


ROS2 LAB REPORT DELIVERY TIPS

Abdelbacet Mhamdi

Senior-lecturer, Dept. of EE

Institute of Technological Studies of Bizerte — Tunisia

abdelbacet.mhamdi@bizerte.r-iset.tn

 a-mhamdi

Abstract — We outline a few rules to adhere to in order to properly prepare your labs. The main programming languages you are going to use to implement some control algorithms in ROS2 are: Python, C++ and shell scripting using GNU Bash. It is preferable to write your lab reports in Typst, given the provided files.

Index terms — ROS2, Python, C++, GNU-Bash, Typst

I. PROGRAMMING LANGUAGES

The programming languages we are going to learn through this module is Python of Figure 1, C++ of Figure 2 and shell scripting through GNU Bash of Figure 3. They are chosen for their high-performance computing capabilities, expressive syntax, and extensive ecosystem.

Python Rapid development and high-level logic.

C++ Performance and control over system resources.

GNU Bash Scripting and automation of tasks.



Figure 1: Python logo



Figure 2: C++ logo



Figure 3: GNU Bash logo

Insert your code in each lab report. The detailed explanation of the objects, along the packages you have used is a must. After each code snippet, add a screenshot that showcases your running work when applying the test provided in each exercise.

II. TYPST

Consider using Typst to write your lab reports. The provided templates allow you to focus on the content and seamlessly create a professional-looking report.

Typst supports Markdown syntax, which provides a range of formatting options [1]. Here are some points to help you write your report:

1. Formatting Text:

- Surround a text with single asterisks ‘(*)’ to make it bold
- Use single underscores ‘()’ around your text to emphasize it
- To create headings, use equal signs ‘(=)’ followed by a space at the beginning of a line. The number of ‘(=)’ symbols determines the heading level.

2. Creating lists:

- Unordered list: use a hyphen ‘(-)’ followed by a space for each list item
- Ordered list: use a plus sign ‘(+)’ followed by a space for each list item

3. Code snippets:

- Inline code: enclose the code within backticks ‘()’
- Block of code: use triple backticks followed by the word ‘python’ to enable syntax highlighting

```
```python
import rclpy
from rclpy.node import Node
```
```

4. Inserting Objects:

- Use this syntax if you need to insert an image:

```
#figure(
  image("IMAGE_NAME.EXT", width: 100%),
  caption: [IMAGE_CAPTION],
) <fig:LABEL>
```

@fig:LABEL shows an image.

- Use this syntax if you need to draw a table:

```
#figure(
  table(
    columns: 4,
    [Row 1], [a], [b], [c],
    [Row 2], [1], [2], [3],
  ),
  caption: [Results],
) <tab:LABEL>

@tab:LABEL displays some results.
```

REMINDER

IN EACH DOCUMENT, YOU HAVE TO INSERT WELL ANNOTATED SCREENSHOTS OF YOUR CODE AFTER BEING EXECUTED.

You can leverage those features using the app's intuitive interface and the provided template at the url <https://typst.app/universe/package/ailab-isetbz>, as shown in Figure 4. No installation is required, however, you may need to sign up in order to use the online editor. Keep an eye on your project size. Do not exceed 200MB. A fully fledged documentation on the usage of Typst is available at typst.app/docs.

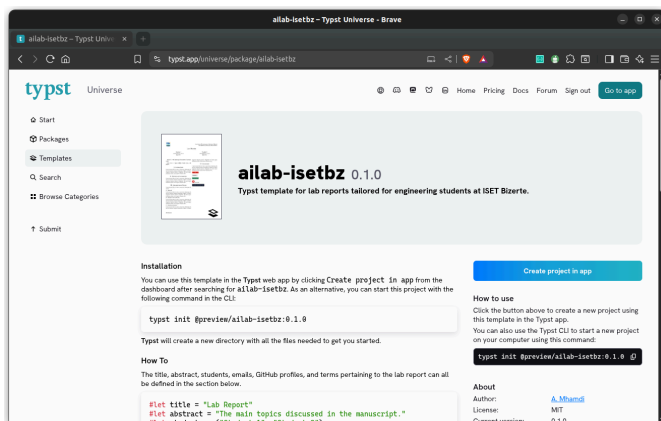


Figure 4: Typst app

III. GitHub

Share your code on GitHub. It's a fantastic way to foster a supportive coding community while gaining exposure to different coding styles and techniques [2], [3], [4].

IV. LINKS BUNDLE

You may find the following links useful:

- GitHub Repository (Figure 5)
github.com/a-mhamdi/ros2

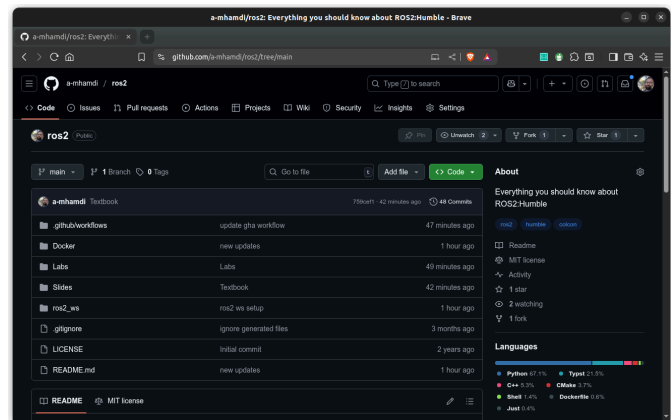


Figure 5: GitHub repository

- Docker Image (Figure 6)
hub.docker.com/repository/docker/abmhamdi/ros2

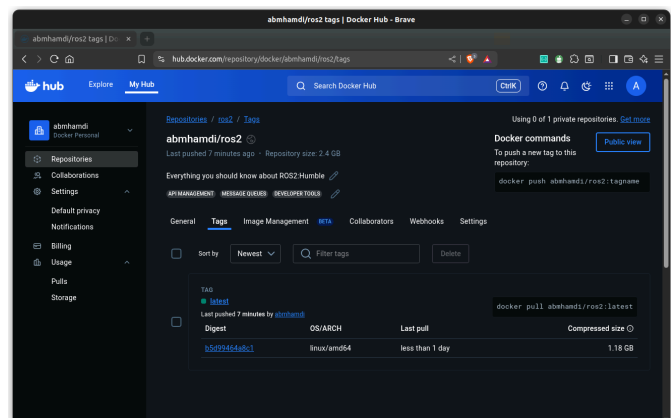


Figure 6: Docker image

V. TIMELINE

The following timeline is proposed to help you organize your work. It is not mandatory to follow it, but it is highly recommended to do so. The labs are designed to be completed in the order they are presented.

| | Sept. | | | | | Oct. | | | | | Nov. | | | | | Dec. | | |
|------------------------------|-------|----|----|----|----|------|----|----|----|----|------|----|----|----|----|------|----|----|
| | W1 | W2 | W3 | W4 | W5 | W1 | W2 | W3 | W4 | W5 | W1 | W2 | W3 | W4 | W5 | W1 | W2 | W3 |
| Lab #1 | | | | | | | | | | | | | | | | | | |
| <i>Develop & Code</i> | | | | | | | | | | | | | | | | | | |
| <i>Review & Update</i> | | | | | | | | | | | | | | | | | | |
| <i>Finalize & Submit</i> | | | | | | | | | | | | | | | | | | |
| Lab #2 | | | | | | | | | | | | | | | | | | |
| <i>Develop & Code</i> | | | | | | | | | | | | | | | | | | |
| <i>Review & Update</i> | | | | | | | | | | | | | | | | | | |
| <i>Finalize & Submit</i> | | | | | | | | | | | | | | | | | | |
| Lab #3 | | | | | | | | | | | | | | | | | | |
| <i>Develop & Code</i> | | | | | | | | | | | | | | | | | | |
| <i>Review & Update</i> | | | | | | | | | | | | | | | | | | |
| <i>Finalize & Submit</i> | | | | | | | | | | | | | | | | | | |
| Lab #4 | | | | | | | | | | | | | | | | | | |
| <i>Develop & Code</i> | | | | | | | | | | | | | | | | | | |
| <i>Review & Update</i> | | | | | | | | | | | | | | | | | | |
| <i>Finalize & Submit</i> | | | | | | | | | | | | | | | | | | |
| Lab #5 | | | | | | | | | | | | | | | | | | |
| <i>Develop & Code</i> | | | | | | | | | | | | | | | | | | |
| <i>Review & Update</i> | | | | | | | | | | | | | | | | | | |
| <i>Finalize & Submit</i> | | | | | | | | | | | | | | | | | | |
| Exam | | | | | | | | | | | | | | | | | | |
| <i>Review Session</i> | | | | | | | | | | | | | | | | | | |
| <i>Evaluation</i> | | | | | | | | | | | | | | | | | | |

REFERENCES

- [1] T. Mailund, *Introducing Markdown and Pandoc*. Berkeley, CA: Apress L. P., 2019.
- [2] S. Guthals, *GitHub*, 2nd edition. in For dummies. Hoboken, NJ: John Wiley & Sons, Inc., 2023.
- [3] P. Bell and B. Beer, *Introducing GitHub*, 1. ed. Beijing: O'Reilly, 2015.
- [4] M. Tsitoara, *Beginning Git and GitHub*. Berkeley, CA: Apress, 2020.