RS2 Technical Documentation (Draft)

Project Overview

We are collaborating with two other groups to create a robot-guided art gallery experience that includes drink service. Our team's contribution focuses on TurtleBot3 robots that autonomously deliver drinks while navigating the gallery space and avoiding both known and detected obstacles.

User Assumptions: This documentation assumes the user has some familiarity with Ubuntu and ROS 2 basics. However, all key software installation and setup steps (e.g., ROS 2 workspace creation, rosdep usage) are clearly explained in the Software Setup section for users less experienced with the platform.

Key Features / Subsystem List

The system has been modularised into four subsystems:

1. **Path Planning and Object Avoidance:** Calculates the shortest navigable route to target locations using A* (A-star) pathfinding logic and dynamically updates the route when new obstacles are detected.
2. **Movement Logic:** Reduces delay between movements, manages a queue of goals, and ensures indefinite operation of multiple robots.
3. **Simultaneous Operation and Goal Distribution:** Allows both robots to be managed from a single computer, and handles task assignment to optimise robot usage.
4. **Initialisation and Object Detection:** Uses visual (camera) and LiDAR data to localise each TurtleBot within the environment and to detect, classify, and publish both known and unknown obstacles in real-time.

Dependencies

Hardware

- TurtleBot3 Waffle Pi robots (2–5 units, scalable depending on deployment size)
- 360-degree 2D LiDAR sensors (one per TurtleBot, used for SLAM and obstacle detection)
- Logitech USB webcams (mounted on each TurtleBot for object and people detection)
- Central control computer running Ubuntu 22.04 with ROS 2 Humble installed

Software

- **ROS 2 Humble** – the core robotics framework used for real-time communication, navigation, goal management, and multi-robot coordination.
- **RViz 2** – a 3D visualisation tool used during both simulation and real-world deployment to monitor localisation, planned paths, sensor data, and debug markers.
- **OpenCV (C++/Python bindings)** – used for real-time image processing from onboard cameras to detect and classify objects and people.

For Simulation and Development Only:

- **Gazebo Simulator** – a virtual environment used during development to test navigation logic, multi-robot coordination, and object detection before deploying to physical hardware.

Installation

Hardware Setup

In-lab Testing Environment:

- Use foam padding to define gallery walls.
- Represent sculpture by placing $0.2m^3$ square obstacles and tables using cylindrical obstacles with a diameter of 0.3m (within acceptable tolerance).
- To create a real map for use in navigation, a physical map must be built using the above specifications. SLAM will be used to generate a PGM map and corresponding YAML file.
- To set up SLAM and save the map, follow these steps using separate terminals. Before each terminal, be sure to export your domain ID:

  ```
  export ROS_DOMAIN_ID=<your_domain_id>
  ```

1. In a terminal, SSH into the TurtleBot via:

   ```
   ssh ubuntu@192.168.0.2XX
   ```

2. Then bring up the robot using:

   ```
   export ROS_DOMAIN_ID=<your_domain_id>
   ros2 launch turtlebot3_bringup robot.launch.py
   ```

3. On your central computer, run the following in separate terminals:

Terminal 1: Launch world in Gazebo (if simulating)

```
export TURTLEBOT3_MODEL=waffle_pi
ros2 launch turtlebot3_gazebo Gallery_Test2.launch.py
```

Terminal 2: Start Nav2 stack

```
ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True
```

Terminal 3: Start SLAM Toolbox

```
ros2 launch slam_toolbox online_async_launch.py use_sim_time:=True
```

Terminal 4: Start RViz

```
ros2 run rviz2 rviz2 -d /opt/ros/humble/share/nav2_bringup/rviz/nav2_default_view.rviz
```

Terminal 5: Teleoperate TurtleBot

```
export TURTLEBOT3_MODEL=waffle_pi
ros2 run turtlebot3_teleop teleop_keyboard
```

Terminal 6: Save the map

```
ros2 run nav2_map_server map_saver_cli -f my_map
```

- Once the SLAM-generated map is saved (as a .pgm and .yaml pair), it can be used in localisation and navigation tasks.
- Note: The launch files that utilise the map (i.e., for navigation post-integration) will be confirmed and finalised during the integration phase.

Real Gallery Deployment:

- Relies on a pre-provided gallery map (e.g., `GalleryMapHD.yaml`) for localisation and navigation.

Connecting to TurtleBots from Central Computer:

1. Open a terminals for the main computer.
2. Open an additional termal for the TurtleBot.
3. In the terminal for the TurtleBot, SSH into the TurtleBot in their own unique terminal using:

   ```
   ssh ubuntu@<ip-address>
   ```

   The IP address for in-lab testing is 192.168.0.2XX. where XX is representative of the robots ID.

4. Create and set up the workspace on each TurtleBot:

   ```
   mkdir -p ~/rs2_ws/src
   cd ~/rs2_ws/src
   ros2 pkg create multi_robot_bringup --build-type ament_cmake
   cd multi_robot_bringup
   mkdir launch
   ```

5. Replace contents of `CMakeLists.txt` with:

```
echo "cmake_minimum_required(VERSION 3.5)
project(multi_robot_bringup)

find_package(ament_cmake REQUIRED)

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

ament_package()" > CMakeLists.txt
```

6. From the terminal for the central computer, copy the launch file into the launch folder for the TurtleBot:

```
scp launch/namespaced_robot.launch.py \
  ubuntu@<ip-address>:~/rs2_ws/src/multi_robot_bringup/launch/
```

7. Repeat steps 2 to 6 for each additional TurtleBot.

8. On each TurtleBot:

```
cd ~/rs2_ws
colcon build --symlink-install
source install/setup.bash
ros2 launch multi_robot_bringup namespaced_robot.launch.py namespace:=<robot_name
```

   Use namespaces in the format `tb1`, `tb2`, ..., `tb5`.

9. On the central computer, run the multi-robot RViz launch file:

```
ros2 launch turtlebot3_multi_robot multi_robot_nav2_bringup.launch.py
```

10. If deploying more than 2 robots, update `general_settings.yaml` in `params/` to include additional robots in the following format:

```
robots:
  - name: tb1
    x_pose: 0.0
    y_pose: 0.0
    z_pose: 0.01
  - name: tb2
    x_pose: 0.0
```

```
      y_pose: 0.0
      z_pose: 0.01
    - name: tb3
      x_pose: 0.0
      y_pose: 0.0
      z_pose: 0.01
    - name: tb4
      x_pose: 0.0
      y_pose: 0.0
      z_pose: 0.01
    - name: tb5
      x_pose: 0.0
      y_pose: 0.0
      z_pose: 0.01
```

Software Setup

1. Install ROS 2 and Dependencies

```
sudo apt update
sudo apt install ros-humble-desktop
```

2. Set Up Workspace and Install rosdep

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
source /opt/ros/humble/setup.bash
rosdep install --from-paths src -r -y
```

3. Install OpenCV

```
sudo apt install libopencv-dev
```

4. Clone Required Packages

```
cd ~/ros2_ws/src

# Clone TurtleBot3 Simulation repository
git clone -b humble https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git

# Clone TurtleBot3 core packages (includes Gazebo and RViz integration)
git clone -b humble https://github.com/ROBOTIS-GIT/turtlebot3.git

# Clone custom waitforme package
git clone https://github.com/Goodbudy/waitforme.git
```

5. Build and Source the Workspace

```
cd ~/ros2_ws
colcon build --symlink-install
source install/setup.bash
```

Directory Structure

```
ros2_ws/
 src/
    turtlebot3_simulations/
    turtlebot3/
    waitforme/
 install/
 build/
```

Running the System

Integrated System Launch

TERMINAL 1:

```
ros2 launch issy multi_launch.py
```

TERMINAL 2: Add Goals

```
ros2 service call /add_goal issy/srv/AddGoal "{x: 1.0, y: 0.8}"
ros2 service call /add_goal issy/srv/AddGoal "{x: 2.0, y: 2.5}"
```

Execute Goals:

```
ros2 service call /execute_goals issy/srv/ExecuteGoals "{}"
```

Path Planning in RViz

```
ros2 run andy astar_planner
```

```
export TURTLEBOT3_MODEL=waffle_pi
ros2 launch turtlebot3_gazebo Gallery_Test2.launch.py
```

```
ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True \
map:=$HOME/ros2_ws/src/waitforme/GalleryMapHD.yaml
```

```
ros2 topic pub /astar_goal geometry_msgs/msg/PoseStamped "{...}" --once
```

Object Detection & Localisation

Launch detection and localisation nodes in sequence after loading environment and map.

Expected Outcome

- **Object Detection:** Black circles (tables) and red squares (artworks) appear in RViz.
- **Path Planning:** A* path appears once a goal is sent. Additionally, a PNG image is exported to the ROS 2 workspace, visualising the occupancy grid overlaid with the computed A* path.
- **Dual Operation:** Two TurtleBots active in both Gazebo and RViz.

Subsystem Specifics

1. Path Planning and Object Avoidance

Topics: /map, /odom, /astar_goal, /astar_path, /planned_path
Files: YAML, Gazebo simulation files

2. Movement Logic

Topics: /tb1/astar_goal, /tb2/astar_goal, /tb1/cmd_vel, /tb2/cmd_vel
Services: /add_goal, /execute_goal
Files: YAML, Gazebo

3. Simultaneous Operation and Goal Distribution

Files: YAML, Gazebo

4. Initialisation and Object Detection

Topics: /scan, /visualisation_marker, /camera/image_raw, /people_detector/image

Known Issues & Workarounds

- **RViz not showing A* path:** Ensure correct topic is subscribed and frame ID is 'map'.
- **TurtleBot not responding to goals:** Confirm robot is localised with /initialpose.
- **No markers in RViz:** Run detection node and confirm topics are added to RViz sidebar.
- **No tf data for Robot(s):** Internal clock - update clock and use sim time false. namespace - checks nodes and topics and see that the publishers are publishing to the same node the being subscribed too (i.e, /tb1/odom. not /odom or /tb_1/odom)

Team Contributions and Contacts

- Path Planning – Andrew Goode
- Movement Logic – Issy Pitt
- Goal Management – Hallie Robins
- SLAM - Thomas Dodgson

Contact for questions: `hallie.r.robins@student.uts.edu.au`