ME462 MECHATRONIC DESIGN

Mini Robot Documentation

Version 1.0

We understand that reading documentation can be challenging, especially when you're eager to dive into the hands-on work. However, taking the time to read the documentation can save you a lot of time in the long run and prevent costly mistakes. That's why we strongly recommend following the documentation carefully..

MINI ROBOT PROJECT
ROS Entegration

Table of Contents

List of Figures

List of Tables

1 Introduction

The ROS part of this documentation is not intended to provide an in-depth tutorial on ROS. Its primary objective is to kickstart your ROS learning journey by guiding you through the basic examples using our mini robot, allowing you to see the ROS structure in action and enabling you to apply what you learn directly with our robot.

In the following sections, you will find the installation steps, an overview of the project structure, basic examples, and a more advanced example. Even if you are already familiar with ROS, I highly recommend reviewing the project structure to understand how it is organized, which will help you add your own improvements effectively.

2 Getting Started

2.1 Installation

To be able to get start the ROS part of project, followings must be installed.

Note

Be sure that you have a compatible version of Ubuntu for ROS2 Humble. If not, it should downloaded to Docker. This document doesn't include Docker installation instructions, but it can be found easily on the internet.

ROS2 Humble

Official installation can be followed: `https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html`

Ad ”opt/ros/humble/setup.bash” to the bashrc to source the enviroment for ROS when the restart, otherwise one must source in every case:

```
sudo nano ~/.bashrc
```

Listing 1: Source setup

Clone the Mini Robot ROS Repository

Clone the repository containing the mini robot's ROS packages:

```
git clone https://github.com/ZNeslisah/MiniROS.git
```

Listing 2: Cloning the Mini Robot Repository

Setup for Workspace

After downloading workspace you need to

```
cd MiniROS/dew_ws
colcon build --symlink-install
source install/setup.bash
```

Listing 3: Setting up Workspace

Note

- `colcon build` is used to build packages in a ROS 2 workspace. It compiles the source code, generates necessary build files, and prepares the packages for use. YOU SHOULD RUN EVERY TIME WHEN YOU MAKE CHANGE (With adding "–symlink-install" no need to do when there is adjustment on pre existing file, run when there is new file.)
- `source` is used to set up the environment for a ROS 2 workspace. Running `source install/setup.bash` configures the shell to recognize the built packages, enabling you to run nodes and use the tools provided by the packages. YOU SHOULD RUN EVERY TIME YOU OPEN A NEW TERMINAL
- If you have other friends who are also using ROS, and connected to same internet. You can choose to add `export ROS_DOMAIN_ID=*number from 1 to 100*` to your the bashrc. This will seperate your communication and prevent you from mistakenly get messages from other peoples.

Additional Setup

Some simulation enviroments and packages must be installed to run the project. I won't directly list them. You can follow the errors and install these when required.

If you have done all the installation until that point, lets run our first launch file:

```
ros2 launch mini_robot letsStart.launch.py
```

Listing 4: Running the Launch File

Recommendation

I highly recommend exploring `tmux` for managing multiple terminal sessions within a single window. Although its keyboard shortcuts may seem challenging at first, mastering them will greatly simplify handling multiple terminals, which is often necessary when working with ROS.

3 Project structure

The project is divided into four main sections, as shown in the figure below:

- **mini_robot**: This section contains configurations and description of the mini robot and the core functionalities.
- **mini_control**: This section includes the control logic for the mini robot. It manages the robot's movements and responses based on sensor inputs and tracking data.

- **aruco_tracker**: This section is responsible for detecting and tracking ArUco markers. It includes nodes and configurations for processing camera input and identifying marker positions.
- **ball_tracker**: This section handles the detection and tracking of a ball. It includes the necessary algorithms and nodes to track the ball's movement using image processing techniques.

The following sections will show the project structure and give small examples. We will start with using `mini_robot` package, but if you are familiar with ROS and want to directly start using it with hardware, you can skip the `mini_control` section.

3.1 mini_robot

3.1.1 Simulation Environments

Before diving into the examples, the following programs should be introduced:

- **Gazebo:** It is a powerful robotics simulator that provides an accurate environment to test robots in both indoor and outdoor settings. It offers physics-based simulation with real-world effects such as gravity, friction, and sensor noise.

  One can simply write "gazebo" to terminal to open a empty gazebo file, but it can be done more with following command

  ```
  ros2 launch mini_robot launch_sim.launch.py
  ```

  Listing 5: Meeting with Gazebo enviroment

- **Rviz:** It is a 3D visualization tool used in ROS for visualizing sensor data, robot models, and other aspects of the robot's environment. It provides an interactive interface to visualize what the robot perceives, such as laser scans, camera feeds, and point clouds, and allows users to monitor the robot's state and debug issues effectively. RViz is primarily used for real-time monitoring and analysis of robot operations.

  Now, open the Rviz and see our robot in there:

  ```
  rviz2 -d src/mini_robot/config/main.rviz
  ```

  Listing 6: Meeting with Rviz enviroment

  Note

  In the above code we have called the rviz2 with pre-build configuration. One can also open rviz2 with only writting **rviz2**, but in that case the robot_model, camera and TF should be added manually with add button in left corner. You should also adjust the fixed frame and the topic names for all added parts.

The mini_robot package includes numerous configuration and description files that define the robot for simulation purposes. While these files won't be explained in this documentation, you have the option to explore their types and structures to gain further insights into how the robot is configured and simulated.

Recommendation

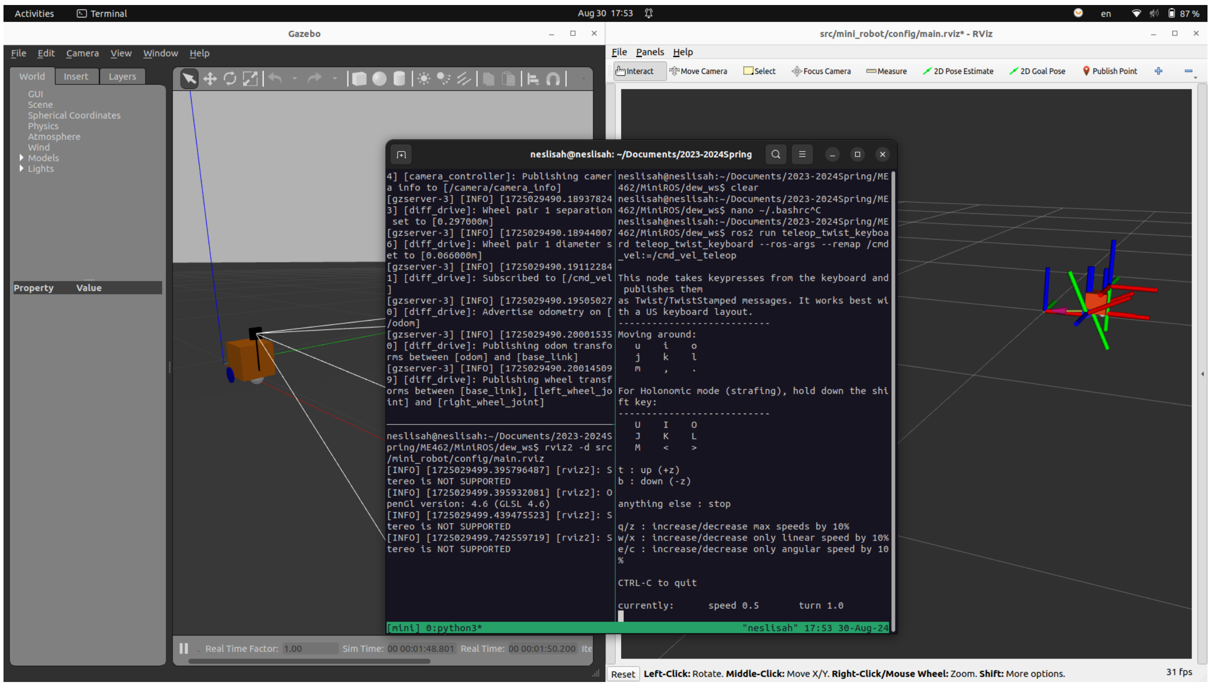
In `mini_robot` package, I have utilized from the tutorial of Articulated Robotics. It was clear and comprehensive tutorial, you can check it if you want. But be careful that this tutorial uses `ros2_control` while we are using custom communication protocol. Checking our protocol is optional and not necessary for understanding ROS implementation or learning, but ensure that the protocol is correctly downloaded onto your Pico device. Check the COMMMMMMM part for that. `https://youtube.com/playlist?list=PLunhqkrRNRhYAffV8JDiFOatQXuU-NnxT&si=yRpMbUpwmb8o5`

3.1.2 teleop_twist_keyboard

Let's try to drive our robot using builtin package of Ros. Run the following command, read the instruction afterwards since the control is a bit weird. If you have closed the gazebo and rviz, you should reopen them for visualization.

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap
    /cmd_vel:=/cmd_vel_teleop
```

Listing 7: Meeting with Rviz enviroment



```
4] [camera_controller]: Publishing camer
a info to [/camera/camera_info]
[gzserver-3] [INFO] [1725029490.18937824
3] [diff_drive]: Wheel pair 1 separation
 set to [0.297000m]
[gzserver-3] [INFO] [1725029490.18944007
6] [diff_drive]: Wheel pair 1 diameter s
et to [0.066000m]
[gzserver-3] [INFO] [1725029490.19112284
1] [diff_drive]: Subscribed to [/cmd_vel
]
[gzserver-3] [INFO] [1725029490.19505027
0] [diff_drive]: Advertise odometry on [
/odom]
[gzserver-3] [INFO] [1725029490.20001535
0] [diff_drive]: Publishing odom transfo
rms between [odom] and [base_link]
[gzserver-3] [INFO] [1725029490.20014509
9] [diff_drive]: Publishing wheel transf
orms between [base_link], [left_wheel_jo
int] and [right_wheel_joint]

neslisah@neslisah:~/Documents/2023-2024S
pring/ME462/MiniROS/dew_ws$ rviz2 -d src
/mini_robot/config/main.rviz
[INFO] [1725029499.395796487] [rviz2]: S
tereo is NOT SUPPORTED
[INFO] [1725029499.395932081] [rviz2]: O
penGl version: 4.6 (GLSL 4.6)
[INFO] [1725029499.439475523] [rviz2]: S
tereo is NOT SUPPORTED
[INFO] [1725029499.742559719] [rviz2]: S
tereo is NOT SUPPORTED
```

```
neslisah@neslisah:~/Documents/2023-2024Spring/ME
462/MiniROS/dew_ws$ clear
neslisah@neslisah:~/Documents/2023-2024Spring/ME
462/MiniROS/dew_ws$ nano ~/.bashrc^C
neslisah@neslisah:~/Documents/2023-2024Spring/ME
462/MiniROS/dew_ws$ ros2 run teleop_twist_keyboa
rd teleop_twist_keyboard --ros-args --remap /cmd
_vel:=/cmd_vel_teleop

This node takes keypresses from the keyboard and
 publishes them
as Twist/TwistStamped messages. It works best wi
th a US keyboard layout.
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shi
ft key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10
%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

```
[mini] 0:python3*                    "neslisah" 17:53 30-Aug-24
```

Activities
Terminal
Aug 30 17:53
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Real Time Factor: 1.00 Sim Time: 00 00:01:48.801 Real Time: 00 00:01:50.200
neslisah@neslisah: ~/Documents/2023-2024Spring
src/mini_robot/config/main.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Goal Pose Publish Point
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps

3.1.3 Launch files

You have done first interraction with a mini robot, before skipping to other package, the launch files should be mentioned:

- **letsStart.launch.py**

  You have already run this launch file, it's only purpose is printing loginfo to the screen.

- **launch_sim.launch.py**

  You have also already run this launch file and observe the robot in similation. It is the launch file which is openning Gazebo enviroment.

- **launch_robot.launch.py**

  We haven't talked about this launch file so far. It's only difference from launch_sim.launch.py is that it is not opening similation enviroment.

- **rsp.launch.py**

  It is the launch file which is openning builtin ROS robot_state_publisher package.

  Note

  The robot_state_publisher is a ROS package that publishes the state of a robot to the rest of the ROS ecosystem. It uses the robot's URDF (Unified Robot Description Format) model to compute and broadcast the positions of all the robot's joints and links in real-time.

- **joystick.launch.py**

  We won't use joystick, but launch setup is provided. If one wants to add that functionality to robot, he or she can use the setup.

- **ball_tracker.launch.py**

  It is the launch file for ball_tracker mode of the robot.

Exercise

3.1 Before going to start with actual robot, can you run the ball tracker node? Don't forget that you should also open simulations. In the repo there is pre built enviroment with balls for gazebo. You should run the gazebo with that configuration. When running ball tracker add "sim_mode:=true" after "ball_tracker.launch.py" keyword.
Answer key is at the end of the documentation

3.2 mini_control

This is the package which we will start to use our mini_robot. Before proceed make sure that you have downloaded our communication package in to your pico wireless.

3.2.1 Nodes

First two nodes are responsible from the communication with pico.

- **pipe_read**

  Messages are get from the pico and converted to Ros topics.

- **pipe_write**

  Ros topics are converted to communication protocol messages and send it to pico.

- **emergency_button**

  Emergency callback, when the button is pushed it stops the car and and activate emergency leds.

- **bumper**

  When mini robot hits its bumper, it stops the car and activate the bumper leds.

- **battery**

  Responsible from showing the battery level with publishing brightness value. In neopixel node brightness value is turned into color between green and red.

- **neopixel**

  Listen all the boolean led messages and send the pixel colors to the pipe_write.

Note

When adding additional nodes to the project, you can utilize the same topics used in pixel_write and pipe_read, ensuring seamless integration without modifying the existing communication flow. However, if modifications are required for pipe_write or pipe_read, it is essential to adjust the communication protocol on the Pico accordingly to maintain compatibility and ensure correct data transmission.
Do not forget to add new created files in to setup.py file.

Exercise

3.2 After reviewing the nodes, you can proceed to run the emergency node. Check to see if it correctly stops when the button is pressed. Run the nodes with **ros2 run package_name node_name**

Have you noticed that even running basic functionalities in ROS requires multiple terminals? This highlights the importance of launch files. Launch files allow us to start multiple nodes simultaneously, streamlining the process and enhancing efficiency.

For example, try to run **ros2 launch mini_control emergency_launch.py** instead of last 4 lines of answer key.

How to use joystick for control?

Install the evtest to check the connection. After running evtest, try to push the joystick buttons, you should see the reactions if your connection is successful.

```
sudo apt install joystick jstest-gtk evtest
evtest
```

Listing 8: Install evtest

Check to joy node to ensure that everything work perfectly. Run the followings in 2 seperate lines. When press the button you should see that topic output is changed.

```
    ros2 run joy joy_enumerate_devices
    ros2 topic echo /joy
```

Listing 9: Check joystick with ROS

Recommendation

If you want to see the which buttons corresponding to which numbers. You can use Josh Newans joy tester node. `$https://github.com/joshnewans/joy_tester.git$`

Exercise

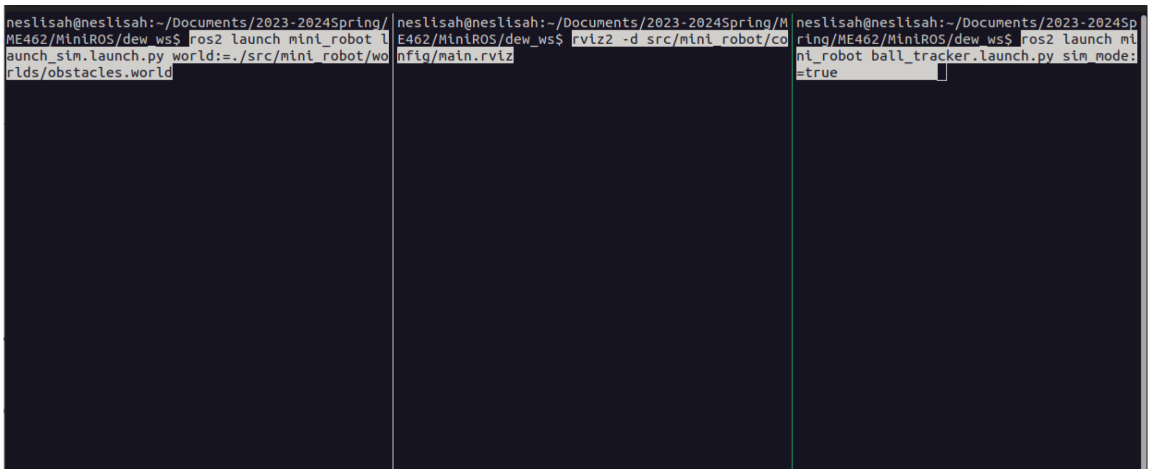
3.3 Can you control the actual robot with joystick now? Search for the needed launch files and nodes, choose proper ones.

4 Answer Key

The lines on answer keys should run on seperate terminals.

3.1

```
ros2 launch mini_robot launch_sim.launch.py world:=./src/mini_robot/
   worlds/obstacles.world
rviz2 -d src/mini_robot/config/main.rviz
ros2 launch mini_robot ball_tracker.launch.py sim_mode:=true
```



```
neslisah@neslisah:~/Documents/2023-2024Spring/
ME462/MiniROS/dew_ws$ ros2 launch mini_robot l
aunch_sim.launch.py world:=./src/mini_robot/wo
rlds/obstacles.world
```

```
neslisah@neslisah:~/Documents/2023-2024Spring/M
E462/MiniROS/dew_ws$ rviz2 -d src/mini_robot/co
nfig/main.rviz
```

```
neslisah@neslisah:~/Documents/2023-2024Sp
ring/ME462/MiniROS/dew_ws$ ros2 launch mi
ni_robot ball_tracker.launch.py sim_mode:
=true
```

3.2

```
ros2 launch mini_robot launch_sim.launch.py world:=./src/mini_robot/
   worlds/obstacles.world
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap
    /cmd_vel:=/cmd_vel_teleop
ros2 run mini_control pipe_write
ros2 run mini_control pipe_read
ros2 run mini_control emergency_button
ros2 run mini_control neopixel
```

3.3

```
ros2 launch mini_robot launch_robot.launch.py
ros2 launch mini_robot joystick.launch.py
ros2 run mini_control pipe_write
```