubuntu 22.04, 24.04, or the latest non-LTS version.
- For non-Gnome Desktop environments, `gnome-terminal` must be installed:

  ```
  $ sudo apt install gnome-terminal
  ```

Note: Although the official docker installation says it requires `gnome-terminal`, other terminals work as well such as `Allacrity` and `Wezterm`. Thus, you can try to use other terminals instead of `gnome-terminal`.

9.2.2 Install using the apt repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker apt repository. Afterward, you can install and update Docker from the repository.

1. Set up Docker's apt repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
↪docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]␣
↪https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

   Note

   If you use an Ubuntu derivative distribution, such as Linux Mint, you may need to use UBUNTU_CODENAME instead of VERSION_CODENAME.

2. Install the Docker packages.

   Latest Specific version

   To install the latest version, run:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin␣
↪docker-compose-plugin
```

---

3. Verify that the installation is successful by running the hello-world image:

```
$ sudo docker run hello-world
```

   This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.

You have now successfully installed and started Docker Engine.

CHAPTER

TEN

SOLUTIONS TO SOME REPORTED ISSUES

10.1 Gazebo issue with multicast

Thanks to Gabriel Elan (@Elan-gab) for spotting and solving the problem.

When running rmf_demos_gz package, an 'Gazebo Sim' window opens alongside with rviz, but does not respond, even after a long time waiting. The cause of this problem is described here and here as a network configuration issue that is caused by firewall restrictions. This troubleshooting suggests enabling multicast, which resolved the problem.

To test the multcast issue run:

```
ros2 multicast receive
```

In another terminal, run:

```
ros2 multicast send
```

If the receiver does not respond, multicast is being blocked by the firewall. To enable multicast:

```
sudo ufw allow in proto udp to 224.0.0.0/4
sudo ufw allow in proto udp from 224.0.0.0/4
```

CHAPTER

ELEVEN

HOW TO BUILD THE HTML AND LATEX DOCUMENTATION

This is a brief guide on how to build the html and/or latex documentation. If you have not installed the documentation build system (Sphinx + extensions), refer to the installation *documentation*.

11.1 Workflow and installed environment

The build system relies on python environments to isolate its dependencies from the system packages. To isolate even further, we use poetry to manage a isolated environment for the build system, and we automatically activate the environment when you type cd to go into the root of the project and automatically deactivates this environment when leaving the project.

Once you are in the project root, you can run:

```
$ make html
```

or

```
$ make latex
```

to generate the desired documentation from the markdown. The documentation files are built into `build/{html, latex}`. regular pdflatex can handle the latex to pdf conversion and any server like github pages can host the html documentation website.

Note: you don't have to change anything in the markdown files that are compatible with GitHub Markdown, you only have to reserve the index.md file to include markdown + extensions from Myst parser so the build process can work. Those extensions can be used in other markdown files, but they are not recognized by GitHub.

11.2 Automatically handling of environments switch

After the installation script set up a python environment for working with Sphinx, you can `cd` into the project root, pyautoenv takes care of automatically deactivate any active python environment activate the current project poetry environment. After you change directory to outside the project pyautoenv automatically deactivate the project environment, but do not activate the previous environment, so if there was any environment activated, you have to reactivate yourself.

If you don't want to isolate the build system from the system or your base environment, refer to the installation *documentation* on how to install the build system into your environment.

11.3 Manually handling of environment switch

to activate the environment, if you have another environment activated, deactivate first to avoid conflicts running:

```
deactivate
```

or for Conda environments:

```
$ conda deactivate
```

Otherwise, run inside the repository folder:

```
$ deactivate || eval $(poetry env activate)
```

After building the docs, deactivate by running and reactivating any environment you were using before this process.

This process isolates the Sphinx from the rest of the system and environment packages from Sphinx and dependencies.

11.4 Building html or latex documentation from the markdown files

build the html or latex documentation: From root of the repository, run:

```
  $ make html
```

or

```
$ make latex
```

or equivalently:

```
$ sphinx-build -b html . build
```

The HTML output will appear in ./build/html/.

CHAPTER

TWELVE

BUILD SYSTEM INSTALLATION

The script `install_build_deps.sh` sets up an isolated environment using Poetry for building Sphinx-based documentation from Markdown files. Here's the high-level explanation of the main package installed:

1. **pipx**:
    - pipx is a tool to install and run Python CLI tools in isolated environments.
    - Installing pipx "globally" (i.e., in your user environment) allows you to cleanly install tools like Poetry without polluting system packages.
2. **Poetry via pipx**:
    - Poetry is a Python dependency manager and virtual environment manager.
    - By installing it via pipx, Poetry itself is isolated from system Python.
3. **Poetry environment**:
    - Once Poetry is installed, we create (or use) a Poetry project that contains all dependencies for generating Sphinx docs.
    - Sphinx, myst-parser, sphinx-autodoc-typehints, and the sphinx_rtd_theme are installed *inside* the Poetry environment, not in your system Python.
4. **Why these packages?**:
    - sphinx: The main Sphinx engine to transform .md/.rst into HTML (and other formats).
    - myst-parser: Lets Sphinx understand Markdown files using MyST syntax.
    - sphinx-autodoc-typehints: Display Python function/method type hints in the docs (useful if you had Python code, but it won't hurt if you don't).
    - sphinx_rtd_theme: Popular "Read the Docs" HTML theme for Sphinx.
    - napoleon (part of sphinx.ext) parses Google or NumPy-style docstrings (again, useful if you had Python code).
5. **Isolation benefits**:
    - You might already use Conda, virtualenv, or other environment managers for your own Python projects. Poetry creates *its own* environment for the documentation tools, so it won't interfere with your personal environment or system packages.
    - You only "activate" this Poetry environment when you want to build docs; otherwise, your personal environment remains unaffected.
6. **Auto-activation**:

- pyautoenv is automatically installed, so the project environment is automatically activated by pyautoenv when you `cd` into the root and deactivate when leaving the project folder.

CHAPTER

THIRTEEN

OPEN-RMF-CUSTOM-MAP

13.1 Preparing the custom map workspace folder

> We highly recommend you to create a new workspace without any other package. Otherwise, if you reuse the same workspace `rmf_ws` you used to install the Open-RMF from source, it will take much more time to build, clean, and rebuild this custom map example if needed.

Create a RMF ROS 2 workspace directory:

```
mkdir -p rmf_ws_custom_map/src
```

Change to the `src` folder:

```
cd rmf_ws_custom_map/src
```

Clone the Open-RMF demos repository in the workspace `src` folder:

```
git clone https://github.com/open-rmf/rmf_demos.git
```

Change to the `jazzy` branch (**this branch is more stable than the main branch for the ROS 2 jazzy**):

```
cd rmf_demos
git checkout jazzy
```

Change back to the workspace folder, check for missing dependencies:

```
cd ~/rmf_ws_custom_map
rosdep install -i --from-path src --rosdistro jazzy -y
```

Create a python virtual environment with system wide packages and activate it:

```
virtualenv --system-site-packages venv_custom_maps
. venv_custom_maps/bin/activate
```

Install Python cmake in the virtualenv:

```
pip install cmake
```

Source your ROS 2 installation and build the workspace:

```
source /opt/ros/jazzy/setup.bash
colcon build
```

If the `colcon build` result in any packages not being built correctly, run it a second time. After the second time, all packages will probably be built, not sure why.

Let us check if the build worked by running a demo example. Source your open-rmf installation (that you installed from source), source the custom map workspace, and run the demo. **Follow exactly this order of `source` commands**:

```
# source the open-rmf installation from source
source ~/rmf_ws/install/setup.bash
# Source the custom_map workspace
source ~/rmf_ws_custom_map/install/setup.bash
# Run the office demo
ros2 launch rmf_demos_gz office.launch.xml
```

The RViz window should open quickly, but the Gazebo Window may take a while depending on your computer specifications (CPU, GPU, and so on). **If Gazebo does not open on the first try, close the RViz and kill the demo command on the terminal and run it again. Wait some time for the Gazebo to open.**

Before closing the demo run from the last step, try running a task. Open a new terminal, source the ROS 2 installation, source the `~/rmf_ws` installation, source the `rmf_ws_custom_map` and run the task command:

```
source /opt/ros/jazzy/setup.bash
source ~/rmf_ws/install/setup.bash
source ~/rmf_ws_custom_map/install/setup.bash
ros2 run rmf_demos_tasks dispatch_delivery -p pantry -ph coke_dispenser -d hardware_2 -
↪dh coke_ingestor --use_sim_time
```

The robot 2 should go to the pantry, get a coke and deliver to the coke ingestor.

13.2 Changing the map

Create your map in the `traffic-editor` and copy the saved `<your_map>.building.yaml` to the `~/rmf_ws_custom_map/src/rmf_demos/rmf_demos_maps/maps/office/office.building.yaml`with the same name. Name it as `office.building.yaml`. Inside the `office.building.yaml` make sure the path to the `office.png` and to the `office_scan.png` is correct.

If you want, you can try with this map example of the custom office. Just move it to the `~/rmf_ws_custom_map/src/rmf_demos/rmf_demos_maps/maps/office/` directory.

Open a new terminal, source ROS 2 and source the workspace:

```
cd ~/rmf_ws_custom_map
source /opt/ros/jazzy/setup.bash
source install/setup.bash
```

Clean your workspace :

```
cd ~/rmf_ws_custom_map
rm -r install/ build/ log/
```

Build your workspace again and source it:

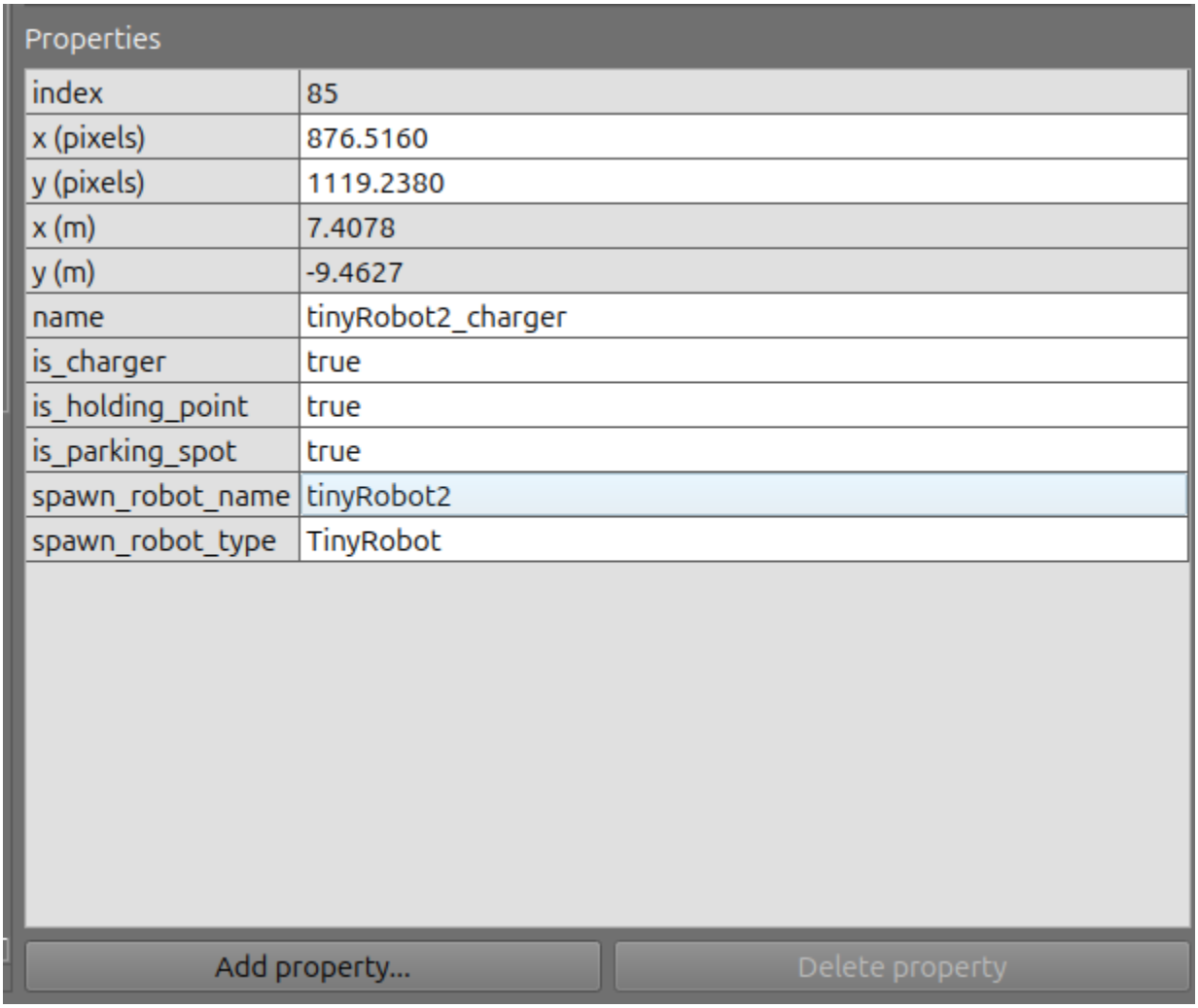
```
cd ~/rmf_ws_custom_map
colcon build
source install/setup.bash
```

If your build fails because of python `cmake`, then activate the virtual environment: `. venv_custom_maps/bin/activate`.

Now, when you run the office demo example, it will launch with your map:

```
ros2 launch rmf_demos_gz office.launch.xml
```

If you want to add robot nodes in the traffic editor, add the following properties to the node in the traffic editor by clicking on the node and clicking on `Add property`:



CHAPTER

FOURTEEN

TIPS ON THE CUSTOM MAP

14.1 Changes to the workspace

For every change you make on any file, you need to rebuild your workspace:

```
cd ~/rmf_ws_custom_map
colcon build
source install/setup.bash
```

If after the `colcon build` the changes you made did not work, you can try cleaning the workspace and building again:

```
rm -r log/ build/ install/
cd ~/rmf_ws_custom_map
colcon build
source install/setup.bash
```

14.2 No repeated node names in the map

Always make sure no two nodes have the same name. Otherwise, the lanes will not be loaded properly.

14.3 Robot `_config` files

For every robot on the scene, there must be a config for it. To do this modify the config file accordingly at `~/rmf_ws_custom_map/src/rmf_demos/rmf_demos/config/office`.

To add new robots to the map, you need to add a config file for the robot at `~/rmf_ws_custom_map/src/rmf_demos/rmf_demos/config/office`. For example, if you want to add a deliveryRobot to the office demo, copy the deliveryRobot_config.yaml from the hotel example located at `/home/marcos/rmf_ws_custom_map/src/rmf_demos/rmf_demos/config/hotel/deliveryRobot_config.yaml`. Just make sure that **all the robot names** inside the `deliveryRobot_config.yaml` are the same from the robot you add/spawn in your map nodes created in the traffic-editor. Check the `name`, the `robots` and their subproperties (for example, the `charger` in the case of the delivery robot). All the name related things should mirror the robot node from the map `.building.yaml` created with the traffic editor.

In this example, you may need to update the `tinyRobot_config.yaml` by adding the `tinyRobot2` and the `tinyRobot3`. You can copy the model that is already in the `tinyRobot_config.yaml` for the `tinyRobot1`.

If you do not want to open the traffic editor to check the properties of each robot node, they are also available in the `.building.yaml` file. For instance, check the following lines inside the office.building.yaml:

```
...
- [1549.2, 468.44600000000003, 0, tinyRobot1_charger, {is_charger: [4, true], is_holding_
↪point: [4, true], is_parking_spot: [4, true], spawn_robot_name: [1, tinyRobot1], spawn_
↪robot_type: [1, TinyRobot]}]
- [895.73500000000001, 251.386, 0, elevator]
- [1992.46, 632.95500000000004, 0, cubicle]
- [1704.5699999999999, 648.94899999999996, 0, tinyRobot3_charger, {is_charger: [4, true],
↪ is_holding_point: [4, true], is_parking_spot: [4, true], spawn_robot_name: [1,␣
↪tinyRobot3], spawn_robot_type: [1, TinyRobot]}]
- [747.221, 614.67600000000004, 0, office1]
- [2385.4520000000002, 413.61000000000001, 0, helpdesk]
- [876.51599999999996, 1119.2380000000001, 0, tinyRobot2_charger, {is_charger: [4, true],
↪ is_holding_point: [4, true], is_parking_spot: [4, true], spawn_robot_name: [1,␣
↪tinyRobot2], spawn_robot_type: [1, TinyRobot]}]
- [1513.9860000000001, 648.31700000000001, 0, deliveryRobot1_charger, {is_charger: [4,␣
↪true], is_holding_point: [4, true], is_parking_spot: [4, true], spawn_robot_name: [1,␣
↪deliveryRobot1], spawn_robot_type: [1, DeliveryRobot]}]
...
```

Last, after you have all the robot nodes properties, you can change the `_config.yaml` files of each robot. For instance, the `tinyRobot_config.yaml` will have the following lines modified (**note the names are exactly the same from the properties above**):

```
...
robots:
    tinyRobot1:
        charger: "tinyRobot1_charger"
        responsive_wait: False # Should responsive wait be on/off for this specific␣
↪robot? Overrides the fleet-wide setting.
    tinyRobot2:
        charger: "tinyRobot2_charger"
        # No mention of responsive_wait means the fleet-wide setting will be used
    tinyRobot3:
        charger: "tinyRobot3_charger"
...
```

and the `deliveryRobot_config.yaml` will have the following modifications:

```
...
robots:
    deliveryRobot1:
      charger: "deliveryRobot1_charger"
...
```

14.4 Add different fleets of robots

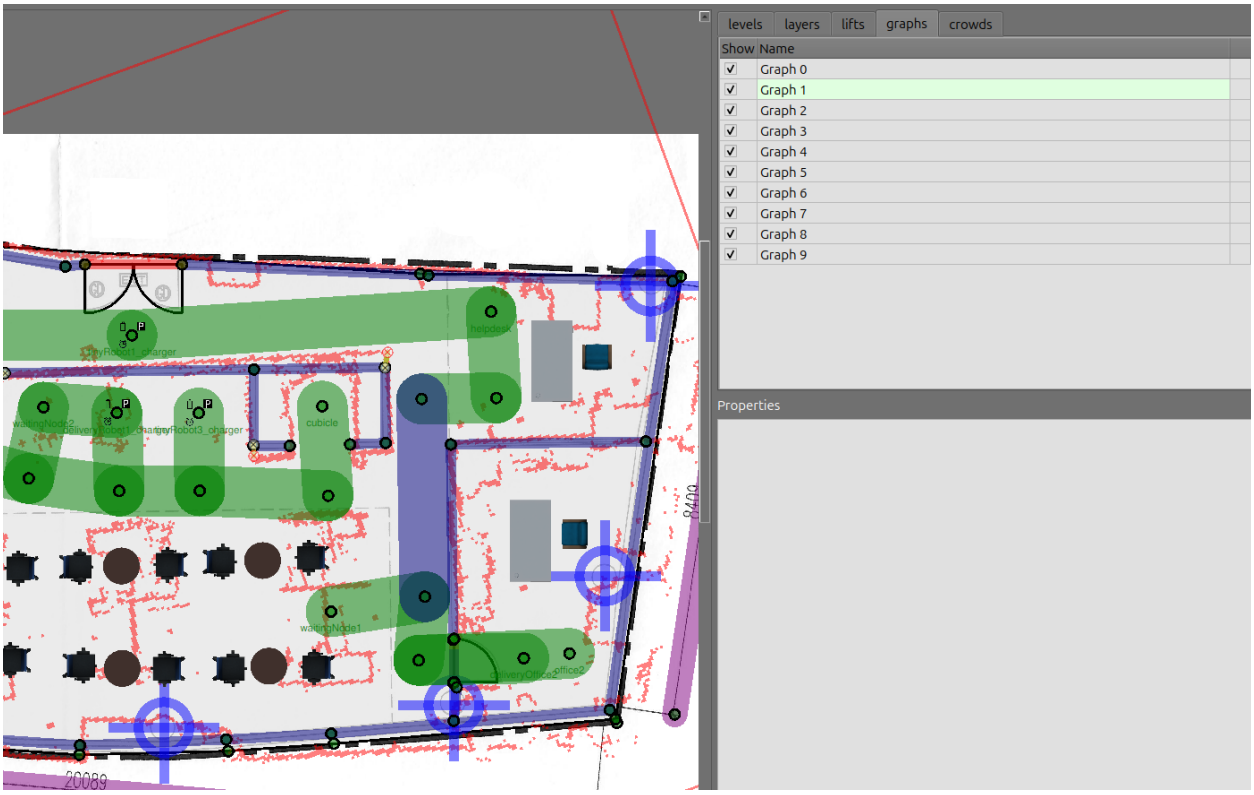
To add a new fleet with different robot types, call the new fleet in the file `~/rmf_ws_custom_map/src/rmf_demos/rmf_demos/launch/office.launch.xml` (we are using the office map as the example modified custom map). Do not forget to change the `nav_graphs/<fleet_number>.yaml` to the lane of the new fleet. **Note that the Open-RMF does not solve conflicts between two different lanes. Thus, if you have different lanes, make sure they will not cause conflicts or collisions.**

```
...
<!-- TinyRobot fleet adapter -->
  <group>
    <include file="$(find-pkg-share rmf_demos_fleet_adapter)/launch/fleet_adapter.launch.
↪xml">
      <arg name="use_sim_time" value="$(var use_sim_time)"/>
      <arg name="nav_graph_file" value="$(find-pkg-share rmf_demos_maps)/maps/office/nav_
↪graphs/0.yaml" />
      <arg name="config_file" value="$(find-pkg-share rmf_demos)/config/office/tinyRobot_
↪config.yaml"/>
    </include>
  </group>

  <!-- ADDING A FLEET OF DeliveryRobot -->
  <!-- DeliveryRobot fleet adapter -->
  <group>
    <include file="$(find-pkg-share rmf_demos_fleet_adapter)/launch/fleet_adapter.launch.
↪xml">
      <arg name="use_sim_time" value="$(var use_sim_time)"/>
      <arg name="nav_graph_file" value="$(find-pkg-share rmf_demos_maps)/maps/office/nav_
↪graphs/0.yaml" />
      <arg name="config_file" value="$(find-pkg-share rmf_demos)/config/office/
↪deliveryRobot_config.yaml"/>
    </include>
  </group>
```

14.5 Using different lanes on the map

If you want to add the fleet on a new lane/route different, you need to change the lane graph in the traffic-editor. To do this, click on the lane you want to change and press a number from 0 to 10. It should change the color of the lane graph. Each graph correspond to a number and a color. The blue lane is on graph 1. When calling the fleet adapter on the `.launch.xml` just change the `nav_graphs` to the layer you want (in this case `1.yaml`):


levels
layers
lifts
graphs
crowds
Show
Name
Graph 0
Graph 1
Graph 2
Graph 3
Graph 4
Graph 5
Graph 6
Graph 7
Graph 8
Graph 9
Properties
helpdesk
cubicle
waitingNode1
deliveryOffice
office2
20089
8400

CHAPTER

FIFTEEN

ISSUES

Some issues may happen. Here we leave some solutions.

15.1 The order of the `source` commands matters

When sourcing multiple workspaces, the last `source` command will take effect over the other ones. Thus, if follow the exact order of `source` commands the tutorial tells you. For instance, when *testing the custom map office*, if you source first the `rmf_ws_custom_map/install/setup.bash` and then the `~/rmf_ws/install/setup.bash`, the map that will be loaded is the one in the `rmf_ws`. Therefore, to run the custom map you need to source first the `~/rmf_ws` and last the `~/rmf_ws_custom_map`.

Last, to prevent any conflicts, always open a new terminal if anything stops working. Every time you open a new terminal, it opens without any previous `source` commands and you can start fresh again.

15.2 Adding new robots to the map

The traffic-editor should add everything you modify on the editor to the `.building.yaml` file. However, we had this issue on the first time the nodes for the new robots were added.

To add the robots to the map, you need to add them in the `crowd_sim` > `agent_groups` > `agents_names` property of the `.building.yaml` file manually in the `agent_groups` > `agents_name` > `[add_names_here_separated_by_commas]`. Also, change the `agents_number` to the number of agents you inserted in the list:

```
crowd_sim:
  agent_groups:
    # Add the agents_name to the list of agents_name
    - {agents_name: [tinyRobot1, tinyRobot2, <ADD_MORE_ROBOTS_HERE>], agents_number:
↪<NUMBER_OF_ROBOTS_USED>, group_id: 0, profile_selector: external_agent, state_
↪selector: external_static, x: 0, y: 0}
```

After changing things on the `office.building.yaml`, rebuild your workspace with `colcon build` and launch the map:

```
ros2 launch rmf_demos_gz office.launch.xml
```

Open a new terminal and send the tasks (this an example, you need to adapt it to your map):

```
ros2 run rmf_demos_tasks dispatch_patrol -p cubicle office2 -n 3 --use_sim_time
ros2 run rmf_demos_tasks dispatch_patrol -p deliveryOffice2 -n 3 --use_sim_time
```