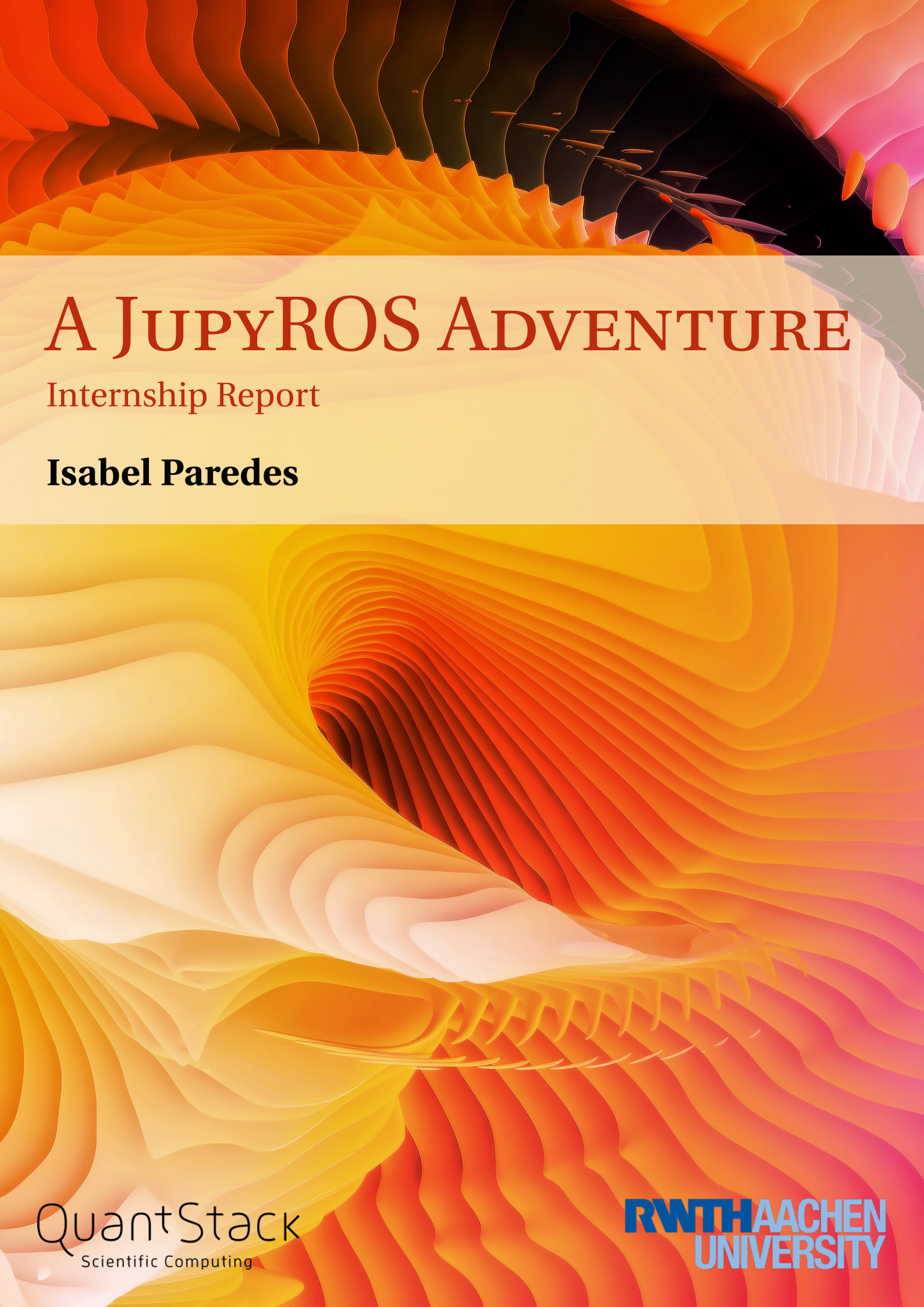


A JupyROS Adventure

Internship Report

Isabel Paredes

QuantStack
Scientific Computing

RWTH AACHEN UNIVERSITY

Contents

Preface

Week 01

1.1 ROS Service and Client

1.1.1 Service

The first task for this week was to expand the functionality of `jupyros` by adding support for ROS services and clients. A service can be easily initialized with the help of the `rospy` library.

```
srv = rospy.Service("add_two_ints", AddTwoInts, handle_add_two_ints)
```

`AddTwoInts` defines the message type for the request and the reply.

```
int64 a
int64 b
---
int64 sum
```

1.1.2 Client Widget

In order to make the interface similar to the already existing publisher and subscriber widgets, a function to automatically generate a client widget was defined.

```
jupyros.client("add_two_ints", AddTwoInts)
```

Figure 1.1 illustrates the output widget generated for the message type shown above. The number and type of input cells will vary depending on the message definition. When the `Call Service` button is pressed, this triggers a request to the specified service which will then return the response.

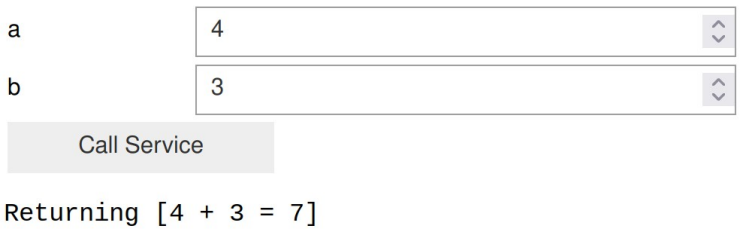


Figure 1.1: Client Widget for an `AddTwoInts` message type

The client widget was also tested with other service message types such as the `Spawn.srv` from the `turtlesim` package. The client widget function successfully generated the expected results. The source code for the client function can be found here.

1.1.3 Pull Request

Once tested, a pull request was created. Additional changes were made after receiving feedback. This marked my first official contribution to `jupyros`.

1.2 Turtlesim I

1.2.1 ipycanvas Turtles

This project consisted of replicating the illustrious turtle simulation within a Jupyter notebook. The first step was to display a turtle image with the help of `ipycanvas` library. All the turtle images can be found in the standard `turtlesim` package.

```
from ipycanvas import Canvas
canvas = Canvas(width=1600, height=1200, layout={"width": "100%"})

# Water
canvas.fill_style = "#4556FF"
canvas.fill_rect(x=0, y=0, canvas.width, canvas.height)

# Turtle
turtle_canvas.draw_image(turtle_img, width=100)
```

1.2.2 Random Turtles

Given that the `turtlesim_node` displays a different turtle every time it is initialized, this behavior was imitated by storing the different turtle names in a list and selecting a random index to pull from that list.

```
turtle_path = turtle_img_path +
              turtle_names[randint(0, len(turtle_names)-1)] + ".png"

turtle_img = Image.from_file(turtle_path)
```

1.2.3 Rotating Turtles

The turtle images by default face "upward" or towards the top of the page. However, for an initial orientation of 0° the turtles must face the right side of the page. To accomplish this, the canvas is first translated to the desired x and y position and then rotated by θ before drawing the turtle image. Once drawn, the canvas transformation is reversed.

Figure 1.2: Spawned turtle with an orientation of 0°

```
canvas.translate(x, y)
canvas.rotate(radians(90))
canvas.draw_image(turtle_img, width=turtle_size)
canvas.rotate(radians(-90))
canvas.translate(-x, -y)
```

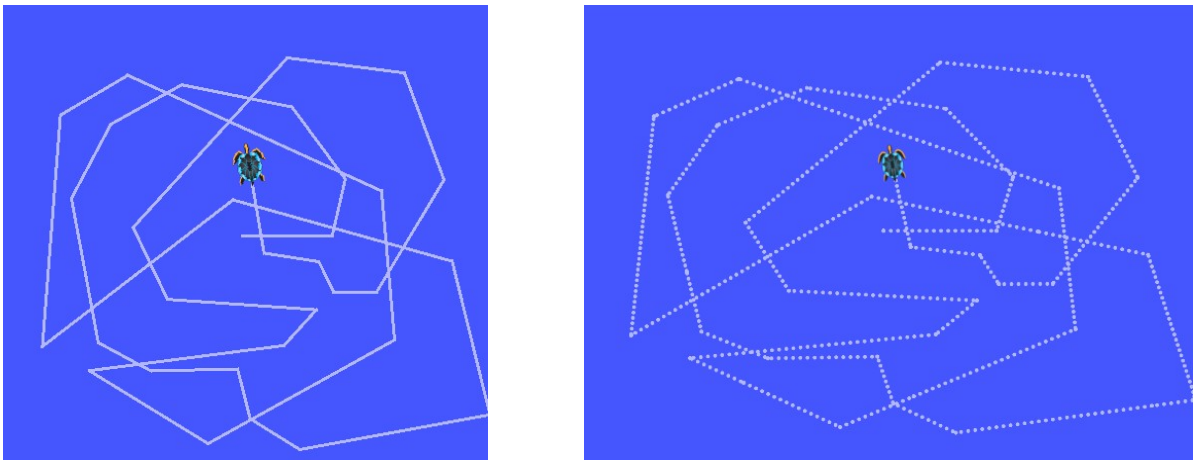
It must be noted that in the canvas coordinate system the $z-$axis points into the page, whereas, in the `turtlesim` coordinate the $z-$axis points in the opposite direction out of the page. The result from spawning a turtle in the middle of the canvas with an initial orientation of 0° can be observed in Figure 1.2.

1.2.4 Animating Turtles

The next step in the process was to create an animation function to move the turtle around the canvas. `ipycanvas` already provides the necessary tools for creating an animation, however, this created some issues with flickering of the image since the canvas was fully cleared during each iteration. To speed up the animation and to avoid the flickering effect, a `MultiCanvas` approach was used. The canvas consisted of four layers: the background, the path, and the last two for turtle motion. In this manner, the background and path can remain static while the turtle image is cleared only when there is another turtle image already drawn.

1.2.5 Tracing a Path

Lastly, the path the turtle traverses was created by a series of markers. Each time the turtle moves to a new position, a small circular marker is drawn at the coordinates of the new position. Because the canvas is multi-layered, the canvas where the markers are placed is never cleared, thus, they form a path as the turtle moves around the canvas. The results of the animation are shown in Figure 1.3b.



(a) Turtlesim original path

(b) Turtle path drawn with canvas markers

Figure 1.3: Comparison of the turtle path of ROS Turtlesim and the turtle path drawn with `ipycanvas`

1.3 Future Work

The next step for this project is to create a turtle widget class. This will allow the storage of important information about the turtle such as the turtle image, the turtle's name, and its current position on the canvas. By having access to the turtle's position, the canvas path markers can be replaced with lines which will more closely resemble the original `turtlesim` path.

Week 02

2.1 PyCon

This week mainly consisted of attending PyCon DE and PyData Berlin. This was a three-day event held at the Berlin Congress Center (Figure 2.1). During the conference, there were several talks conducted by Python experts. The following are a few highlights from the talks attended:

- *Trojan Source Malware* by Cheuk Ting Ho → It is a bad practice to blindly copy code from the internet because there could be hidden ASCII characters which may result in undesirable program behavior.
- *Your Data, Your Insights* by Paula Gonzalez → Developing projects to visualize one's own personal data is another way to rekindle the joy for data science.
- *Can You Read This?* by Asya Frumkin → To aid people with visual impairments, it is the duty of developers to consider text readability on uniform and non-uniform backgrounds.
- *The Magic of Python Objects* by Coen de Groot → Magic or Dunder methods can be overridden in a custom class to create overloaded behaviors.
- *JupyterLite* by Jeremy Tuloup → JupyterLite is another version of JupyterLab which runs entirely on a web browser without the need to install anything.

Figure 2.1: Photograph of BCC by Ernesto Arbitrio

2.2 Turtlesim II

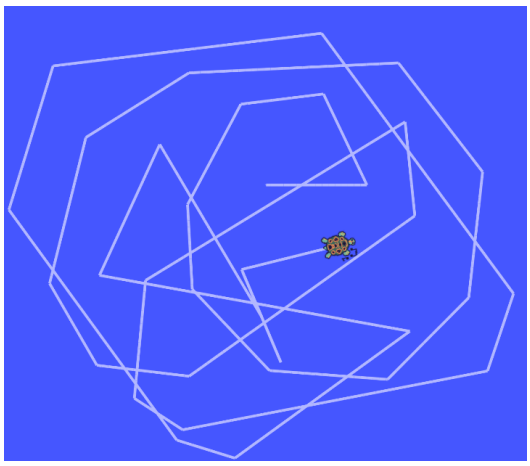
To continue the work from last week, a `TurtleWidget` class was created. This class contains the following methods:

- `randomize` → this method selects a random turtle image with the exception of the turtle named *palette* because this image does not appear to be a turtle.
- `spawn` → this method spawns a turtle in the middle of the canvas if no specified spawning pose is given.
- `move_to_pose` → this method draws the turtle at the new pose given and generates a line from the old position to the new position.
- `draw_turtle` → this method transforms the canvas to be able to draw the turtle at the specified position and orientation.

Initially, some flickering issues had been observed while performing the animation, this had been resolved by using two canvas layers dedicated to the turtle so that there could always be a turtle visible on the overall canvas. However, the flickering was no longer observed after creating the class, thus, the multi-canvas was reduced to only three layers: background, path, and turtle.

The generated linear path appears to be working correctly as is illustrated in Figure 2.2a. However, if the turtle takes a sharp turn, the intersection of the lines is not as smooth as it could be. This could potentially be resolved by dropping a circular marker at every turn but it could also slow down the performance.

Lastly, a spiral path publisher was created to generate the path seen in Figure 2.2b. This publisher follows the Archimedean spiral equations to traverse five loops from the center of the canvas.



(a) Linear path created from saving previous position.

(b) Path drawn by the spiral publisher.

Figure 2.2: Turtle paths

2.3 Future Work

For the next steps, it would be useful to be able to call a spawn service to display new turtles on the canvas. This could be implemented by adding an additional canvas layer for every new turtle, or by simply storing the information for each turtle separately and keeping all the turtle images on the same layer.

Week 03

3.1 Turtlesim III

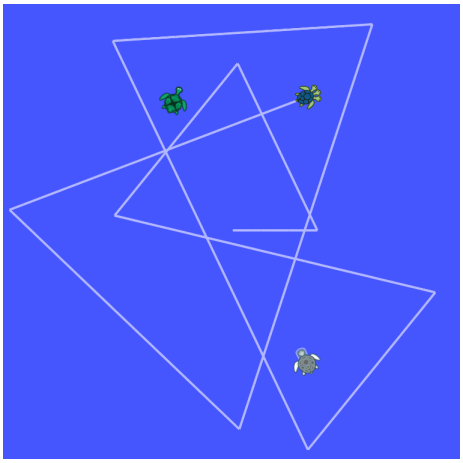
3.1.1 Spawning Turtles

To continue last week's project, more features were added to the Turtlesim animation. A new `Turtle` subclass was created to more efficiently organize and store all the attributes of each turtle. With this `Turtle` class, it was easier to implement the spawn service by simply adding a new turtle object to the Turtlesim widget.

3.1.2 Path Tracing with Multiple Turtles

Previously, it was not possible to draw multiple turtles on the canvas because when running the animation the turtle layer was cleared and only the pose of the first turtle was stored; this caused any other turtles to disappear. With the new addition however, the poses of each turtle can be more easily accessed and the animation can loop through all the turtles created and draw them on their respective canvas layer.

(a) Spawning more turtles after drawing a path



(b) Drawing a path after spawning multiple turtles

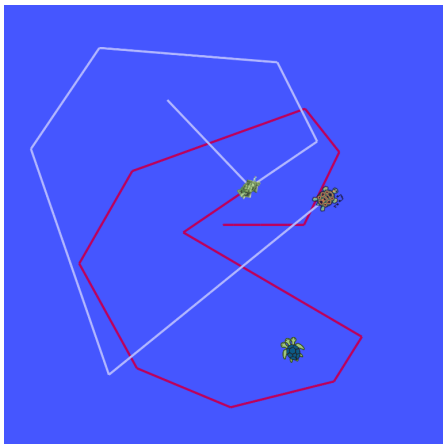
Figure 3.1: Spawning service

The results are depicted in Figure 3.1a where the spawn *service* is called multiple times after a turtle has traced a path throughout the canvas, and in Figure 3.1b where the spawn *method* is directly called before animating a turtle. The accomplishment here is that the turtles and the traced path remain on the canvas regardless of the order in which they are generated.

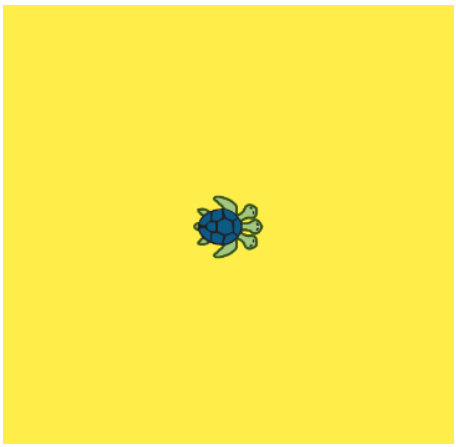
3.1.3 Path Coloring

In order to be able to distinguish the turtles on the canvas, it was important to give each turtle its own unique and customizable characteristics such as path color. This was easily accomplished by adding another attribute to the `Turtle` class. The different path colors can be observed in Figure 3.2a. Additionally, the canvas color can also be configured during initialization as illustrated in Figure 3.2b.

```
turtlesim = jupyros.TurtleWidget(background_color="#FFED4A")
turtlesim.turtles["turtle2"].path_color = "#BF0059"
```



(a) Customizing the path color of a turtle



(b) Specifying the background color of the canvas

Figure 3.2: Turtle widget customization

3.2 tf2 Transforms

3.2.1 Follower

To start introducing navigation into jupyros, the `turtle_tf2` tutorial demo was replicated with a canvas animation. This demo uses the tf2 library to broadcast the leading turtle's coordinate frame and to listen for these frames so that the second turtle can compute the differences and chase the first turtle.

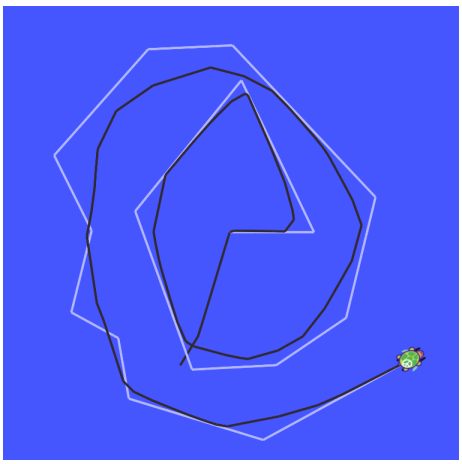


Figure 3.3: The leading turtle traces the light path while the follower traces the dark path.

Several issues were encountered when creating this animation. The biggest hurdle was to update the canvas at the fastest possible rate. However, the `draw_image` function takes a short amount of time to draw an image on the canvas; if that time is not provided, the turtle images will not appear on the canvas. To partially alleviate this, the turtle images were stored as canvas objects because they can be drawn quicker on other canvases. As a further remedy, a time counter was integrated into the `TurtleWidget` class so that the canvas can only be updated every 0.1 s. It was determined that this was the shortest amount of time needed two draw both turtles moving.

3.3 Future Work

It would be useful to add examples of how to change a turtle's path color by publishing the specified color to the topic `/turtle1/color_sensor`, and to change the background color by adjusting the ROS parameters.

Also, more content involving the tf2 library needs to be included. The next steps will involve implementing a broadcaster and a listener directly from Jupyter. The ability of Zethus to display transforms needs to be explored as well. And a more elegant solution to the time constraint for drawing images should be implemented, this can potentially be solved from within the ipycanvas library itself.

Week 04

The work continues with `jupyros` development. For this week, the focus was on navigation or being able to set up a navigation stack for any robot. The goal is to be able to visualize everything involved with navigation inside JupyterLab, this includes any sensor streams, maps, path planning, etc.

4.1 tf2 Broadcaster

Following the ROS tutorial examples, it was simple task to implement a tf2 broadcaster within Jupyter. As of now, additional widgets for tf2 broadcasters and listeners does not appear to bring any immediate benefits.

4.2 Panda Transforms

As mentioned last week, Zethus is capable of displaying `tf` data. This was tested with Franka Emika's panda robot as seen in Figure 4.1. A few issues were encountered when trying to publish the robot description, this was because the `panda_arm_hand.urdf` was renamed to `panda_arm.urdf` but this was fixed within the launch file.

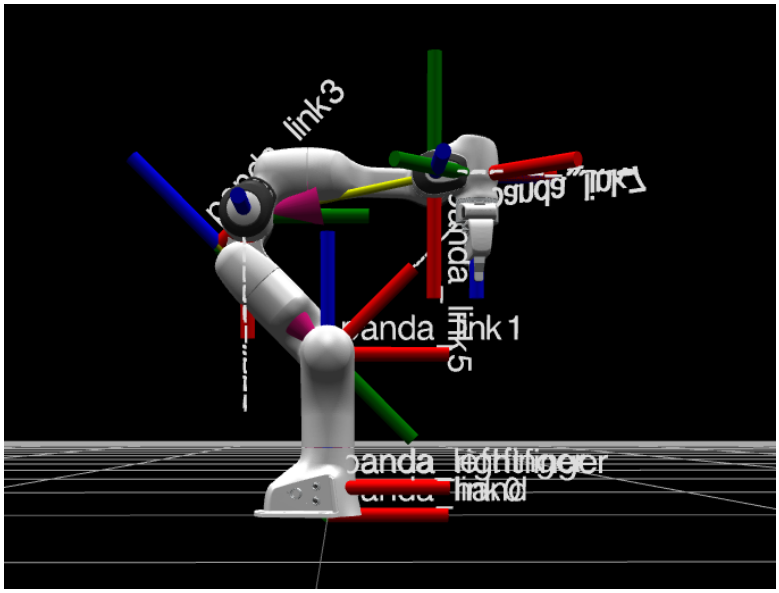
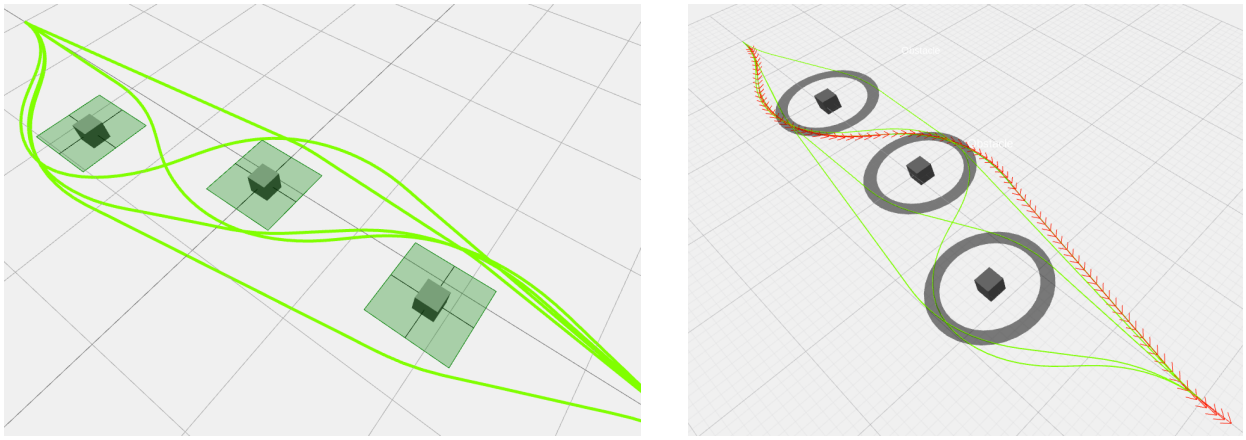


Figure 4.1: Transformations of Franka Emika's panda robot as displayed by Zethus

4.3 Interactive Markers

An example for using for using interactive markers and displaying them as a Jupyter widget was already included in the `jupyros` repository. However, this notebook used a `interactive_marker_proxy` package which was archived by the authors. Thus, the package had to be built from source and following the suggestions of Witalij Siebert's pull request, the package was adapted to work with `tf2` and be compatible with ROS noetic.



(a) Zethus display **(b)** RViz display

Figure 4.2: Interactive markers

As illustrated in Figure 4.2b, Zethus is capable of representing the interactive markers and the possible paths. These markers can be easily moved with the mouse. The Zethus display can be compared to the RViz display of the same markers in Figure 4.2a. Currently there is no synchronization between Zethus and RViz, in other words, if one of the markers is moved to a new position in Zethus that new position will not be reflected in RViz and vice versa. However, synchronization may not be necessary because at the moment Zethus is meant to act as an RViz replacement.

4.4 Laser Scanner

To prepare for the addition of sensors to the simulation, the ability of Zethus to display laser scans was examined. Without access to a live laser scanner, a *fake* scanner was created. The generated fake scan can be observed in Figure 4.3. The size of the points here is exaggerated for display purposes, but this along other attributes can be directly modified with Zethus.

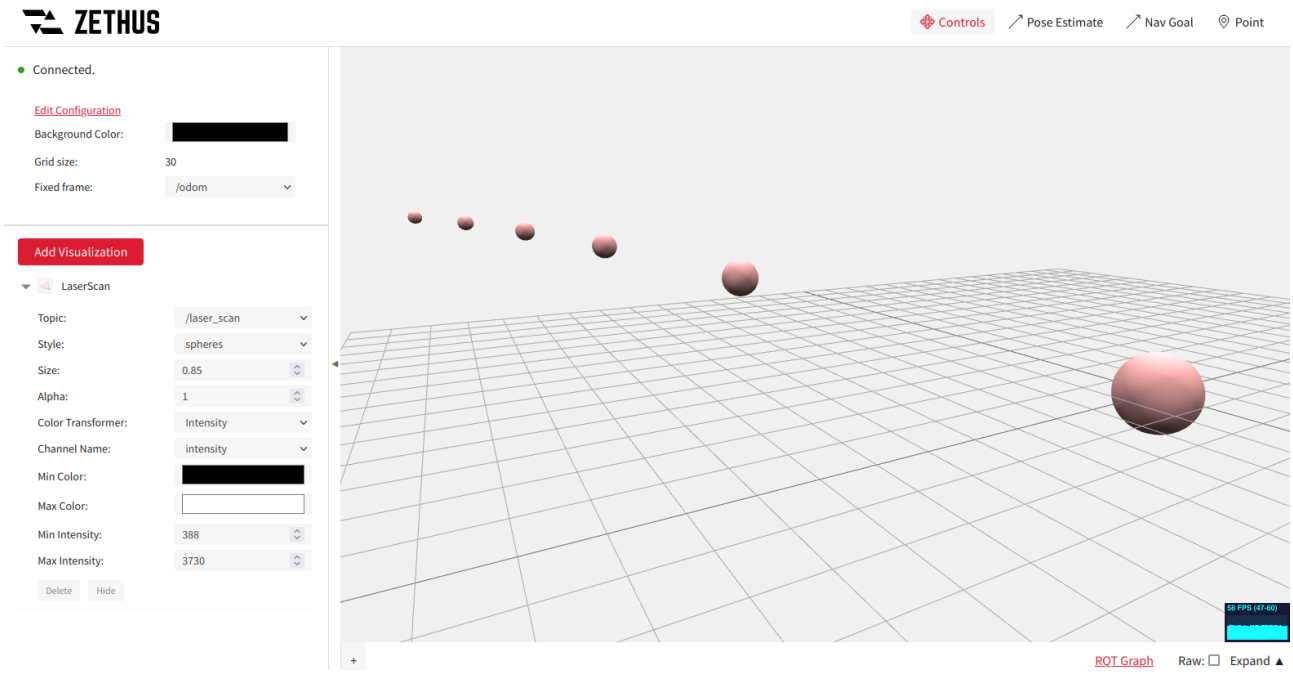


Figure 4.3: Display of a fake laser scan with Zethus

4.5 Gazebo

Since robot simulations are an essential part of the robotics ecosystem, adding support for a Gazebo simulator became the next logical step. Fortunately, a web client for Gazebo has already been developed by Open Robotics which is called GzWeb. The functionality of this web simulator needed to be tested to determine if it could augment the capabilities of `jupyros`.

The testing process began by installing and building GzWeb. Initially there were a few issues with the build, but these were resolved by installing the missing dependencies and the supported versions of `nodejs` and `npm`. Afterwards, GzWeb ran as expected on a web browser.

The next step was to display different world models to test the limitations of GzWeb. From this it was discovered that some colors and textures are not displayed correctly when compared to the classic Gazebo simulator; this issue requires further investigation. Figure 4.4 shows a simulation of an underwater world displayed in a web browser.

A few additional issues were encountered during testing, mainly involving the left panel. When the light gray buttons are pressed to add more models or display the tree view, the entire panel disappears.

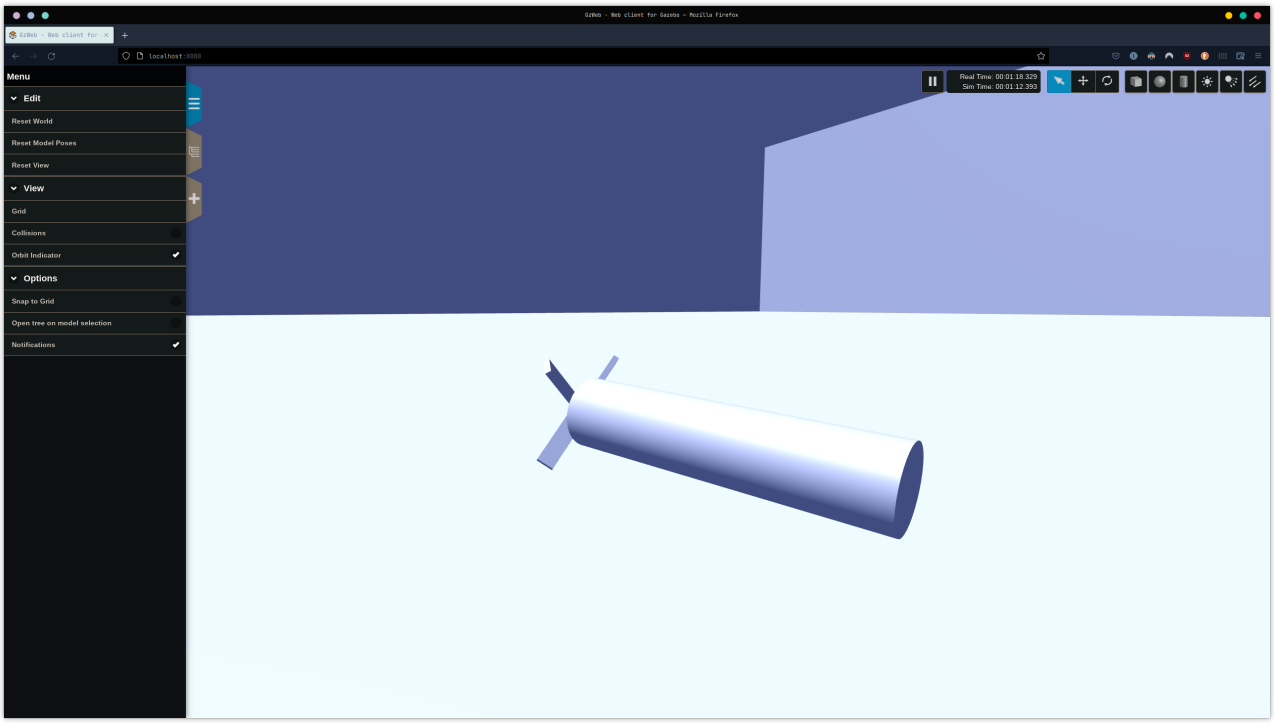


Figure 4.4: Display of an underwater world with Gazebo's web client

4.6 Future Work

Zethus provides other visualization options such as image streams, maps, and point clouds. The limitations of these visualization types needs further exploration.

Similarly, GzWeb also requires more investigation; the display issues can probably be solved by configuring the resource paths correctly. Additionally, some form of communication between GzWeb and the Jupyter ROS environment needs to be established in order to simulate robots which are managed from within a Jupyter notebook. And as a final step, GzWeb needs to be integrated into JupyterLab in the same manner that Zethus is integrated.

Week 05

For this week, there was more progress achieved with the turtle simulations, and the process of incorporating Gazebo into JupyterLab was commenced.

5.1 Turtlesim IV

The turtle widgets were successfully merged into the `jupyros` repository (PR #94) after some minor changes which involved creating a separate file for the `TurtleSim` class. Additionally, thanks to the newest `ipycanvas` release 0.12.0, the frame rate of the turtle animations was able to improve by 0.02 *s* (PR #97).

5.2 Gazebo II

To test if a connection to Gazebo was possible without the GUI, the idea of running a headless Gazebo was explored. However, after several failed attempts it was discovered that the `"headless"` argument is deprecated (`gazebo_ros_pkgs` issue # 491) and what was actually needed was simply to set `"gui"` to False.

The next test comprised of displaying a robot on GzWeb. With the classic Gazebo application and with the help of the Gazebo ROS packages, a user can simply launch a file which will automatically spawn a robot and any desired world in the Gazebo application. Technically, this should also be possible with the GzWeb client if the communication to the gzserver is properly established. To confirm this, the Gazebo packages for the panda robot and the turtlebot3 were utilized. Initially, GzWeb was unable to display the robots because it could not locate the robots' description packages. As a temporary fix, these description packages were manually copied to the `assets` directory of GzWeb. Nonetheless, the test failed for the turtlebot3 but it partially passed for the panda robot. The panda robot was able to spawn in GzWeb but it required heavy manipulation of the default launch file to avoid all the errors.

Additionally, it was discovered that GzWeb works better with Chrome than with Firefox. As mentioned last week, the tab buttons on the left panel make the entire panel disappear without sign of return when used with Firefox, however, these buttons are fully functional in Chrome. With a fully functional panel, a turtlebot was finally able to be displayed on GzWeb, but the material textures were still not loading correctly.

For display purposes, additional time was dedicated for building the thumbnails for the models. With the thumbnails, the menu on the left panel for adding new models to GzWeb can display a small preview of each model. Several attempts were required to build the thumbnails, however, once the process was completed a few thumbnails were still not able to be generated.

5.3 JupyterLab Extensions

In order to prepare for integrating Gazebo into JupyterLab, I needed to acquire more knowledge in the development of JupyterLab extensions. For this, I followed some of the basic extension examples provided on the JupyterLab documentation.

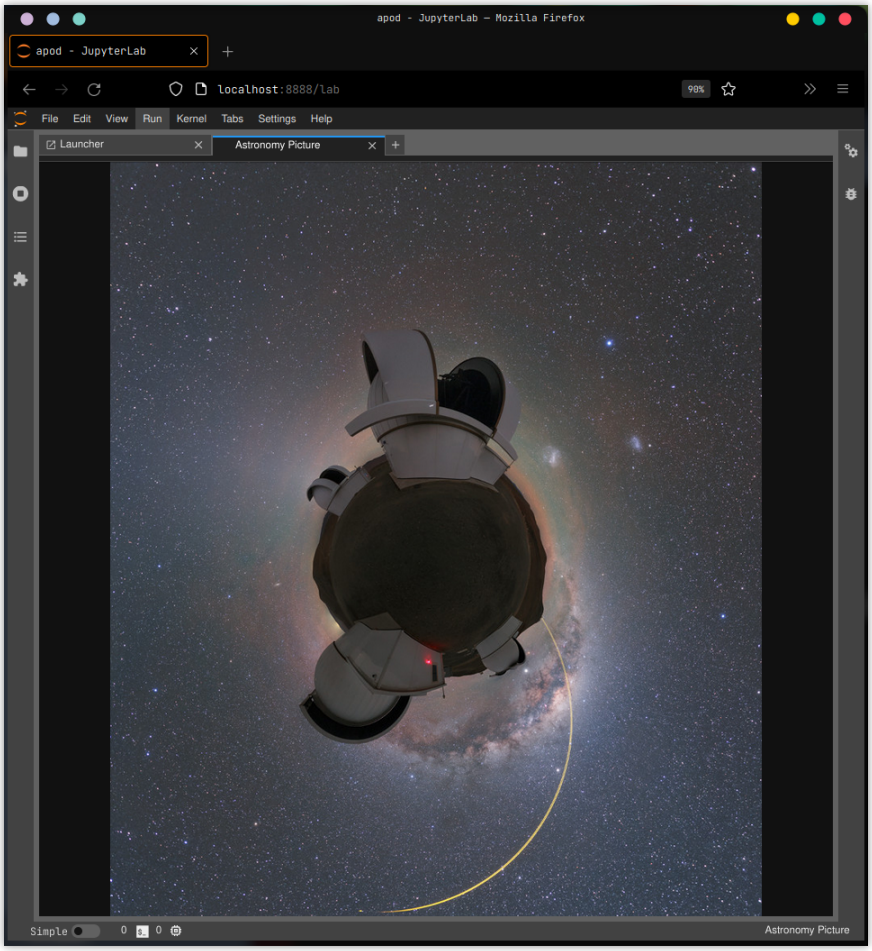


Figure 5.1: JupyterLab extension displaying a random astronomy picture.

5.3.1 Astronomy Picture of the Day

The Astronomy Picture of the Day or `apod` shown in Figure 5.1 is the result of one of the extension examples. With this example I learned to add a command to the command palette, fetch information from the internet, display that information on a new tab panel, and how to style such a panel.

5.3.2 TypeScript

To familiarize myself with TypeScript, I complete all the tutorials from the W3Schools. These covered all the basics of variable types to functions and classes. Although more practice is required to become competent in the language, the tutorials were sufficient to follow through the JupyterLab extension tutorials.

5.4 Future Work

Although there are several issues with the GzWeb client, the next steps will focus on the development of the Gazebo extension for JupyterLab. Ideally, the issues with GzWeb will be resolved along the way. The solutions will include automating the process for linking the desired media files to GzWeb, this could be accomplished by updating the Python 2 scripts which come with the application. Along the same lines, the generation of thumbnails for the models could also be simplified.

Week 06

The focus of this week was to integrate Gazebo into JupyterLab. The process involved four main steps: creating a new tab for Gazebo, building the website, displaying the website inside the created widget tab panel, and the consolidation with `jupyros`.

6.1 Gazebo Extension I

6.1.1 Widget Tab Panel

By following the widget extension example from JupyterLab, a new tab panel was created for Gazebo. With these modifications, the Gazebo tab can only be opened from the command palette. The appearance of the new panel is shown in Figure 6.1. The background of the panel was adjusted in the `css` file for illustration purposes.

```
class GazeboWidget extends Widget {
  constructor() {
    super();
    this.addClass('gazebo-view');  // Used for css
    this.id = 'gazebo-widget';
    this.title.label = 'Gazebo';
    this.title.closable = true;
```

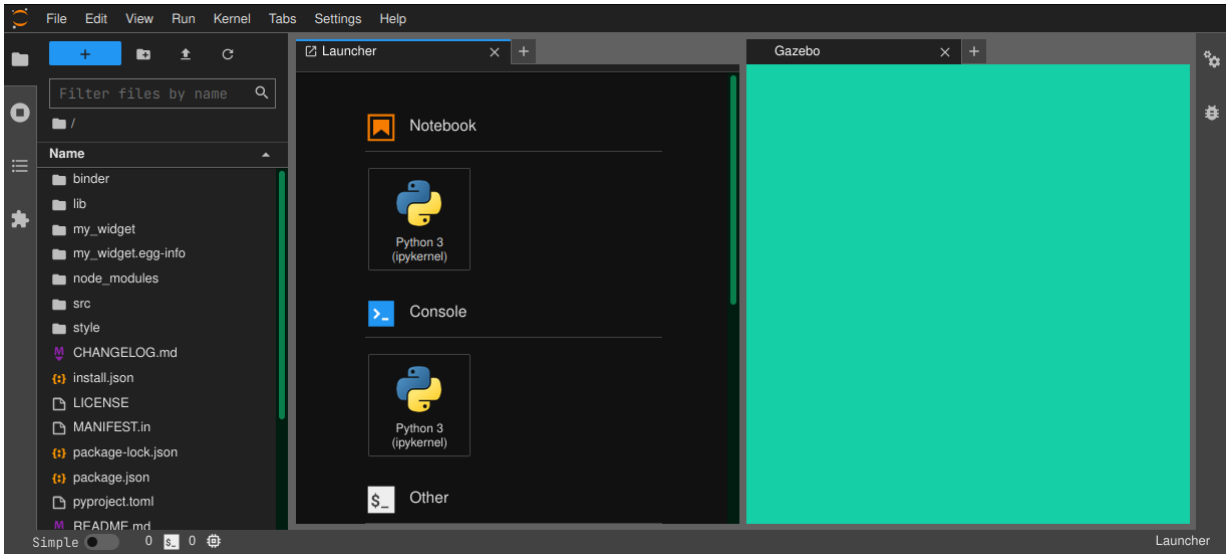


Figure 6.1: A Gazebo widget tab inside JupyterLab

6.1.2 Gazebo Website

Having previously built GzWeb, all the built files were simply copied and pasted into the `jupyterlab-gazebo` extension directory. The website displayed most of the components correctly in its new location with the exception of the model thumbnails once again. Although the thumbnails were in the correct folders, configuring the correct paths to the thumbnails required further investigation.

6.1.3 Gazebo Widget

The next step consisted of putting the Gazebo website in an `IFrame` so that it could be displayed inside the tab panel. This step was mainly accomplished by following the configuration of Zethus. A new menu item was also included to make the opening and closing of the Gazebo view more easily accessible. Similar to Zethus, the URL for Gazebo was also configured so that the website could be accessed at `http://localhost:8888/gazebo/app/index.html`. The results can be observed in Figure 6.2.

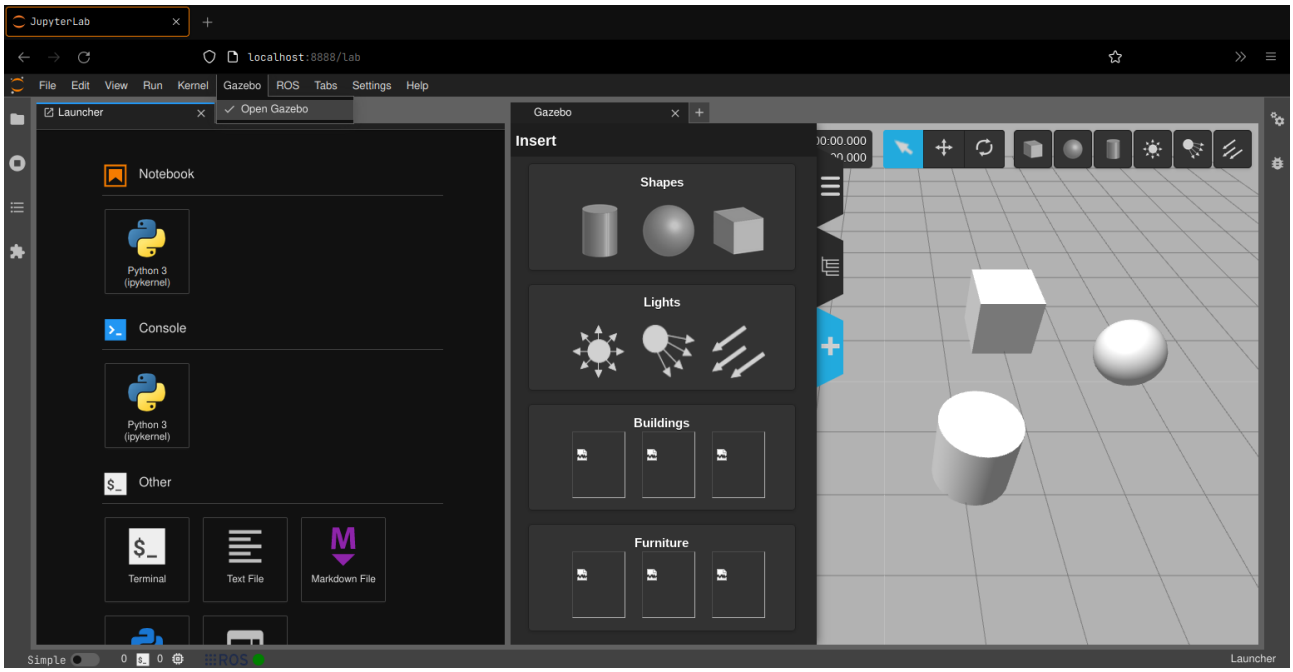


Figure 6.2: Display of Gazebo extension in JupyterLab

6.1.4 Integration with jupyros

Including the Gazebo widget as part of the `jupyros` tools required the simultaneous modification of the `jupyterlab-ros` package and the `jupyterlab-gazebo` extension. The `jupyterlab-ros` package creates the ROS menu and this is where a new list item needed to be added to open Gazebo as part of the ROS tools. And the Gazebo extension needed to be modified to reflect the change in the menu by removing the stand-alone Gazebo menu.

Although the idea was very simple, the execution was very involved. To begin, a new environment needed to be created where the Gazebo extension and `jupyterlab-ros` were in development mode. I also attempted to set `jupyter-ros` and `gzweb` in development mode within the same environment. Needless to say, no environment could be created which satisfied all the dependencies. Thus, the integration with `jupyros` remained pending.

6.2 Future Work

Much work is left to be done to complete the Gazebo extension, this will include fixing the thumbnails issue, automating the build for GzWeb, and most importantly testing with ROS. Additonally, it would be a good idea to be able to export the ROS menu from the `jupyterlab-ros` extension to be able to add a new item from the Gazebo extension; this is in anticipation to future JupyterLab ROS extensions.

Week 07

The main goals of this week were to continue the work on the Gazebo extension and to begin testing `jupyros` with physical robots such as the Niryo.

7.1 Gazebo Extension II

7.1.1 Development Environments

By far, the greatest challenge in developing JupyterLab extensions has been to set up the development environment with all the tools necessary for testing the extension. Learning from the mistakes from last week, I attempted to set up an environment where only the Gazebo extension was in development mode. The only required dependencies for this environment were the core ROS packages, `jupyros`, `jupyterlab-ros`, and Gazebo. Initially, the environment was created successfully. However, it was quickly discovered that Gazebo could not be launched. Every time Gazebo was initialized, it would immediately crash with a segmentation fault.

```
process[gazebo-2]: started with pid [87991]
process[gazebo_gui-3]: started with pid [87995]
Segmentation fault (core dumped)
[gazebo_gui-3] process has died [pid 87995, exit code 139, cmd
    /home/user/miniconda3/envs/gazebo/lib/gazebo_ros/gzclient __name:=gazebo_gui
```

After several failed environments, it was discovered that Gazebo could only be launched when the environment included Python 3.8 and that Gazebo would fail in any environment with Python 3.9. However, the Gazebo extension could not be developed with Python 3.8 because the latest release of `jupyterlab-ros` requires Python 3.9. The best course of action was to figure out the root cause of the segmentation fault. With plenty of help from the experts, it was determined that segmentation issue was linked to the `qt` package version. Gazebo required the very specific version of `qt` = 5.12.9 = hda022c4_4.

Once the dependencies issues were resolved, the environment was created successfully. A few minor changes needed to be made on the Gazebo extension before the setup was completed, these included modifying the Gazebo menu so that the "menu" plugin would not conflict with the already existing ROS menu from `jupyterlab-ros`.

7.1.2 GzWeb Issues

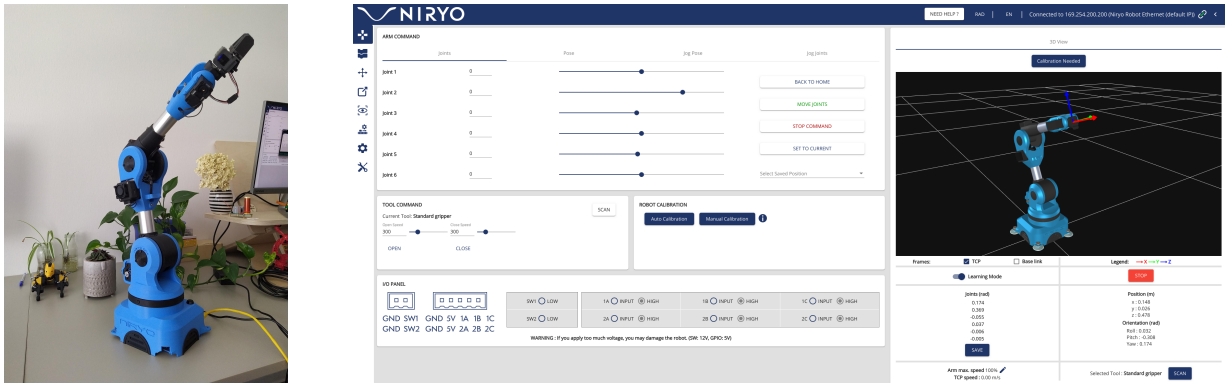
From the created environment, the functionality of GzWeb was tested once again. One of the first errors that occurred is that GzWeb was not able to receive information from the Gazebo server. The reason for this is that `gzbridge` was not running; the function of this Gazebo bridge is to facilitate the communication between the browser client and the Gazebo server. In order to run `gzbridge`, the GzWeb application needed to be rebuilt. For the application to work in the new environment, the rebuilding process demanded several updates to make it compatible with newer releases of `nodejs` and `npm`. More expert help was required to accomplish this step.

After all was *set* and done, `gzbridge` could be initialized but the Gazebo website would not display correctly. The error below was encountered, meaning that the `js` files which are vital for the Gazebo website were being blocked. This issue necessitated further investigation.

```
Cross-Origin Read Blocking (CORB) blocked cross-origin response
http://localhost:8888/login?next=%2Ffiles%2Fgz3d.gui.js with MIME
type text/html. See https://www.chromestatus.com/feature/5629709824032768
for more details.
```

7.2 Niryo One

In preparation for using the Niryo One robot in future experiments, the robot needed to be reset and reconfigured. This endeavour consisted of reinstalling the Raspberry Pi image, configuring the Ethernet and WiFi interfaces, and testing the joints range of motion (Figure 7.1).



(a) Niryo One robot **(b)** Niryo Studio

Figure 7.1: Calibration of the Niryo robot through an Ethernet connection to the Niryo Studio

7.3 Workshops

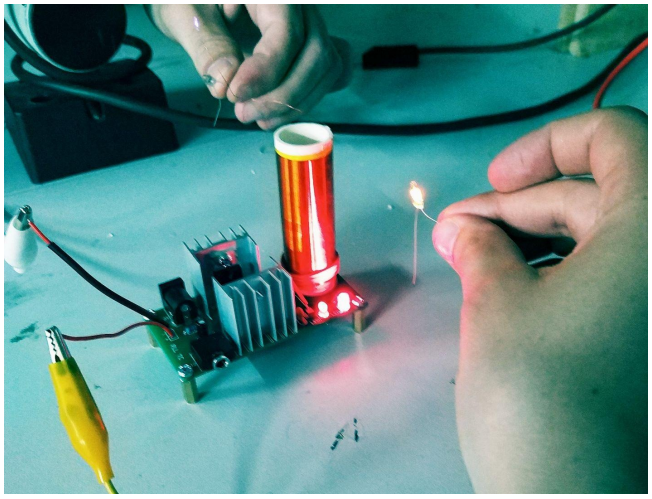


Figure 7.2: Tesla coil built during the electronics lab workshop

As part of my extracurricular education, I attended two makerspace workshops this week. The first workshop was related to the use of the electronics laboratory, during which basic skills such as soldering and using electronic measuring systems like multi-meters and oscilloscopes were taught by building a miniature Tesla coil (Figure 7.2). The second workshop was focused on laser cutting

techniques, during this workshop I learned about laser-cutting materials, design, and the best practices for adjusting equipment settings. The skills learned are vital for development of robotic prototypes and to augment the Niryo One and Bittle robots in the laboratory.

7.4 Future Work

Given the number of issues with the Gazebo extension, its development will continue. It will be necessary to fix the CORB issue to properly display the Gazebo website again. Afterwards, the extension still needs to be tested with a running Gazebo bridge and server. And furthermore, there is plenty of experimentation and testing left to do with Niryo and Bittle.

Week 08

8.1 JROS Profile

After much anticipation, the JROS profile has been added to the RWTH JupyterHub. And as expected, the extensions and packages needed to work with ROS were not correctly installed. To elaborate, when a new profile is spawned, neither the `jupyter-ros` nor the `jupyterlab-ros` extensions are activated and none of the ROS packages are installed in the default environment.

This issue was caused because in the `Dockerfile`, a new conda environment was being created where the extensions and packages were installed but this new environment was never activated. The solution was to install the required packages in the base environment instead.

Once the packages were installed in the base environment, a newly spawned profile was able to access them. However, a new issue emerged because the ROS commands were not available and none of the ROS environment variables were set. In a local environment, this would indicate that the `setup.bash` file needed to be sourced before running any ROS commands. But including the sourcing as an instruction in the `Dockerfile` did not accomplish the desired outcome because that would make the environment variables only available during the building process and they would be reset once the container was running. Further investigation is required to solve these issues.

8.2 Gazebo Extension III

The development of the Gazebo extension continued starting with fixing the CORB issue encountered last week. The following actions were taken to successfully clear the error:

- A new development environment was created from scratch.
- JupyterLab was removed from the base environment and only installed in the development environment. It can become problematic when multiple environments share the same JupyterLab configuration.

Additionally, a server was added and configured for the extension. This differs from the Gazebo server which will run as a separate process. When the extension is installed, the server extension should also be enabled.

```
$ jupyter serverextension enable jupyterlab_gazebo
```

8.2.1 Testing

With the Gazebo client, bridge, and server in working condition, it was time to test the models which could be simulated with the extension. There were two ways of accomplishing this, one was to use launch files to spawn specific models and the other way was to add models directly from the client's menu. The first approach failed because the extension still has issues finding the correct path to the model files; it may be necessary to manually reconfigure the assets path when building the client. However, the second approach was successful as seen in Figure 8.1. The same simulation is also shown in Figure 8.2 with the stand-alone application.

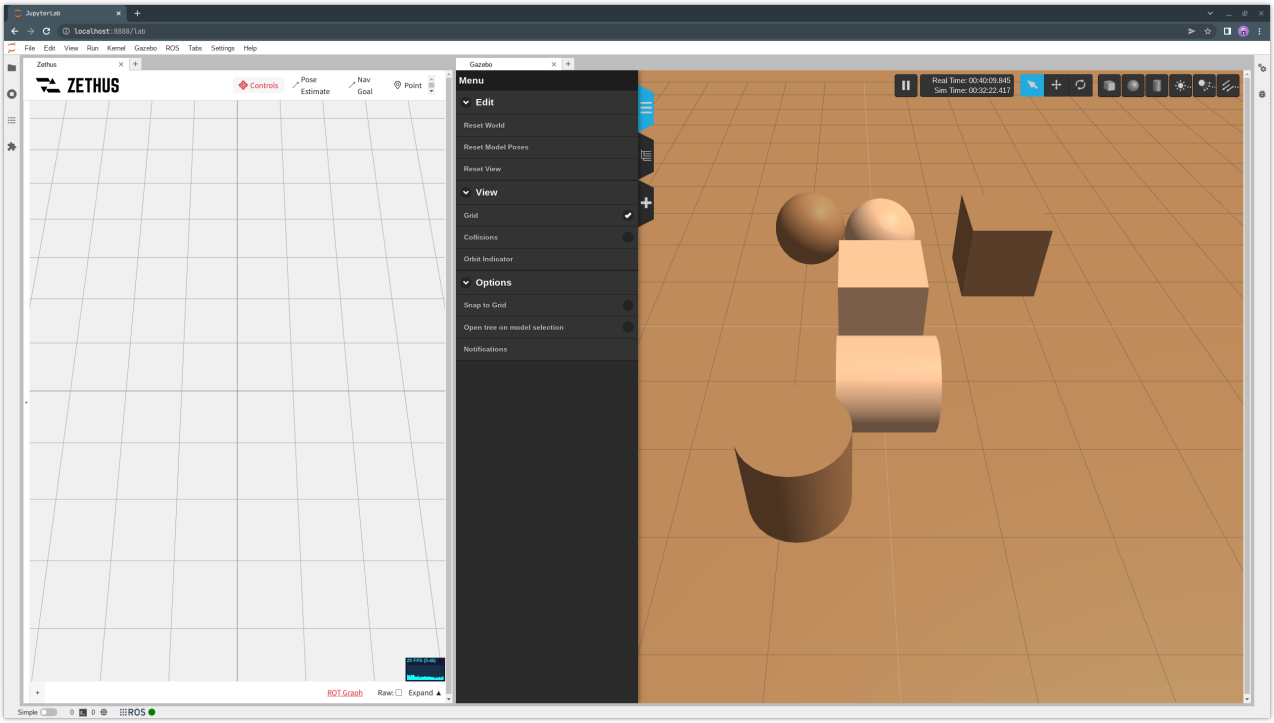


Figure 8.1: The Gazebo extension displaying the same shapes the Gazebo server is publishing

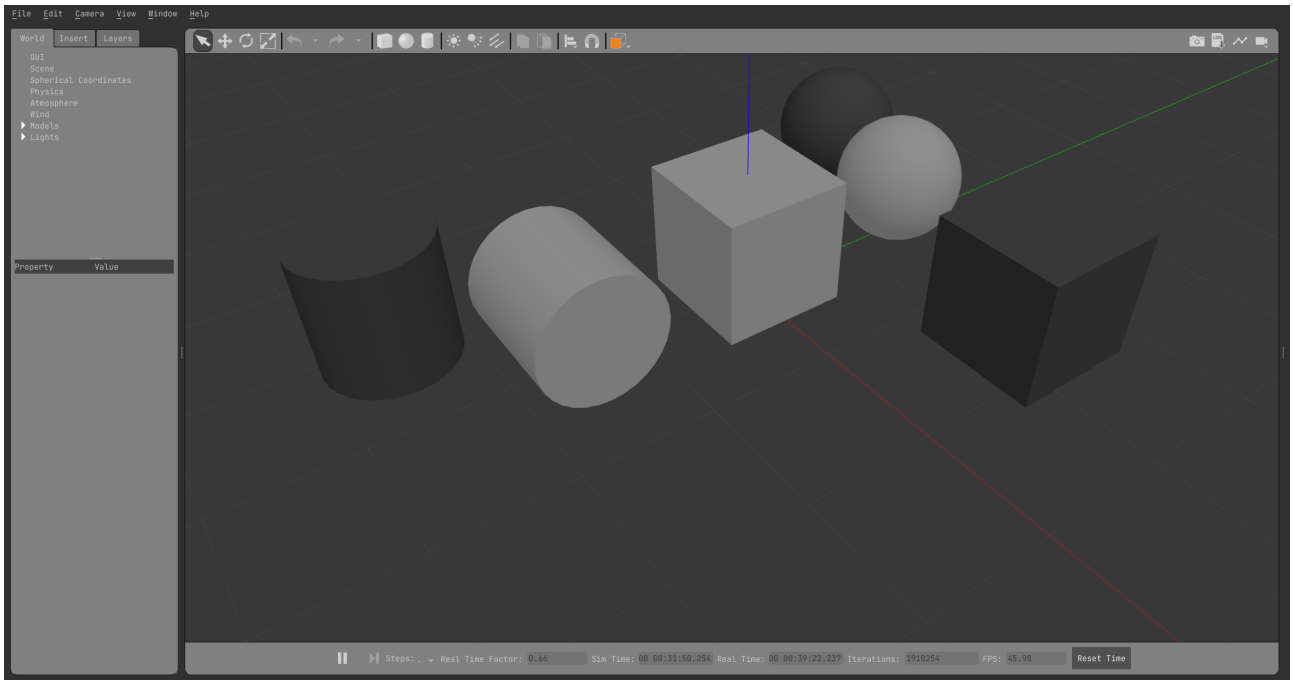


Figure 8.2: The original Gazebo application displaying a variety of shapes

8.3 Gazebo Ignition

The second version of the Gazebo web client was also tested this week. The source code can be found in the Ignition Robotics GitLab page. This version can also be accessed on the gazebosim.org site. In contrast to its predecessor, this application was significantly easier to install since it had been more recently updated. The application can be seen in Figure 8.3. However, the repository is

already archived and it is uncertain if the developers plan on continuing with an additional version.

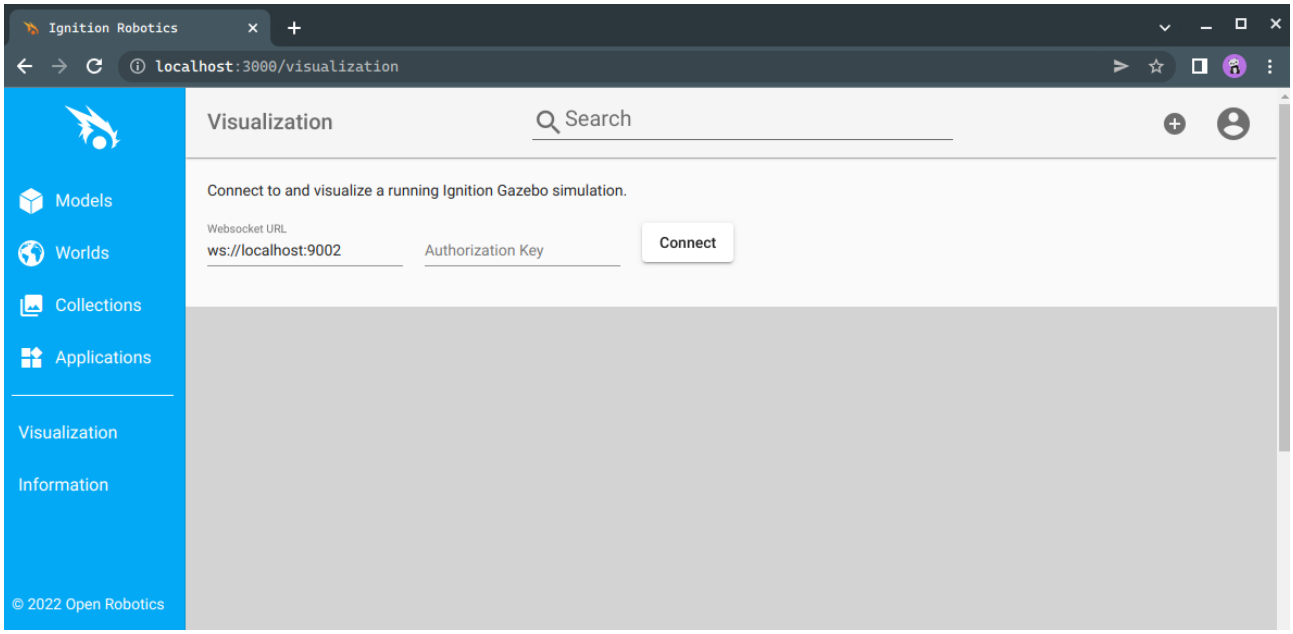


Figure 8.3: `gz3d_v2`, the newest version of the Gazebo web client

8.4 Future Work

The development of the Gazebo extension will continue, there are several aspects still left to be accomplished. The main challenges will include configuring the extension so that the Gazebo client and bridge are built automatically during installation. Additionally, when activating the extension, the Gazebo server and bridge should also be launched. And lastly, a more efficient way of installing model files which will also include the robot descriptions from the ROS packages still needs to be determined.

Week 09

9.1 JROS Profile II

The issues with the JROS profile continued this week, beginning with not being able to run the ROS master. When initialized, JupyterLab displayed the warning below:

```
Master: WARNING

WARNING: cannot load logging configuration file, logging is disabled
Resource not found: roslaunch
The traceback for the exception was written to the log file
```

It was determined that the main cause of this issue was because the `conda` base environment was not being automatically activated when the JupyterHub profile is spawned. Because of this, none of the ROS environment variables required for a smooth operation were set.

9.1.1 Conda Base Environment

The simplest solution was to manually activate the base environment from a terminal with `conda activate`, however, that resulted in

```
CommandNotFoundError: Your shell has not been properly configured to use 'conda activate'.
To initialize your shell, run

    $ conda init <SHELL_NAME>
```

This problem has been documented in conda issue #7980. With the suggested workaround of using `source activate` instead, the base environment can be successfully activated. However, this environment is only activated in the current terminal but the environment variables are required globally. Additionally, ROS master could not be initialized, not even from the terminal. Trying to run `roscore` resulted in the following error:

```
RLException: Unable to contact my own server at [http://jupyter-4295:34937/].
This usually means that the network is not configured properly.
```

Thankfully, the solution for this error was to simply set the ROS IP. Afterwards, the ROS master could be ran from a terminal but not yet from the ROS extension.

```
$ export ROS_IP=127.0.0.1
```

Later, it was discovered that an environment variable could be set from the `Dockerfile` and persist when running the container by using the `ENV` command.

```
ENV ROS_IP=127.0.0.1
```

9.1.2 Local Docker Containers

The next attempt was to manually activate the base environment by including the instructions in the `Dockerfile`. To speed up the testing, the docker images were saved locally. Since the base image

for `rwth-courses` is stored in a private repository, it is not accessible without the proper credentials. As a workaround, Ubuntu 20.04 was used as the base image. The only problem was that the `ubuntu` image does not come with `conda` preinstalled, thus, it had to be manually installed in the `Dockerfile`.

Once all the requirements and extensions for JupyterLab were installed, activating the base environment was a matter of simply sourcing the `conda` setup file. Although all the ROS variables were set during this build process, these same environment variables were not available when running the container.

9.1.3 Jovyan

Furthermore, it was observed that the default user for the JupyterHub profile should be set to `jovyan` and not `root`. Although adding a new user to the `Dockerfile` is simple, `jovyan` has special permissions which would be more involved to replicate manually. This led to using the Jupyter Docker Stacks as base images.

9.1.4 Jupyter Docker Stack

With a Jupyter base image, it was no longer necessary to manually install `conda` nor to add a new user. The local containers used for testing used the `minimal-notebook` as base image.

```
FROM jupyter/minimal-notebook:latest
```

A few more issues arose when attempting to install the required packages with `mamba` because there were conflicts with the pinned version of Python; the packages require Python 3.9 but the `base-notebook` starts off with Python 3.10. The simplest solution was to remove the pinned file in `/opt/conda/conda-meta/`. Nonetheless, the errors with ROS continued even in the newly created containers. Even explicitly setting the ROS environment variable did not solve the problem of running the ROS master from the JupyterLab extension because the configuration was still incorrect.

9.2 ROSCon 2022 Proposal

Since the submission deadline for proposals is quickly approaching, this week we also dedicated some time to writing a proposal for ROSCon 2022 which will be held in Kyoto, Japan. We believe that sharing our ideas about how ROS can be made more accessible to students could be of great interest to any robotics instructors attending the conference.

9.3 Future Work

In order to fix the JROS issues with ROS, there are a few more things to try:

- `Kubespawner` → figure out how the spawning process works and if the spawning script can be modified to activate the environment before launching JupyterLab.
- `nb_conda_kernels` → this is a JupyterLab extension which permits access to kernels located in other `conda` environments.
- `micromamba-docker` → this repository on GitHub uses `ENTRYPOINT` scripts to activate the base environment when running the Docker container.

Moreover, the development of `jupyros` continues and this involves finalizing the JupyterLab extension for Gazebo.

Week 10

10.1 JROS Profile

Another attempt was made at fixing the ROS master issue in the docker container by exploring some of the alternatives mentioned last week. Starting with the easiest approach, the `nb_conda_kernels` extension was added to the `Dockerfile` to be installed in the base environment. This proved to be unfruitful because the issue with the ROS master is not related to any kernels.

The second approach was to follow the example of `micromamba-docker` to activate the base environment by modifying the scripts executed during startup. A script to activate the environment with all possible commands was added to the container. This script gets sourced by `start-notebook.sh` which is specified in the arguments of the `Dockerfile`'s `CMD` instruction.

```
# Dockerfile
ENTRYPOINT ["tini", "-g", "--"]
CMD ["start-notebook.sh"]
```

```
# activate_env.sh
# For robustness, try all possible activate commands.
conda activate "${ENV_NAME}" 2>/dev/null \
    || mamba activate "${ENV_NAME}" 2>/dev/null \
    || micromamba activate "${ENV_NAME}"
```

Nevertheless, the ROS master issue persisted. After further investigation it was determined that the subprocess which runs the ROS master does not maintain the communication as expected, it immediately shuts down. The exact cause is yet to be identified.

```
# master.py
cls.proc = subprocess.Popen(command,
                            stdout=subprocess.PIPE,
                            stderr=subprocess.PIPE,
                            universal_newlines=True)

...

out, err = cls.proc.communicate()    # Failure
```

10.2 Gazebo Extension

The development of the Gazebo extension continued. The majority of this week was spent on deciphering how the Gazebo bridge and the client are built so that this process can be automated in the extension. Furthermore, the extension was modified so that the Gazebo server and bridge can be ran as subprocesses when the extension is activated. However, it first must be ensured that subprocess can function properly inside Docker containers or it will be necessary to find alternatives for initializing them. More testing needs to be conducted to complete this extension.

10.3 URDF Viewer

To add to the robotics tool set, it was determined that adding a URDF editor would be quite practical. Since JupyterLab can already display XML files and highlight the syntax in a text editor, the focus of this new JupyterLab extension would be on the viewer.

10.3.1 URDF Extension Plan

Considering the most common use cases, the goal for this extension is to enable the user to edit a URDF while being able to simultaneously visualize all the changes. As shown on Figure 10.1, the user would be presented with an text editor (left side) and a robot viewer (right side). The robot viewer should have the standard controls for panning, rotating and zooming. This viewer should also have additional options for changing the robot file, showing or hiding specific elements such as links, changing the background color, etc. Additionally, if there are errors in the URDF, the extension should display the error message to alert the user.

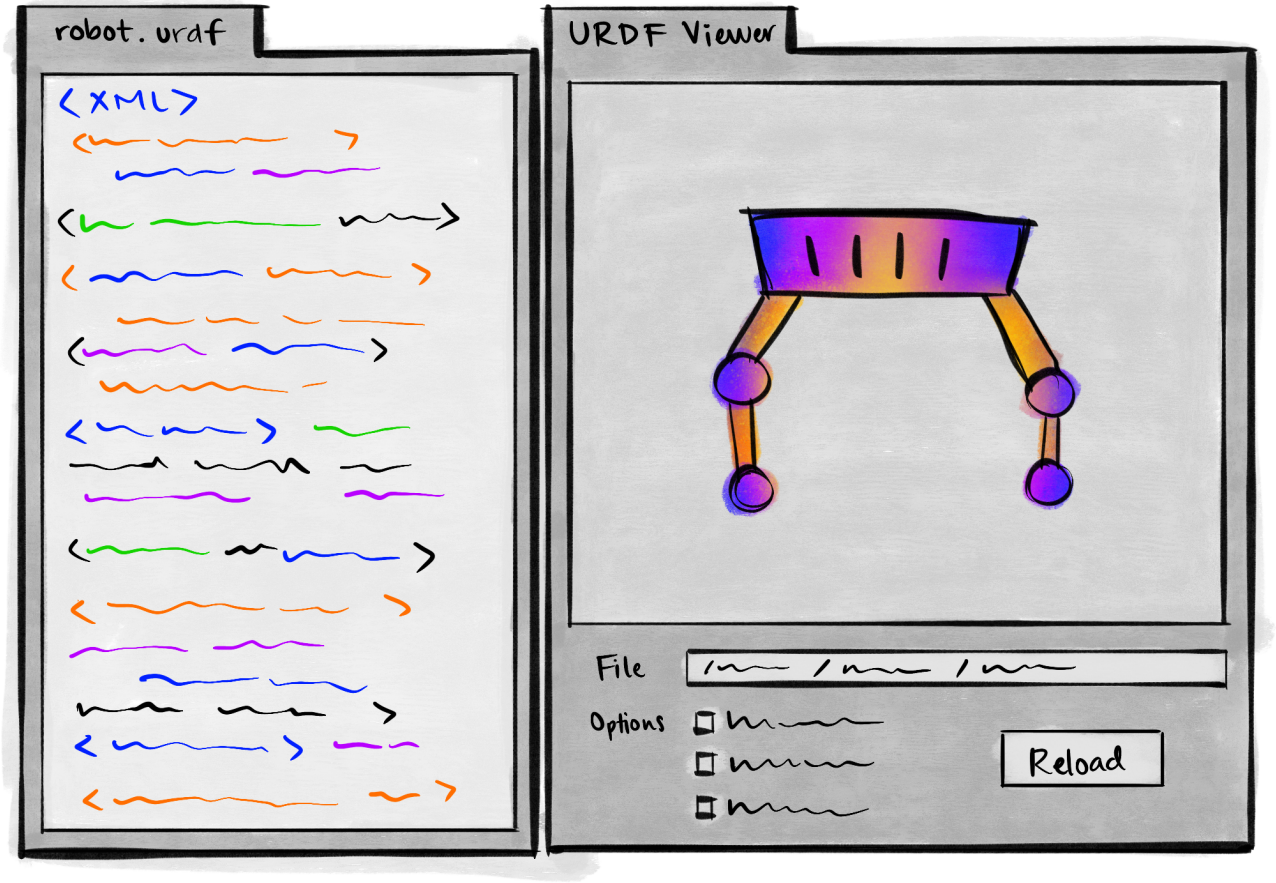


Figure 10.1: Initial concept of URDF extension including a URDF editor and viewer.

10.4 Future Work

One of the main priorities moving forward is to have a functioning JROS profile; this is the major hurdle preventing `jupyros` from being deployed in a JupyterHub environment. Completing the Gazebo and URDF extensions will also be in the scope of the following weeks.

Week 11

11.1 URDF Extension II

The URDF Extension was initialized from the JupyterLab extension *cookiecutter* which greatly simplifies the process. Afterwards, the model and widget and their respective factories were created by following the Documents extension example. In order to make the widget more accessible, a button to open the viewer was added to the main menu and the launcher page as shown in Figure 11.1. The same command was also included in the command palette. The viewer can also be accessed by double clicking on any `.urdf` file.

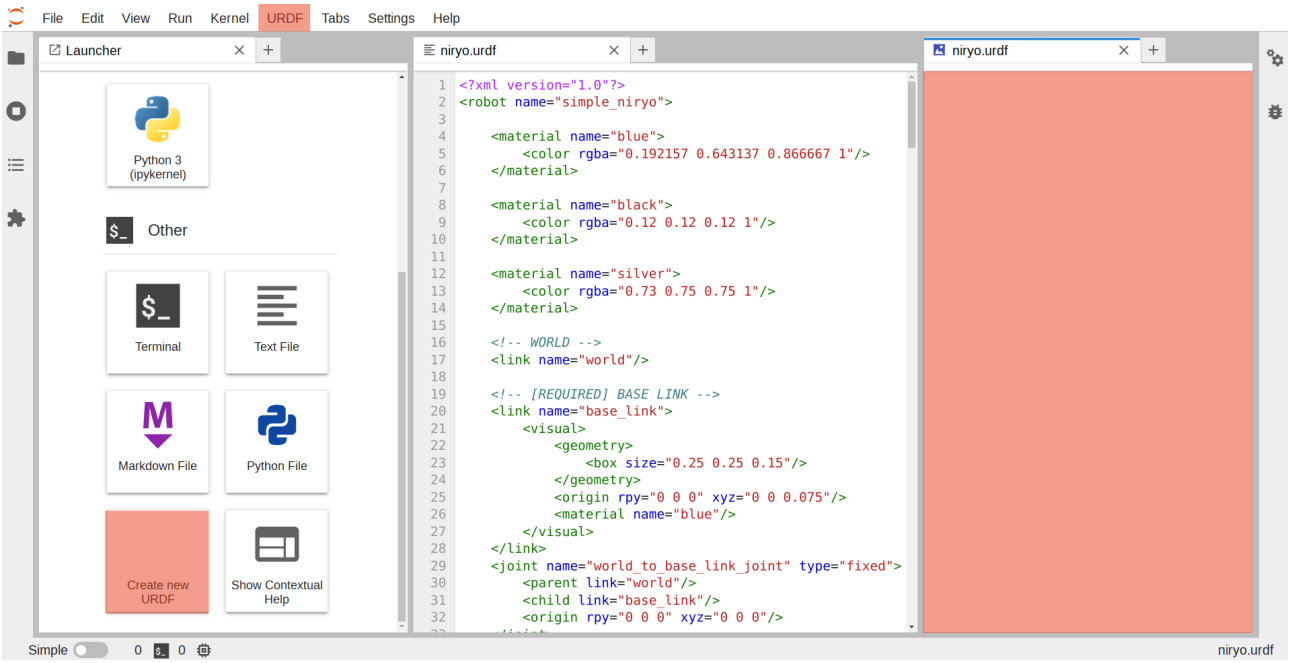


Figure 11.1: The URDF widget (right) can be accessed from the main menu (top) and the launcher (left).

One of the current issues is that the buttons will only open the URDF viewer widget, but in order to read the file, this needs to be separately opened with a text editor. The current opening command needs to be modified, or another command needs to be added, which will open both a text editor for the XML file and the URDF viewer to display the contents of the file.

Given that working with URDF's is very common in the robotics community, other developers have already tackled the problem of visualizing the contents of these files. An example of this is the `urdf_editor` observed in Figure 11.2; this example uses the p5.js library to sketch all the robot components on a browser canvas. Although this `urdf_editor` accomplishes the same goal as what is desired for our extension, the project has mostly been abandoned and has some issues with user interaction.

Another example is the `urdf-loaders` illustrated in Figure 11.3. One of these loaders uses the THREE.js library which is very popular in contemporary visualization applications. And since this loader has been more recently updated, this makes it an ideal choice to integrate into our URDF extension.

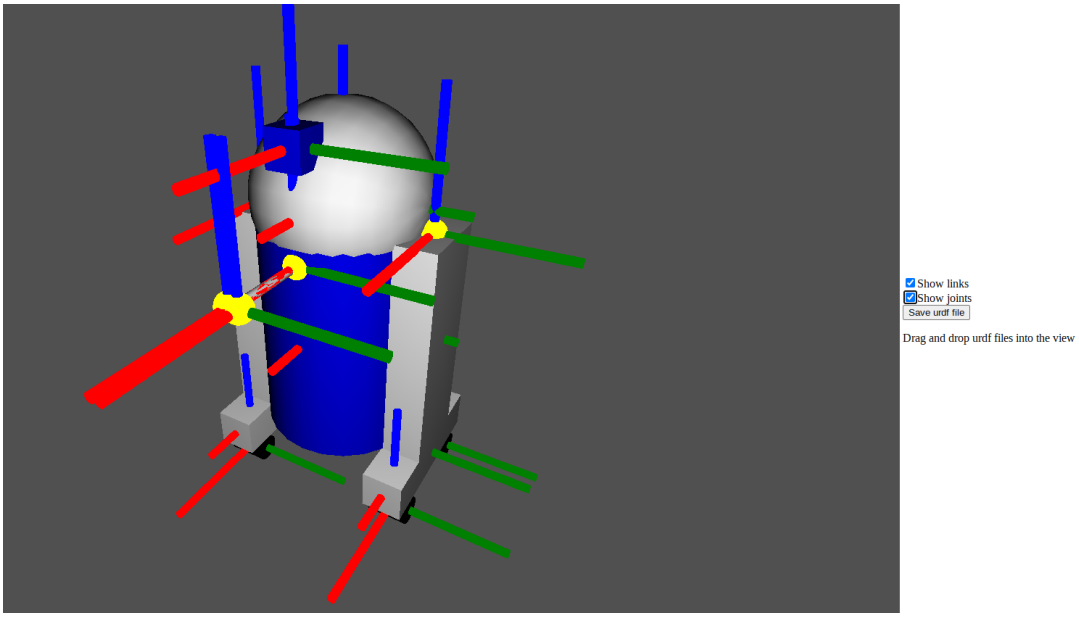


Figure 11.2: URDF editor by Dani Dask

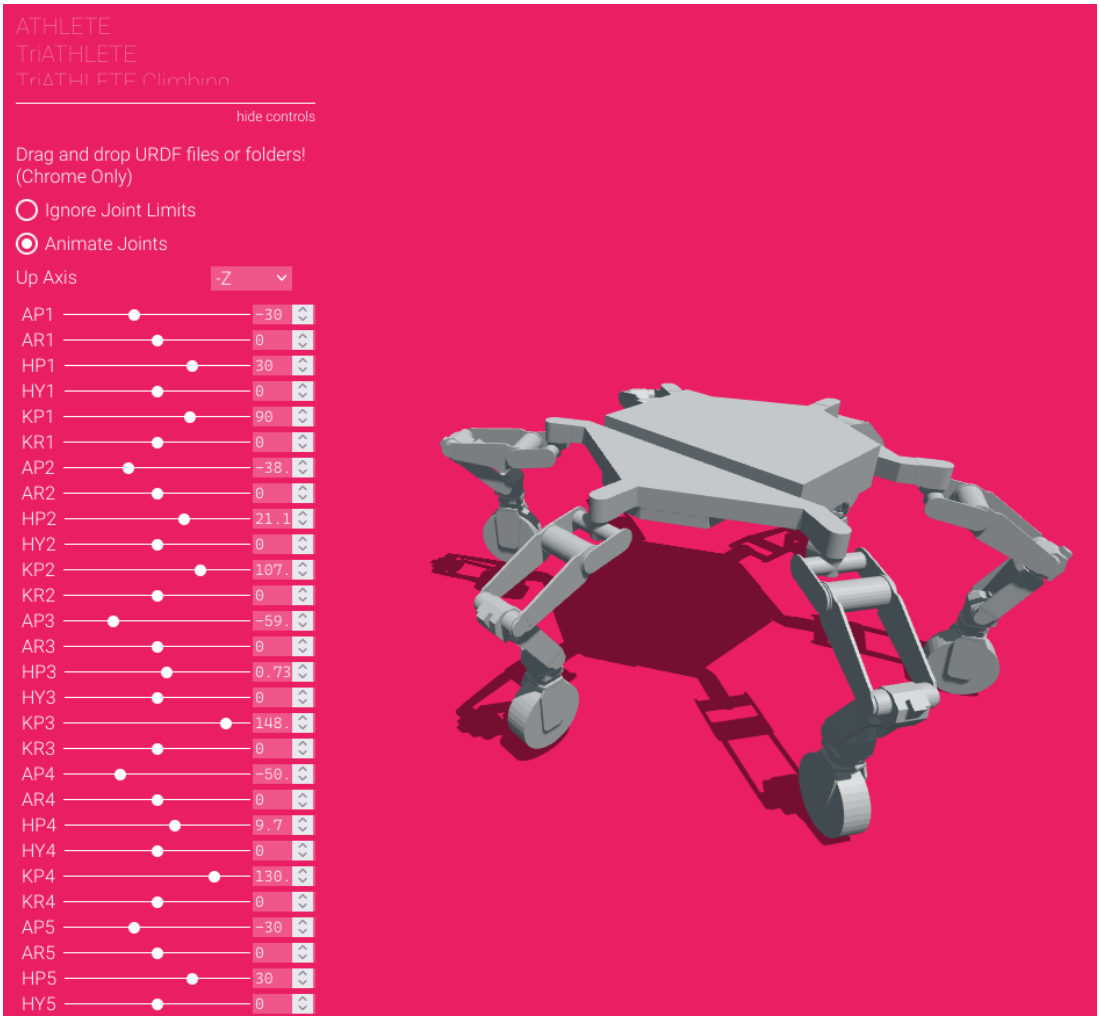


Figure 11.3: URDF loader by Garrett Johnson

11.2 JROS Profile

More time was dedicated this week to figuring out the issue with the ROS master in the Docker container. By modifying the extension's source code inside the container, the root cause of the issue was again identified as the base environment not being activated when JupyterLab is launched. This meant that none of the commands (`conda activate`, `mamba activate`, nor `micromamba activate`) in the `activate_env.sh` script had any effect in activating the environment. As mentioned in Week 09, this problem is related to conda issue #7980. By using the workaround below in the startup scripts, the ROS master was able to successfully run in the Docker container.

```
# activate_env.sh
# Instead of using conda, mamba, or micromamba
source activate base
```

In spite of that, the ROS master can still not run in the JupyterHub environment. This can perhaps be an indicator that some of the configuration of the Docker image is being overwritten during the building process of the JROS profile. Further investigation is required.

11.3 Future Work

The next big step in the project is to display the URDF loader in the widget tab of the extension. Once this is accomplished, it will need to be tested to ensure it dynamically responds to changes in the URDF. Lastly, the visual aspects of the extension such as layout and appearance will need to be assessed.

Week 12

12.1 JupyROS Presentation

This week I prepared a JupyROS demonstration and presented it to the lecturers of the Automated and Connected Driving Challenges (ACDC) at RWTH. The team expressed interest in using JupyROS and incorporating it into their course material.

Having gone through all the functions in the `jupyter-ros` and `jupyterlab-ros` packages, it was noticed that a few small modifications could be made to improve the user experience.

For the JupyterLab extension:

- specify what the *default* ROS path is in the settings,
- have an indication when new paths are saved,
- and make the ROS bag widget compatible with the dark theme.

And for `jupyros`:

- include the `ipycanvas` dependency so that the `turtlesim` widgets work out of the box,
- and make the axis labels for the live-plotting widget configurable or default to *time* on the $x-$axis since this will be the most common use case.

12.2 URDF Extension III

Continuing with the URDF extension, the challenge for this week was to load the contents of a URDF file and be able to add the 3D objects to the scene in the widget. The initial attempt involved following Garrett Johnson's example to create a simple scene and add a six-legged robot to it.

```
scene = new Scene();
scene.background = new Color(0x263238);

renderer = new WebGLRenderer({ antialias: true });
this.node.appendChild(renderer.domElement);

const directionalLight = new DirectionalLight(0xffffff, 1.0);
scene.add(directionalLight);

const ambientLight = new AmbientLight(0xffffff, 0.2);
scene.add(ambientLight);

const ground = new Mesh(new PlaneBufferGeometry(),
                        new ShadowMaterial({ opacity: 0.25 }));
scene.add(ground);
```

The creation of the scene was straight forward, however, the issues began when trying to load the robot's mesh files. The extension on it's own is unable to find local files unless there is a server providing those files.

```
// Load robot
const manager = new LoadingManager();
const loader = new URDFLoader(manager);
loader.load('../../../urdf/T12/urdf/T12_flipped.URDF', result => {
    robot = result;
});
```

The workaround for this was to use Amphion which uses `urdf-js`, a derivative of the `urdf-loaders` package. Amphion works with ROS to find the mesh files indicated in a URDF file. As long as the mesh files are included in a ROS package such as `panda_description`, the extension will be able to locate and load the files by using Amphion.

Several issues were encountered when attempting to use Amphion itself with the extension, but once the developers had corrected the configuration of the extension, robots could finally be visualized in the widget as seen in Figure 12.1. For this example, all the mesh files had to be converted to `.stl` from the original `.obj` extension; and a ROS package called `spot_description` was created to include all the mesh files. Once the environment was sourced properly, the URDF extension was able to locate all the necessary files.

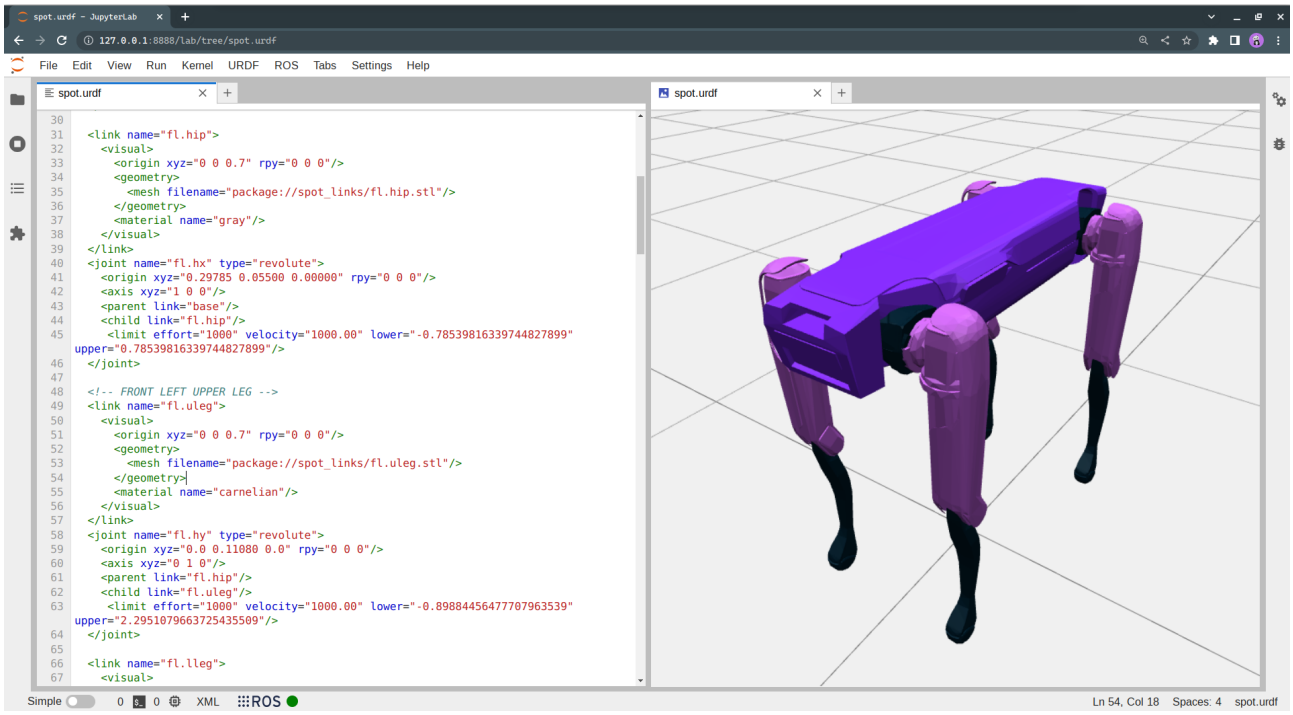


Figure 12.1: The resulting robot display after the incorporation of Amphion for loading ROS package files

12.3 Future Work

The next steps will be to finalize the URDF extension, this will include giving the user more options for configuring the scene such as changing the background and moving the robot joints manually. Additionally, documentation needs to be written for the extension so that users can easily set up and work through any of the given examples.

Conclusion