**Multi-Map-Nav-AR100**

**Introduction:**

The Multi-Map-Nav-AR100 package is for navigating the AR100 robot to separate mapped rooms. Each map is mapped in separate sessions, and saved in the maps directory in the Multi\_Map\_Nav package. Wormhole mechanism allows the robot to switch between maps.

**Key Features:**

* Switching maps based on Wormhole
* SQLite database to store the map connection coordinates.
* Dynamic Map Switching using map\_server
* ROS Action Server for receiving goals
* move\_base Integration for local navigation

**Scenario Description:**

Imagine a robot in am office with multi\_rooms. Each room is mapped separately (e.g., reception = map1, office = map2, meeting room = map3). A user requests the robot to go from reception (map1) to meeting room (map3).

The robot:

* Identifies the wormhole coordinates (map1 → map2 → map3)
* Switch maps as needed
* Continues navigation across maps

**System Architecture:**

The system consists of three main components that work together to enable seamless navigation across multiple maps:

**1. WormholeManager**

* Connects to the SQLite database
* Retrieves wormhole coordinates between maps
* Handles queries to find paths between maps

**2. MapSwitcher**

* Loads map YAML files from the specified directory
* Launches map\_server with the appropriate map
* Manages the transition between different environments

**3. NavigationServer**

* Implements ROS action server for handling navigation goals
* Orchestrates the entire navigation process
* Implements navigation logic for direct and indirect paths
* Utilizes move\_base for actual robot movement

**How It Works:**

A wormhole is a defined position in one map that links to a corresponding position in another map—usually an overlapping area such as a doorway or hallway.

**Wormhole Transition Steps:**

1. Navigate to the wormhole in the current map.

2. Stop the map\_server.

3. Launch map\_server with the target map.

4. Teleport the robot to the corresponding wormhole in the new map.

5. Resume navigation toward the final goal.

**Navigation Process – Step by Step**

**1. Goal Receive**

The system receives a navigation request specifying:

* Target position (x, y)
* Target map (map\_name)

**2. Current Map Evaluation**

* If the target is within the current map, directly forward the goal to move\_base.
* If the target is in another map, initiate multimap path planning.

**3. Path Planning Logic**

a. Direct Path

Check for a direct wormhole from the current map to the target map. If found,

* Navigate to the wormhole position
* Switch to the destination map
* Continue to the target position

b. Indirect Path (via Central Hub)

If no direct path exists:

* Navigate to a wormhole leading to the central map (map2)
* Switch to map2
* Navigate to a wormhole connecting map2 to the target map
* Switch to the target map
* Navigate to the final target position

**4. Move Base Integration**

For each segment:

* Send a goal to move\_base
* Wait for the result (success/failure)
* If successful, proceed to the next stage
* If failed, abort with an error message

**SQL Database Structure:**

wormholes.db file creation steps using command line

*1. sqlite3*

*2. sqlite> .open wormholes.db*

*3. sqlite> CREATE TABLE wormholes\_coord (from\_map TEXT, to\_map TEXT, from\_x REAL, from\_y REAL);*

*4. sqlite> INSERT INTO wormholes\_coord VALUES ('map1', 'map2', 2.7, 4.9);*

*5. sqlite> INSERT INTO wormholes\_coord VALUES ('map2', 'map1', 2.7, 4.9);*

*6. sqlite> INSERT INTO wormholes\_coord VALUES ('map2', 'map3', 12.3, 6.9);*

*7. sqlite> INSERT INTO wormholes\_coord VALUES ('map3', 'map2', 12.3, 6.9);*

*8. sqlite> SELECT \* FROM wormholes\_coord;*

The terminal output looks like this  
*sqlite> SELECT \* FROM wormholes\_coord;*

map1|map2|2.72|4.97

map2|map1|2.72|4.97

map2|map3|12.38|6.9

map3|map2|12.38|6.9

Each row from the database represents a wormhole connection

from\_map | to\_map | from\_x | from\_y

**Action Definition**

NavToGoal.action:

#Request

float64 target\_x

float64 target\_y

string target\_map

---

#Result

bool success

string message

---

#Feedback

string feedback\_msg

**Folder Structure:**

The folder strcture of the Multi\_Map\_Nav package:

multi\_map\_nav

.

├── action

│   └── NavToGoal.action

├── database

│   └── wormholes.db

├── include

│   └── multi\_map\_nav

│   ├── map\_switcher.h

│   ├── navigation\_server.h

│   └── wormhole\_manager.h

├── launch

│   └── multi\_map\_nav.launch

├── maps

│   ├── map1.pgm

│   ├── map1.yaml

│   ├── map2.pgm

│   ├── map2.yaml

│   ├── map3.pgm

│   └── map3.yaml

├── src

│ ├── map\_switcher.cpp

│ ├── navigation\_server.cpp

│ └── wormhole\_manager.cpp

├── CMakeLists.txt

├── package.xml

└── README.md

**Files to launch:**

1. Launch the AR100 robot in the simple\_world

*roslaunch start\_anscer start\_anscer.launch*

2. Launch the localization and navigation server

*roslaunch anscer\_navigation anscer\_navigation.launch*

3. Launch the multi\_map\_nav

*roslaunch multi\_map\_nav multi\_map\_nav.launch*

4. Send a navigation goal:

You can send navigation goals using rostopic:

*rostopic pub /navigate\_to\_goal/goal multi\_map\_nav/NavToGoalActionGoal "goal:*

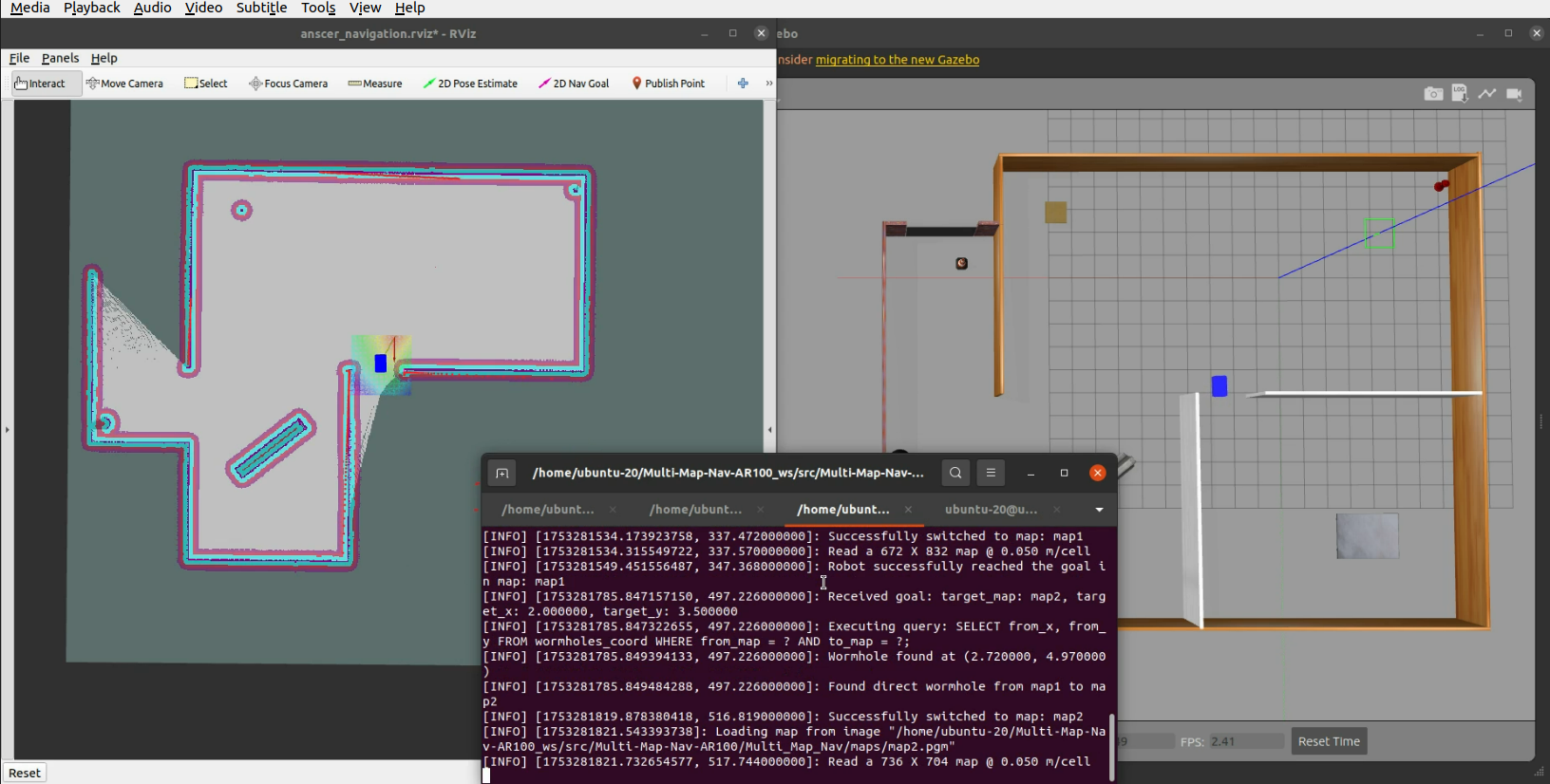
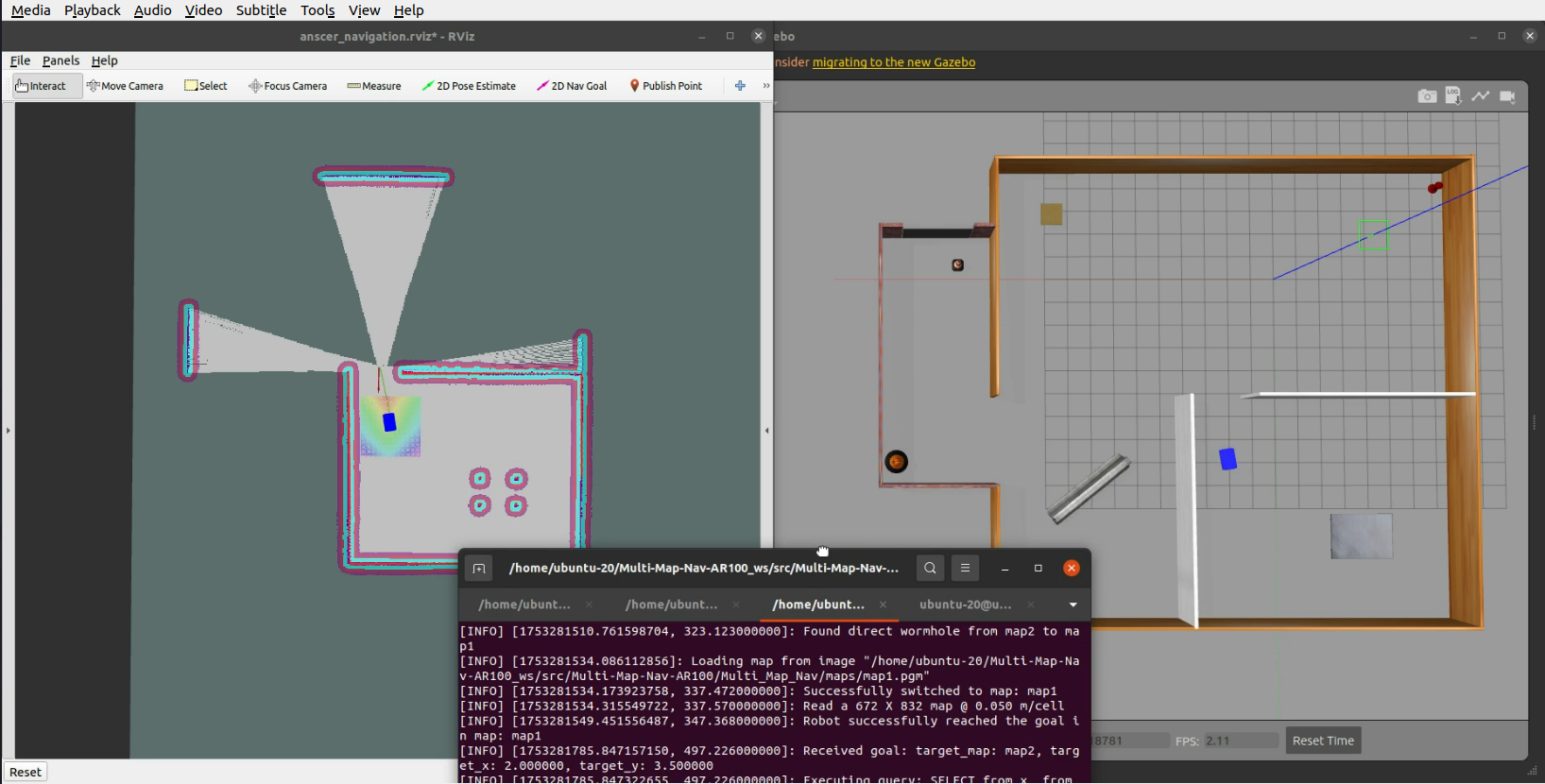
*target\_x: 1.5*

*target\_y: 9.0*

*target\_map: 'map1'"*

Results:

1. Robot is moving from the Reception(current map) to Office(target map)



2. Robot is moving from the Office(map2) to Meeting room(map3)

