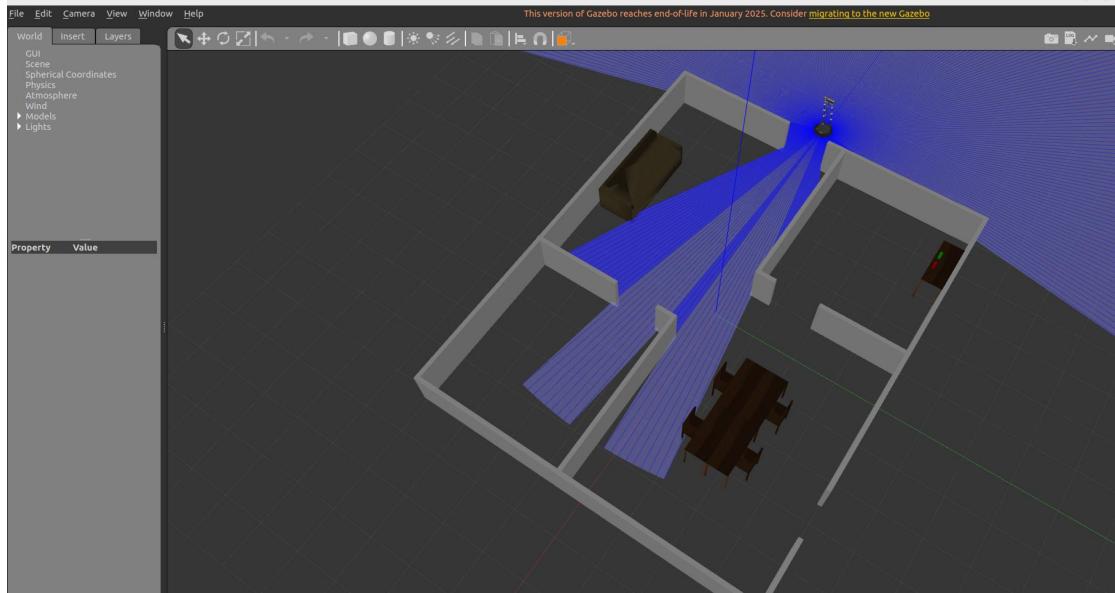


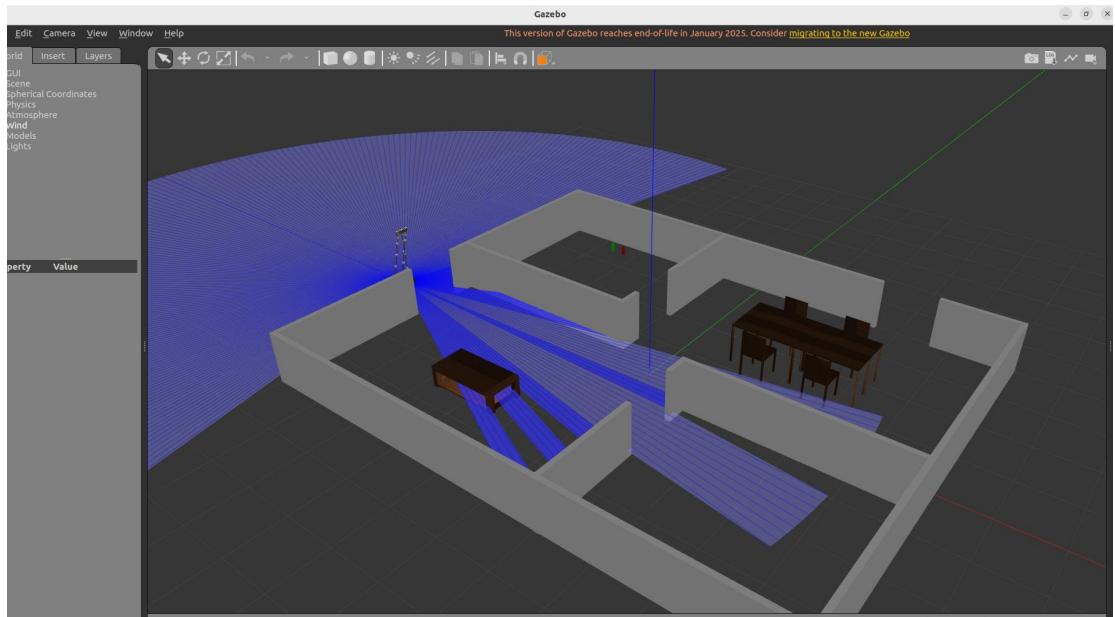
ROS2 Simulation Environment Loading Issue

Problem: When launching the simulation, objects may randomly disappear.

This occurs each time Gazebo is run.



The first launch



The second launch

This is strange... further investigation is required...

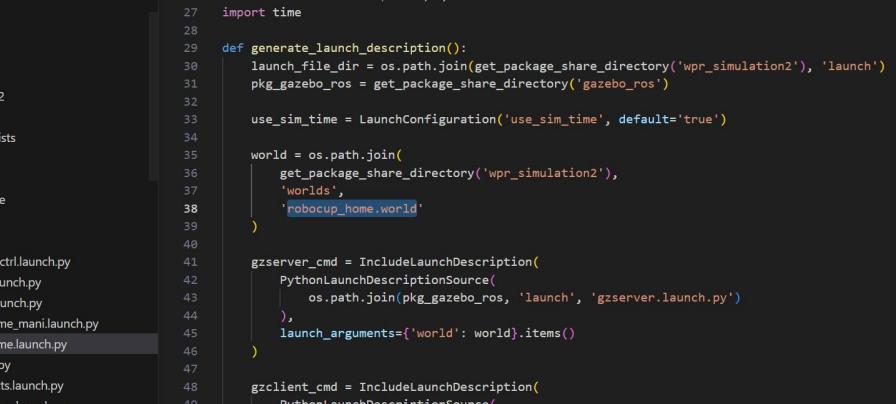
Maptnh@S-H4CK13

Inside `slam.launch.py`, at line 32, this program launches a `robocup_home.launch.py` file... continuing the trace...

```
topic_pkg
vel_pkg
wp_map_tools
wpr_simulation2
> config
> demo_cmakeLists
> demo_spp
> demo_launch
> demo_package
> exercises
> launch
demo_mani_ctrl.launch.py
map_tools.launch.py
navigation.launch.py
robocup_home.mani.launch.py
robocup_home.launch.py
slam.launch.py
spawn_objects.launch.py
spawn_persons.launch.py
spawn_wpb_head_up.launch.py
spawn_wpb_lidar.launch.py
spawn_wpb_mani.launch.py
spawn_wpb.launch.py
world.launch.py
wpb_balls.launch.py
wpb_face.launch.py
wpb_mani.launch.py
wpb_objects.launch.py
wpb_scene_1.launch.py
wpb_simple.launch.py
wpb_table.launch.py
> maps

19 import os
20 from ament_index_python.packages import get_package_share_directory
21 from launch import LaunchDescription
22 from launch.actions import IncludeLaunchDescription
23 from launch.launch_description_sources import PythonLaunchDescriptionSource
24 from launch_ros.actions import Node
25
26
27 def generate_launch_description():
28     launch_file_dir = os.path.join(get_package_share_directory('wpr_simulation2'), 'launch')
29
30     gazebo_cmd = IncludeLaunchDescription(
31         PythonLaunchDescriptionSource(
32             [os.path.join(launch_file_dir, 'robocup_home.launch.py')]
33         )
34     )
35
36     slam_cmd = Node(
37         package="slam_toolbox",
38         executable="sync_slam_toolbox_node",
39         parameters=[{
40             "use_sim_time": True,
41             "base_frame": "base_footprint",
42             "odom frame": "odom",
43             "map frame": "map"
44         }
45     )
46
47     return LaunchDescription([gazebo_cmd, slam_cmd])
48
49
50 if __name__ == '__main__':
51     generate_launch_description()
52
53 r than all the data in the transform cache'
[rviz2:6] [INFO] [1763977432.645782669]: [rviz2]: Message Filter dropping message: frame 'laser_link' at time 267.027 for reason 'the timestamp on the
r than all the data in the transform cache'
[rviz2:6] [INFO] [1763977432.741882179]: [rviz2]: Message Filter dropping message: frame 'laser link' at time 267.133 for reason 'the timestamp on the
```

It is necessary to verify whether the issue is related to the map...



The screenshot shows a ROS2 workspace interface. On the left, there's a tree view of the workspace structure under 'OPEN EDITORS'. The file 'robocup_home.launch.py' is highlighted in blue. The main area displays the content of this file:

```
src > wpr_simulation2 > launch > robocup_home.launch.py
23     from launch.launch_description_sources import PythonLaunchDescriptionSource
24     from launch.substitutions import LaunchConfiguration
25     from launch_ros.actions import Node
26     from launch.actions import OpaqueFunction
27     import time
28
29 def generate_launch_description():
30     launch_file_dir = os.path.join(get_package_share_directory('wpr_simulation2'), 'launch')
31     pkg_gazebo_ros = get_package_share_directory('gazebo_ros')
32
33     use_sim_time = LaunchConfiguration('use_sim_time', default='true')
34
35     world = os.path.join(
36         get_package_share_directory('wpr_simulation2'),
37         'worlds',
38         'robocup_home.world'
39     )
40
41     gzserver_cmd = IncludeLaunchDescription(
42         PythonLaunchDescriptionSource(
43             os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')
44         ),
45         launch_arguments={'world': world}.items()
46     )
47
48     gzclient_cmd = IncludeLaunchDescription(
49         PythonLaunchDescriptionSource(
50             os.path.join(pkg_gazebo_ros, 'launch', 'gzclient.launch.py')
51         )
52     )
```

Use the find command to locate the robocup_home.world map...

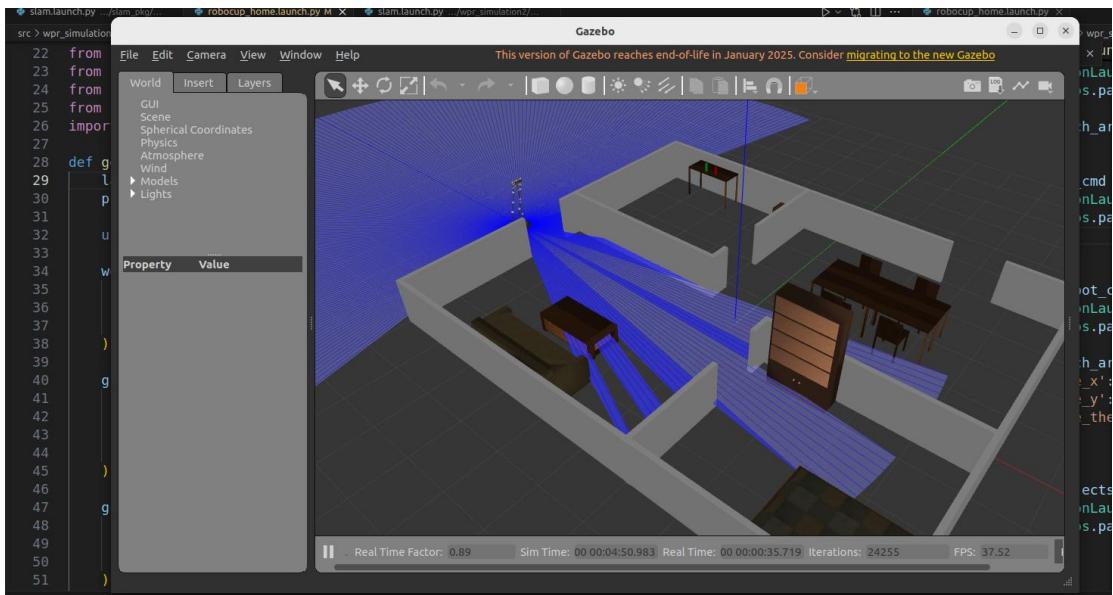
```
$ find ./ -name robocup_home.world
```

Check the two MD5 values — it's clear that they are unrelated to the map loading error...

```
gaz [ ] two@two-virtual-machine: ~/ros2_ws
sim two@two-virtual-machine:~/ros2_ws$ find . -name robocup_home.world
./src/wpr_simulation2/worlds/robocup_home.world
ld = ./install/wpr_simulation2/share/wpr_simulation2/worlds/robocup_home.world
get two@two-virtual-machine:~/ros2_ws$ md5sum ./src/wpr_simulation2/worlds/robocup_home.world ./install/wpr_sim
'WO lation/share/wpr_simulation2/worlds/robocup_home.world
[RO 35a94f0b7ca975482e774287a46763ce ./src/wpr_simulation2/worlds/robocup_home.world
 35a94f0b7ca975482e774287a46763ce ./install/wpr_simulation2/share/wpr_simulation2/worlds/robocup_home.world
two@two-virtual-machine:~/ros2_ws$ server_cmd = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
    os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py'))
```

Then, directly run `$ ros2 launch wpr_simulation2 robocup_home.launch.py`.

Maptnh@S-H4CK13



This time there was no response... I suspect it might be related to memory loading, so I plan to first launch the Gazebo simulation, then delay 20 seconds before loading the components, giving the system enough time to reclaim memory.

Refer to the official documentation:

https://docs.ros.org/en/ros2_packages/humble/api/launch/doc/source/architecture.html

A screenshot of a web browser displaying the ROS2 launch documentation. The URL is https://docs.ros.org/en/ros2_packages/humble/api/launch/doc/source/architecture.html. The page has a sidebar with links to "Welcome to launch's documentation!", "Architecture of launch", and "Launch Entities and Launch Descriptions". The main content area is titled "Architecture of *launch*". It explains that the *launch* package provides core features like describing actions, generating events, introspecting launch descriptions, and executing launch descriptions. It also mentions extension points and the ability to integrate with additional packages. Below this, a section titled "Launch Entities and Launch Descriptions" discusses the *LaunchDescriptionEntity* class, which is the main object in *launch*. It describes how entities inherit from this class and how they can be visited during launching. It also mentions the use of *asyncio.Future* and the *get_asyncio_future()* method for handling asynchronous activity.

The official ROS2 documentation clearly states that launch files are asynchronous and event-driven. This means that using a standard time delay will not work.

Refer to the TimerAction tutorial: <https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Using-Event-Handlers.html>

Maptnh@S-H4CK13

The screenshot shows a web browser displaying the ROS documentation at <https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Using-Event-Handlers.html>. The page content is as follows:

Writing an action server and client (C++)
Writing an action server and client (Python)
Composing multiple nodes in a single process

Launch

- Creating a launch file
- Integrating launch files into ROS 2 packages
- Using substitutions
- Using event handlers
- Managing large projects

t2r
Testing
URDF
Advanced
Demos
Miscellaneous

How-to Guides

- Concepts
- Contact
- The ROS 2 Project
- Related Projects
- Glossary
- Citations
- Other Versions

The `OnExecutionComplete` event handler is used to register a callback function that is executed when the `spawn_turtle` action completes. It logs a message to the console and executes the `change_background_r` and `change_background_r_conditioned` actions when the spawn action completes.

```
RegisterEventHandler(  
    OnExecutionComplete(  
        target_action=spawn_turtle,  
        on_stdout=lambda event: LogInfo(  
            msg='Spawn request says "{}".format(  
                event.text.decode().strip()  
            )  
        ),  
        ),  
)
```

The `OnProcessExit` event handler is used to register a callback function that is executed when the turtlesim node exits. It logs a message to the console and executes the `EmitEvent` action to emit a `Shutdown` event when the turtlesim node exits. It means that the launch process will shutdown when the turtlesim window is closed.

```
RegisterEventHandler(  
    OnProcessExit(  
        target_action=turtlesim_node,  
        on_exit=[  
            LogInfo(msg=(EnvironmentVariable(name='USER')),  
                action=Shutdown(),  
                condition=EnvironmentVariable(name='USER').value != 'root')  
        ],  
    ),  
)
```

Try modifying `robocup_home.launch.py` by adding from `launch.actions import`

`IncludeLaunchDescription, TimerAction` at line 26.。

The screenshot shows a terminal window with the file `robocup_home.launch.py` open. The code is as follows:

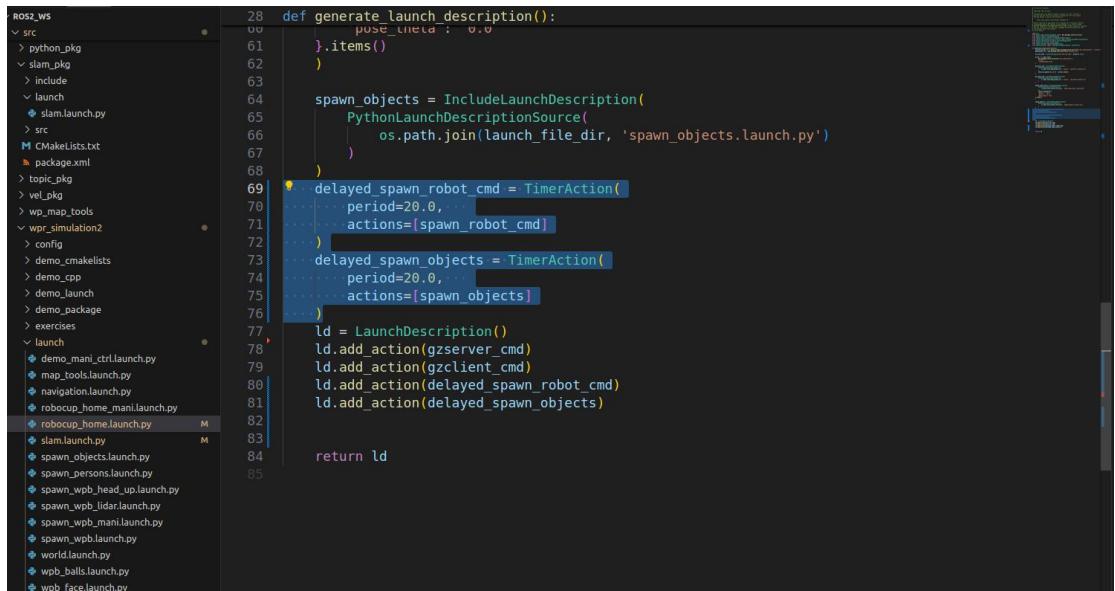
```
# Fix By Maptnh  
import os  
from ament_index_python.packages import get_package_share_directory  
from launch import LaunchDescription  
from launch.actions import IncludeLaunchDescription  
from launch.launch_description_sources import PythonLaunchDescriptionSource  
from launch.substitutions import LaunchConfiguration  
from launch_ros.actions import Node  
from launch.actions import OpaqueFunction  
from launch.actions import IncludeLaunchDescription, TimerAction  
  
def generate_launch_description():  
    launch_file_dir = os.path.join(get_package_share_directory('wpr_simulation2'), 'launch')  
    pkg_gazebo_ros = get_package_share_directory('gazebo_ros')  
  
    use_sim_time = LaunchConfiguration('use_sim_time', default='true')  
  
    world = os.path.join(  
        get_package_share_directory('wpr_simulation2'),  
        'worlds',  
        'robocup_home.world')
```

Add the following code...

```
delayed_spawn_robot_cmd = TimerAction(  
    period=20.0,  
    actions=[spawn_robot_cmd]  
)  
delayed_spawn_objects = TimerAction(  
    period=20.0,  
    actions=[spawn_objects]  
)  
<SNIP>  
ld.add_action(delayed_spawn_robot_cmd)  
ld.add_action(delayed_spawn_objects)
```

Delay the component startup...

Maptnh@S-H4CK13

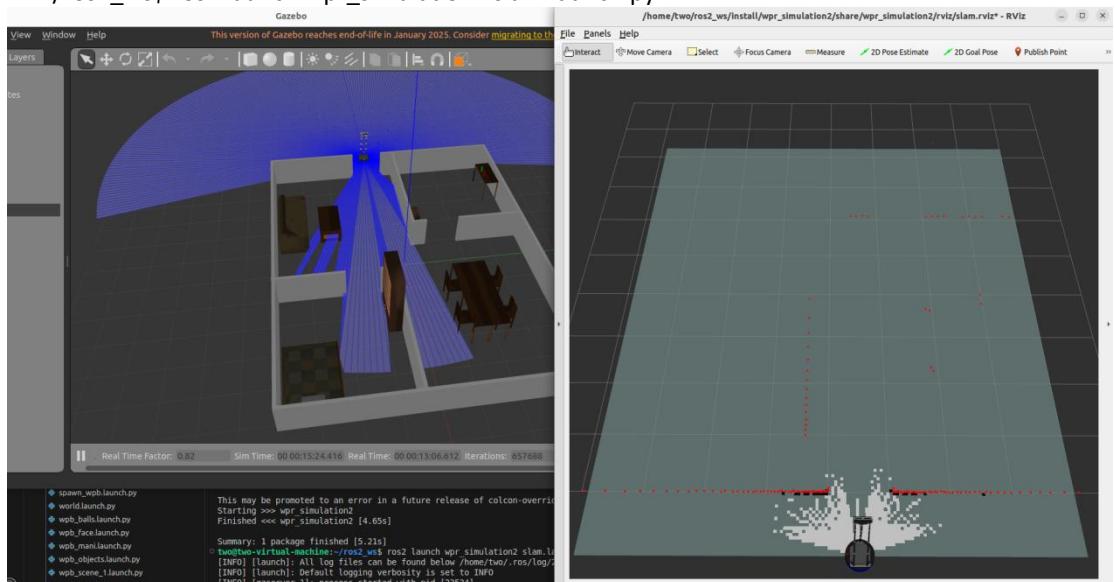


```
28 def generate_launch_description():
  29     pose_theta : 0.0
  30     }
  31     )
  32
  33     spawn_objects = IncludeLaunchDescription(
  34         PythonLaunchDescriptionSource(
  35             os.path.join(launch_file_dir, 'spawn_objects.launch.py')
  36         )
  37     )
  38
  39     delayed_spawn_robot_cmd = TimerAction(
  40         period=20.0,
  41         actions=[spawn_robot_cmd]
  42     )
  43
  44     delayed_spawn_objects = TimerAction(
  45         period=20.0,
  46         actions=[spawn_objects]
  47     )
  48
  49     ld = LaunchDescription()
  50     ld.add_action(gzserver_cmd)
  51     ld.add_action(gzclient_cmd)
  52     ld.add_action(delayed_spawn_robot_cmd)
  53     ld.add_action(delayed_spawn_objects)
  54
  55
  56     return ld
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
```

Compiling the code...

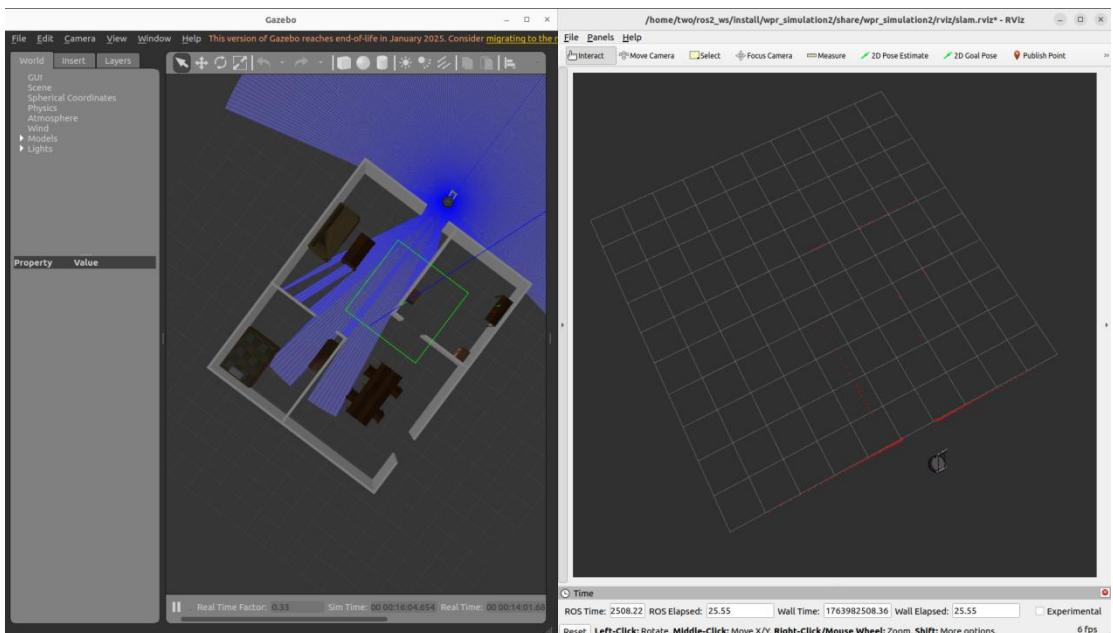
Xx:~/ros2_ws\$ colcon build --packages-select wpr_simulation2

Xx:~/ros2_ws\$ ros2 launch wpr_simulation2 slam.launch.py

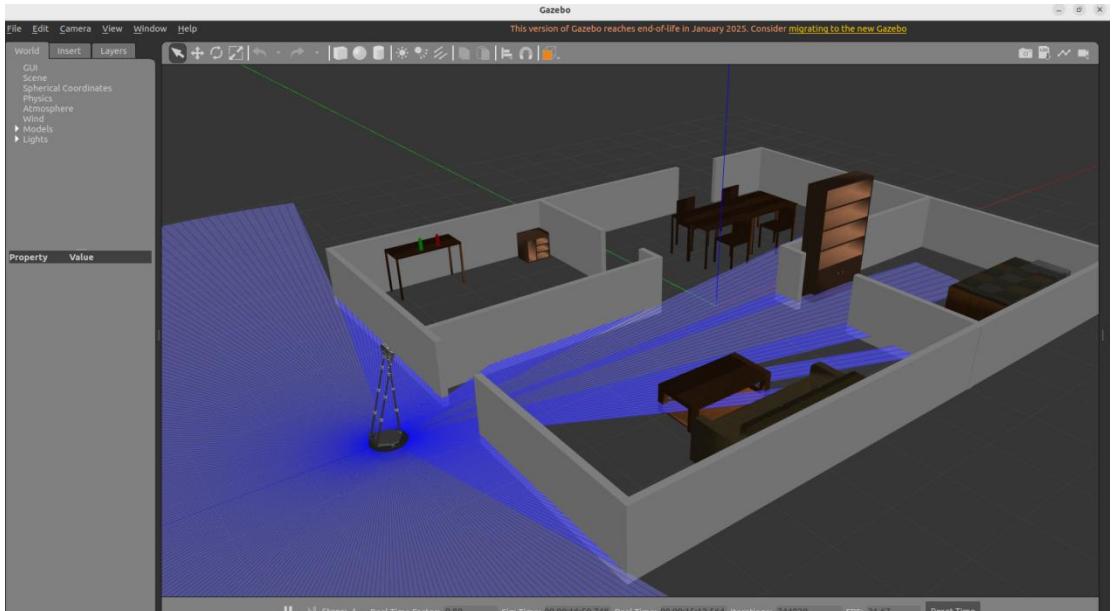


The first launch

Maptnh@S-H4CK13



The second launch



The third launch

```
#!/usr/bin/env python3
# wpr_simulation2/launch/robocup_home.launch.py
# Copyright 2023 6-robot.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Maptnh@S-H4CK13

```
# Fix By Maptnh

import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_descriptions import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch.actions import OpaqueFunction
from launch.actions import IncludeLaunchDescription, TimerAction

def generate_launch_description():
    launch_file_dir = os.path.join(get_package_share_directory('wpr_simulation2'), 'launch')
    pkg_gazebo_ros = get_package_share_directory('gazebo_ros')

    use_sim_time = LaunchConfiguration('use_sim_time', default='true')

    world = os.path.join(
        get_package_share_directory('wpr_simulation2'),
        'worlds',
        'robocup_home.world'
    )

    gzserver_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')
        ),
        launch_arguments={'world': world}.items()
    )

    gzclient_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(pkg_gazebo_ros, 'launch', 'gzclient.launch.py')
        )
    )

    spawn_robot_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(launch_file_dir, 'spawn_wpb_lidar.launch.py')
        ),
        launch_arguments={
            'pose_x': '-6.0',
            'pose_y': '-0.5',
            'pose_theta': '0.0'
        }.items()
    )

    spawn_objects = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(launch_file_dir, 'spawn_objects.launch.py')
        )
    )
    delayed_spawn_robot_cmd = TimerAction(
        period=40.0,
        actions=[spawn_robot_cmd]
    )
```

Maptnh@S-H4CK13

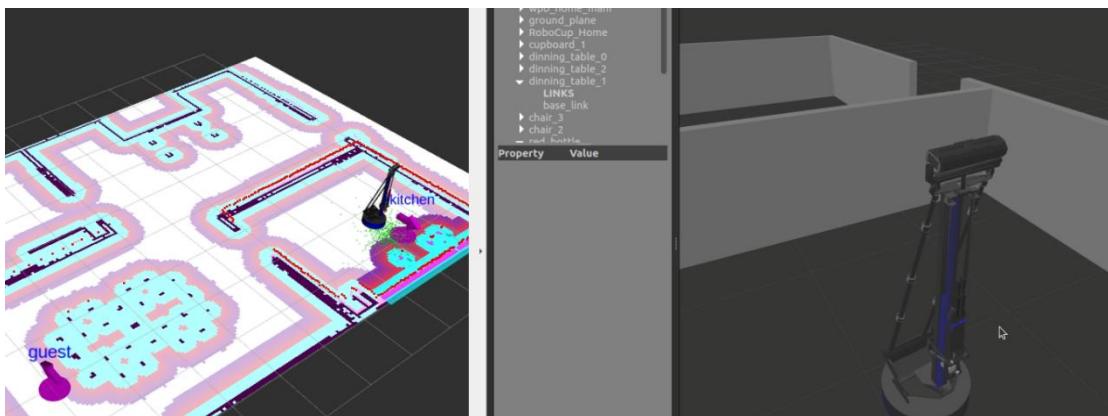
```
delayed_spawn_objects = TimerAction(  
    period=40.0,  
    actions=[spawn_objects]  
)  
ld = LaunchDescription()  
ld.add_action(gzserver_cmd)  
ld.add_action(gzclient_cmd)  
ld.add_action(delayed_spawn_robot_cmd)  
ld.add_action(delayed_spawn_objects)  
  
return ld
```

The original implementation overlooked that starting numerous programs at once could immediately trigger memory overflow, resulting in system failure.

ROS2 Robot Handling Program: Reverse Analysis

and Control Hijack Bypass

There is no denying that this part is quite complex; I also spent a long time analyzing how it works... The first time I used the official command `$ ros2 run home_pkg fetch`, the robot seemed unable to perform the grasp, and inexplicably moved backward...



Open fetch.cpp

Maptnh@S-H4CK13

The screenshot shows the VS Code interface with two tabs open: 'robocup_home.launch.py' and 'fetch.cpp'. The 'fetch.cpp' tab is active, displaying C++ code. Line 69 is highlighted in blue, indicating the current line of interest.

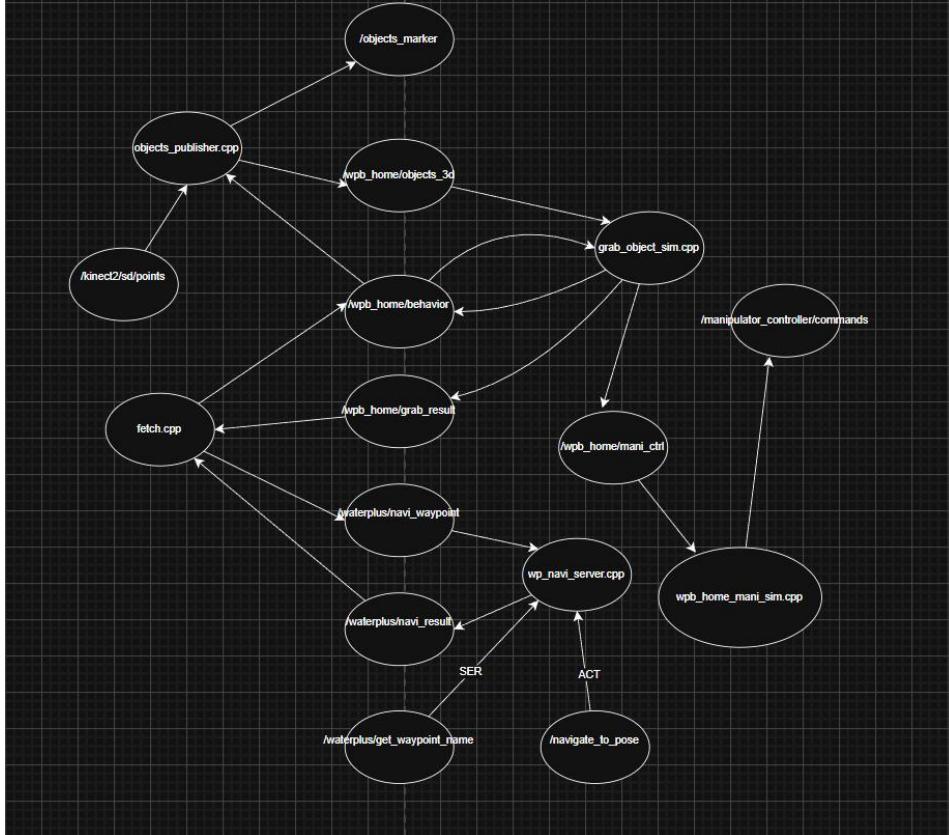
```
...  
    auto grab_result_sub = node->create_subscription<std_msgs::msg::String>(  
        "/wpb_home/grab_result",  
        10,  
        GrabResultCallback  
    );  
    rclcpp::Rate loop_rate(30);  
    while(rclcpp::ok())  
    {  
        if(fetch_step == STEP_WAIT)  
        {  
            std_msgs::msg::String msg;  
            msg.data = "id:1chan";  
            navi_pub->publish(msg);  
            fetch_step = STEP_GOTO_KITCHEN;  
            RCLCPP_INFO(node->get_logger(), "[STEP_WAIT] -> [STEP_GOTO_KITCHEN]");  
        }  
        rclcpp::spin_some(node);  
        loop_rate.sleep();  
    }  
    rclcpp::shutdown();  
    return 0;  
}
```

At line 69, the code sends the navigation coordinates to the navi_pub node...

```
...  
int main(int argc, char** argv)  
{  
    rclcpp::init(argc, argv);  
    node = std::make_shared<rclcpp::Node>("fetch_node");  
    navi_pub = node->create_publisher<std_msgs::msg::String>( //waterplus/navi_waypoint,  
    10 );  
    behavior_pub = node->create_publisher<std_msgs::msg::String>( //wpb_home/behavior,  
    10 );  
    auto navi_result_sub = node->create_subscription<std_msgs::msg::String>( "waterplus/navi_result",  
    10,  
    NaviResultCallback );  
    auto grab_result_sub = node->create_subscription<std_msgs::msg::String>( //wpb_home/grab_result,  
    10,  
    GrabResultCallback );
```

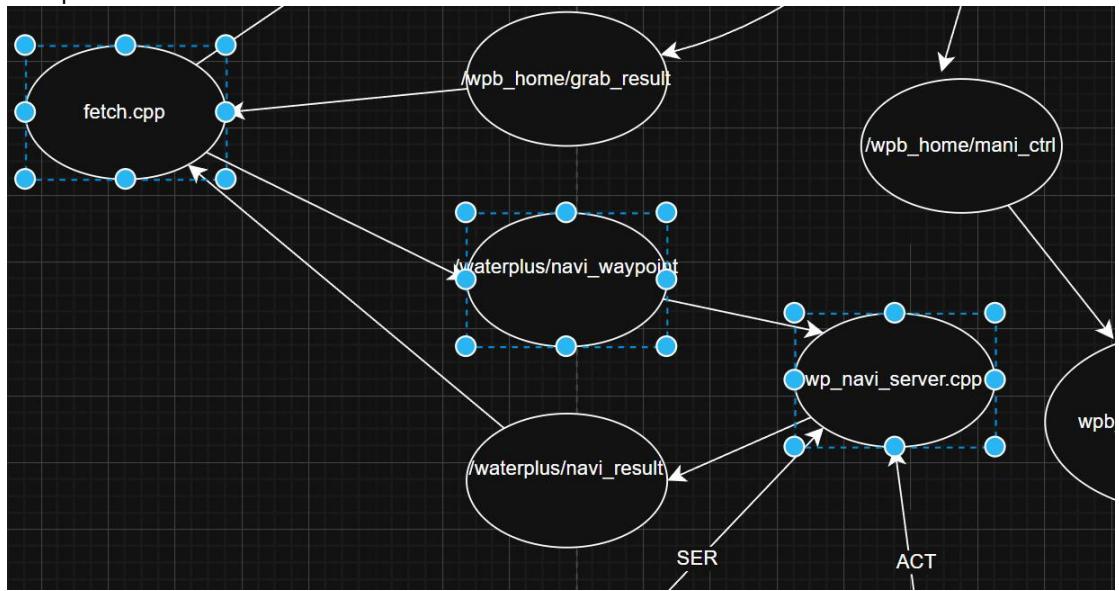
We need to find its actual handler.....

I spent a lot of time mapping out their relationships, which looks like this...



Maptnh@S-H4CK13

We need to follow the chain to locate the program that controls the manipulator.....

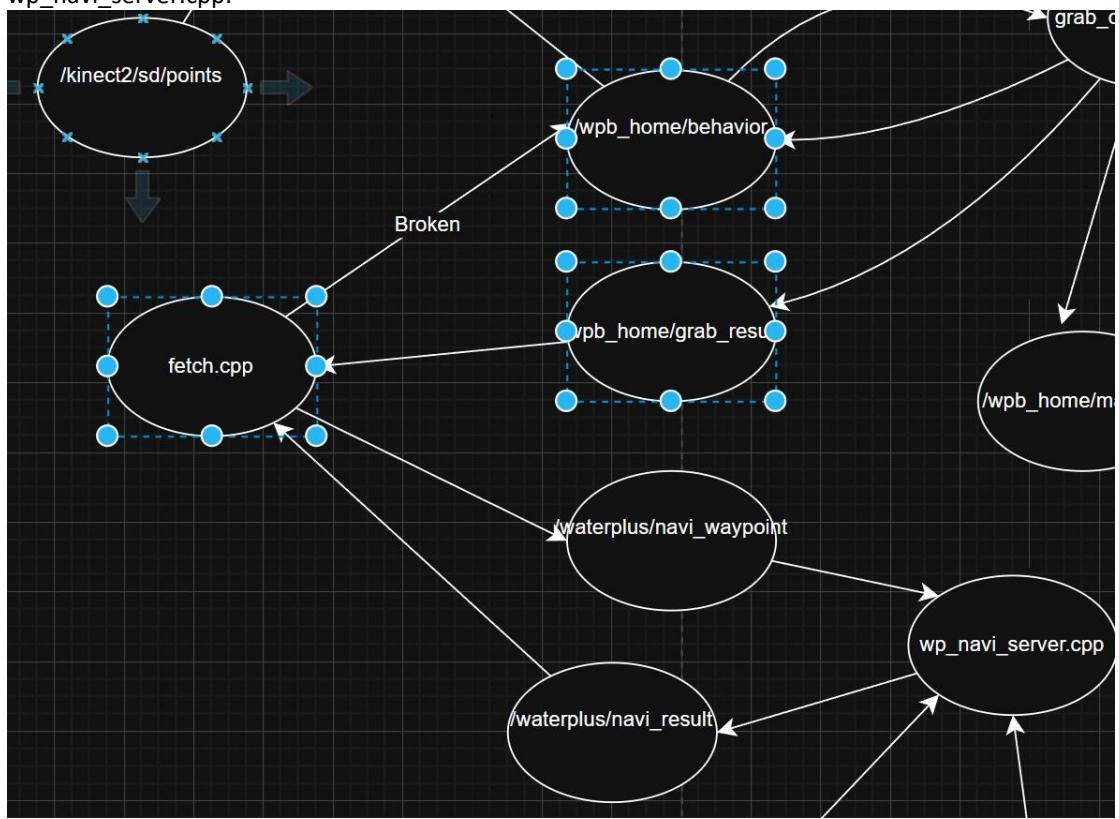


Upon receiving the execution point name, it will attempt to navigate.....

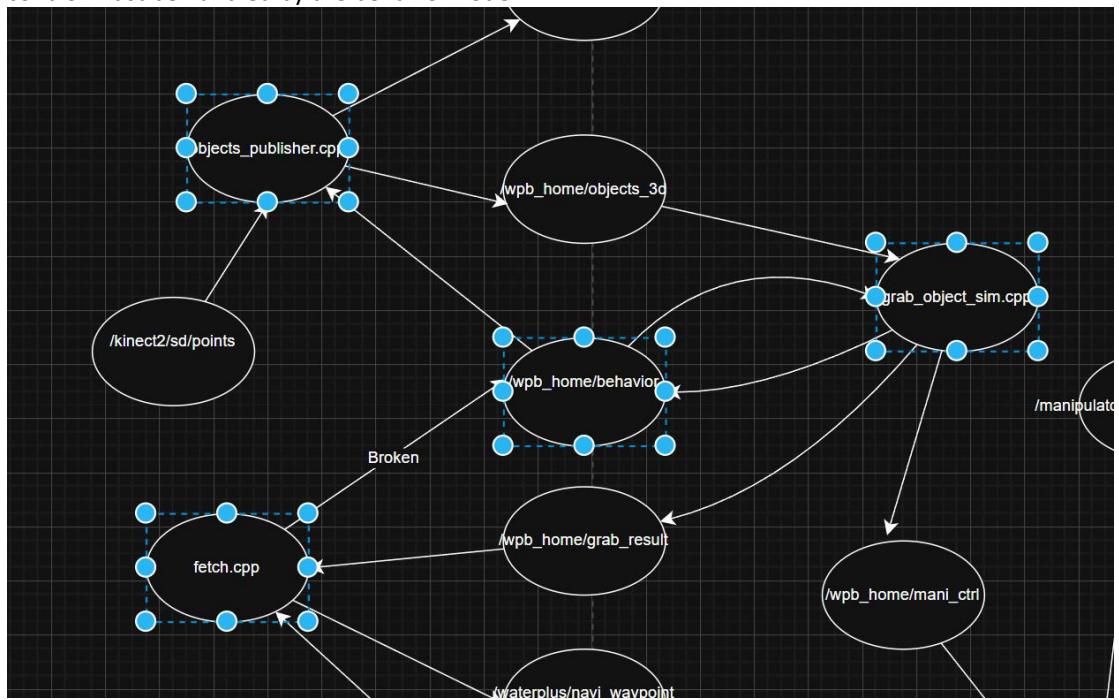
```
34     }
35
36     void resultCallback(const GoalHandleNavigateToPose::WrappedResult & result)
37     {
38         switch (result.code) {
39             case rclcpp_action::ResultCode::SUCCEEDED:
40                 RCLCPP_INFO(node->get_logger(), "Success!!!");
41                 result_msg.data = "navi done";
42                 break;
43             case rclcpp_action::ResultCode::ABORTED:
44                 RCLCPP_INFO(node->get_logger(), "Goal was aborted");
45                 result_msg.data = "navi aborted";
46                 break;
47             case rclcpp_action::ResultCode::CANCELED:
48                 RCLCPP_INFO(node->get_logger(), "Goal was canceled");
49                 result_msg.data = "navi canceled";
50                 break;
51             default:
52                 RCLCPP_INFO(node->get_logger(), "Unknown result code");
53                 result_msg.data = "navi Unknown result";
54                 break;
55         }
56         result_pub->publish(result_msg);
57     }
```

The data is then returned to fetch.cpp for the next grasping operation.

Therefore, our control target is not in `wp_navi_server.cpp`.



By excluding the navi node in fetch.cpp, we are left with the grab_result and behavior nodes. Further excluding grab_result (which typically only receives results), it is clear that fetch's grasping control must be handled by the behavior node...



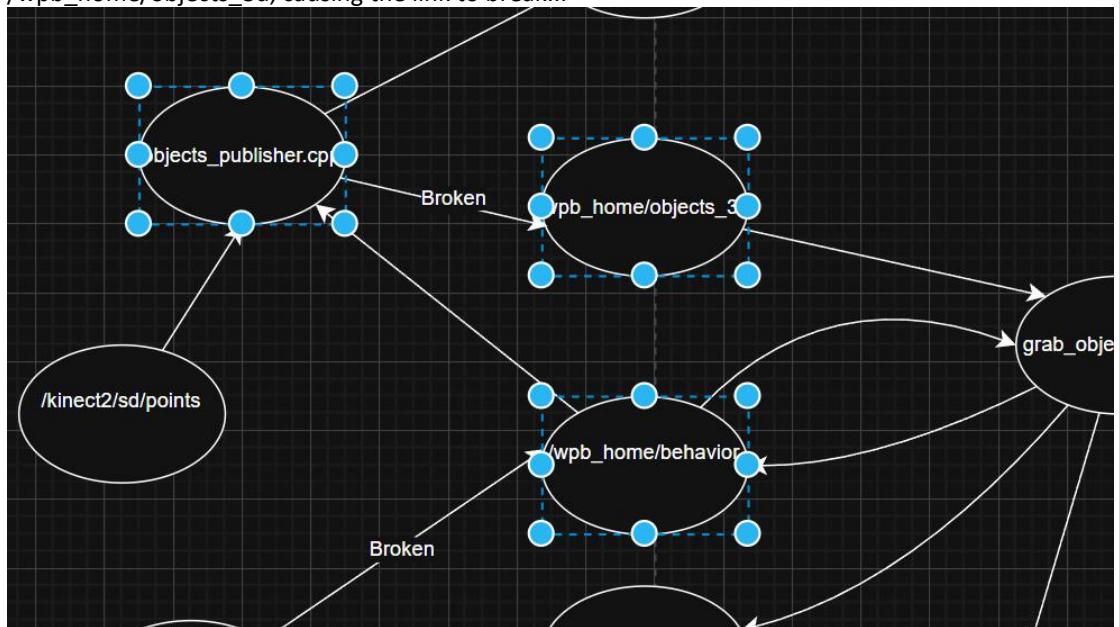
The `/wpb_home/behavior` node is linked to `objects_publisher.cpp` and `grab_object_sim.cpp`. `objects_publisher.cpp` handles the recognition of object point clouds.

Maptnh@S-H4CK13

The screenshot shows a ROS2 workspace named 'wpr_simulation2'. The file tree on the left includes 'src' (containing 'wpr_simulation2', 'scripts', and 'src' with files like 'ball_random_move.cpp', 'face_detector.py', 'grab_object_sim.cpp', 'keyboard_vel_cmd.cpp', 'objects_publisher.cpp', 'wpb_home_mani_sim.cpp'), 'worlds', 'CMakeLists.txt', 'package.xml', 'README.md', 'setup.cfg', 'setup.py', '.data', 'posegraph', 'bypass.py', 'command', and 'waypoints.yaml'. The right pane shows the code for 'objects_publisher.cpp' with lines 61 to 192 visible. The code handles point accumulation and box marker definition based on cluster indices.

```
61     private:
76         {
165             int point_index = cluster_indices[i].indices[j];
166             points_x_sum += cloud_src.points[point_index].x;
167             points_y_sum += cloud_src.points[point_index].y;
168             points_z_sum += cloud_src.points[point_index].z;
169         }
170         float points_x_sum = 0;
171         float points_y_sum = 0;
172         float points_z_sum = 0;
173         bool bFirstPoint = true;
174         for(int j = 0 ; j < point_num ; j++)
175         {
176             int point_index = cluster_indices[i].indices[j];
177             points_x_sum += cloud_src.points[point_index].x;
178             points_y_sum += cloud_src.points[point_index].y;
179             points_z_sum += cloud_src.points[point_index].z;
180         }
181         pcl::PointXYZRGB p = cloud_src.points[point_index];
182         if(bFirstPoint == true)
183         {
184             boxMarker.xMax = boxMarker.xMin = p.x;
185             boxMarker.yMax = boxMarker.yMin = p.y;
186             boxMarker.zMax = boxMarker.zMin = p.z;
187             bFirstPoint = false;
188         }
189         if(p.x < boxMarker.xMin) { boxMarker.xMin = p.x; }
190         if(p.x > boxMarker.xMax) { boxMarker.xMax = p.x; }
191         if(p.y < boxMarker.yMin) { boxMarker.yMin = p.y; }
192         if(p.y > boxMarker.yMax) { boxMarker.yMax = p.y; }
193         if(p.z < boxMarker.zMin) { boxMarker.zMin = p.z; }
194         if(p.z > boxMarker.zMax) { boxMarker.zMax = p.z; }
```

Failure to detect the object prevents the coordinate data from being transmitted to /wpb_home/objects_3d, causing the link to break...



Maptnh@S-H4CK13

OPEN EDITORS 1 unsaved

ROS2_WS

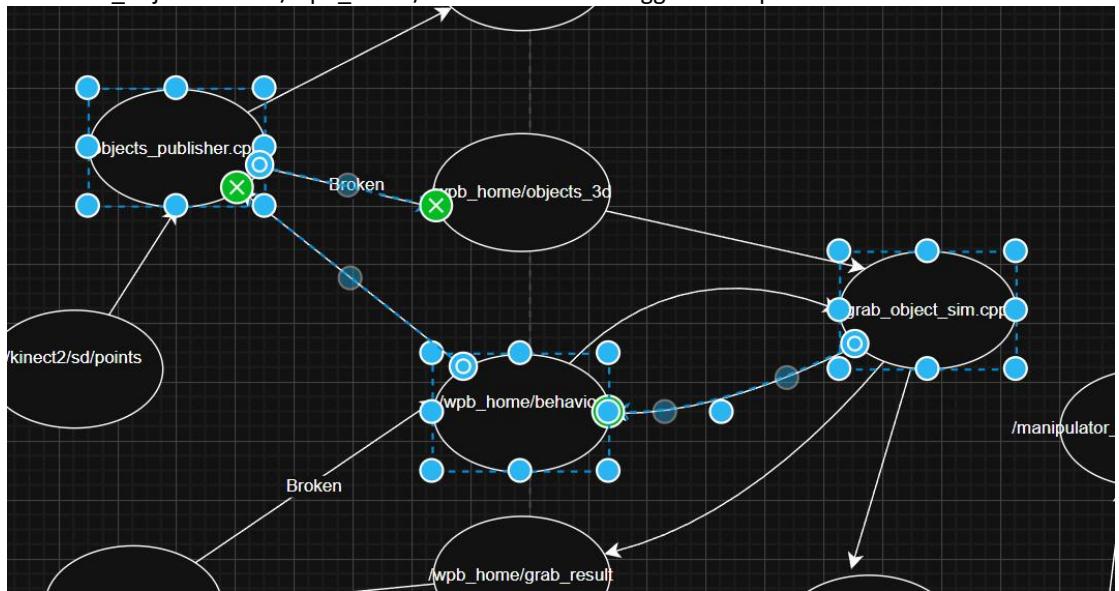
src > wpr_simulation2 > src > grab_object_sim.cpp

```
25     float object_y = 0.0;
26     float object_z = 0.0;
27     int count = 0;
28
29     float align_x = 1.0;
30     float align_y = 0.0;
31
32 void BehaviorCallback(const std::msgs::msg::SharedPtr msg)
33 {
34     if(grab_step == STEP_WAIT && msg->data == "start_grab")
35     {
36         std::msgs::msg::String msg;
37         msg.data = "start objects";
38         behavior_pub->publish(msg);
39         count = 0;
40         grab_step = STEP_FIND_OBJ;
41     }
42 }
43
44 void ObjectCallback(const wpr_simulation2::msg::Object::SharedPtr msg)
45 {
46     if(grab_step == STEP_FIND_OBJ)
47     {
48         object_x = msg->x[0];
49         object_y = msg->y[0];
50         object_z = msg->z[0];
51         grab_step = STEP_ALIGN_OBJ;
52     }
53 }
```

OUTLINE

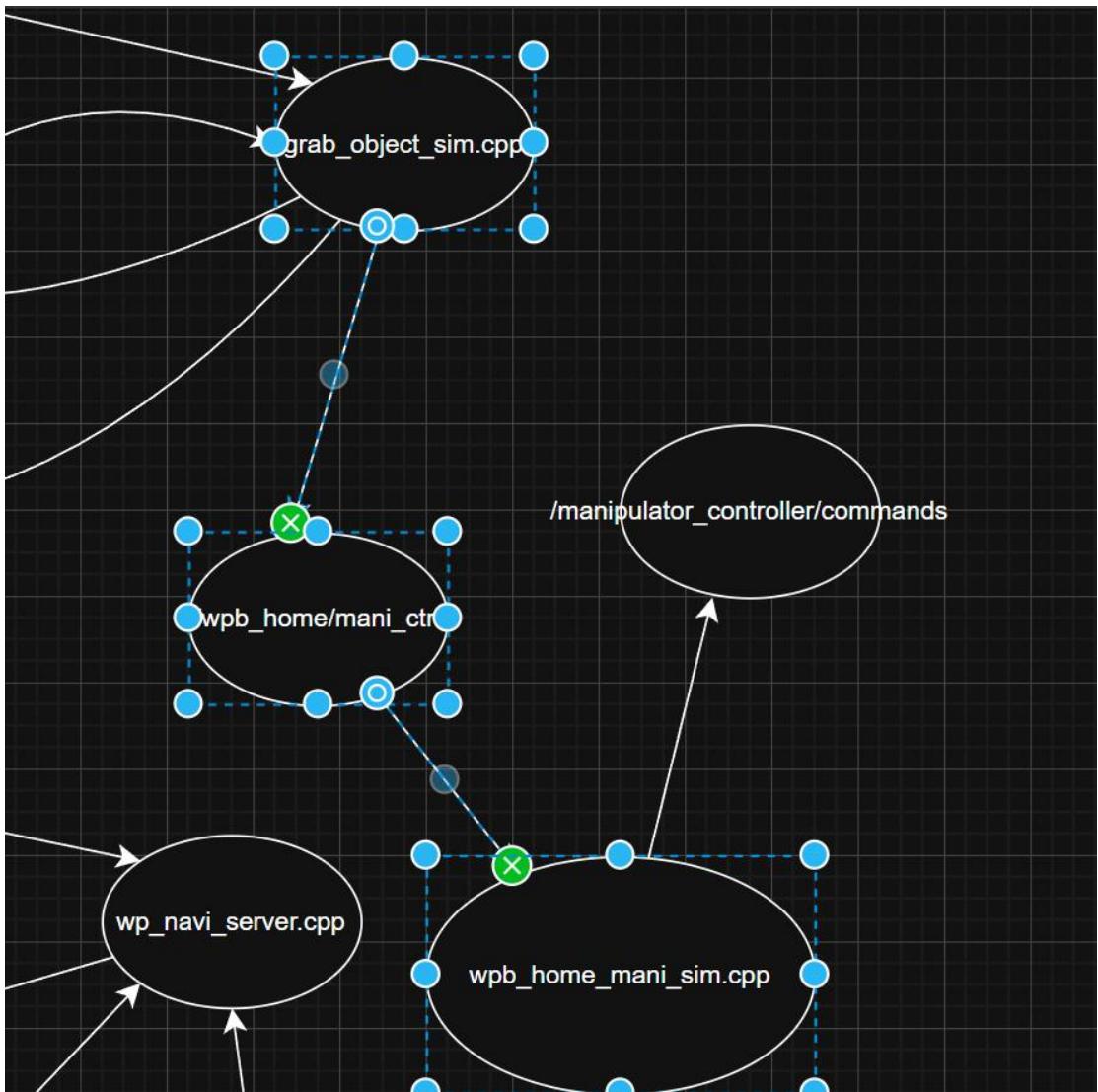
src > wpr_simulation2 > src > ball_random_move.cpp
src > wpr_simulation2 > scripts
src > wpr_simulation2 > src > face_detector.py
src > wpr_simulation2 > src > grab_object_sim.cpp
src > wpr_simulation2 > src > keyboard_vel_cmd.cpp
src > wpr_simulation2 > src > objects_publisher.cpp
src > wpr_simulation2 > src > wpb_home_mani_sim.cpp
src > worlds
src > wpr_simulation2
CMakeLists.txt
package.xml
README.md
setup.cfg
setup.py
.data
.posegraph
bypass.py
command
waypoints.yaml

Sent start_objects to the /wpb_home/behavior node to trigger startup.



In reality, this approach is ineffective, as the object coordinates cannot be retrieved...

Maptnh@S-H4CK13



Follow through to
wpb_home_mani_sim.cpp.

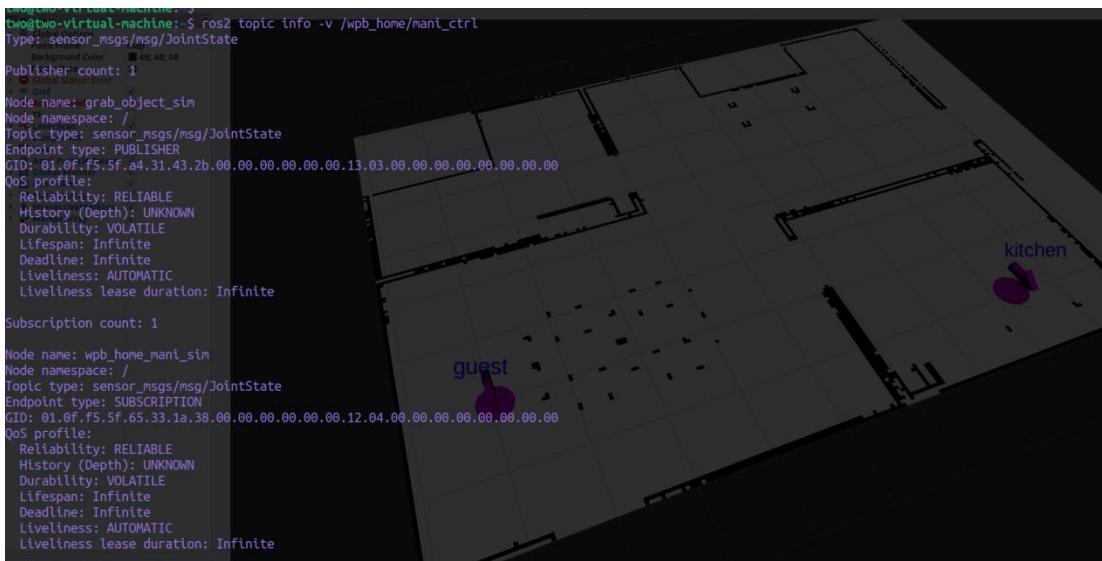
```
61    {
62        continue;
63    }
64    if(grab_step == STEP_HAND_UP)
65    {
66        RCLCPP_INFO(node->get_logger(), "[STEP_HAND_UP]");
67        sensor_msgs::msg::JointState mani_msg;
68        mani_msg.name.resize(2);
69        mani_msg.name[0] = "lift";
70        mani_msg.name[1] = "gripper";
71        mani_msg.position.resize(2);
72        mani_msg.position[0] = object_z;
73        mani_msg.position[1] = 0.15;
74        mani_pub->publish(mani_msg);
75        rclcpp::sleep_for(std::chrono::milliseconds(8000));
76        grab_step = STEP_FORWARD;
77        continue;
78    }
79    if(grab_step == STEP_FORWARD)
80    {
81        RCLCPP_INFO(node->get_logger(), "[STEP_FORWARD] object_x = %.2f", object_x);
82        geometry_msgs::msg::Twist vel_msg;
83        vel_msg.linear.x = 0.1;
84        vel_msg.linear.y = 0;
85        vel_pub->publish(vel_msg);
86        int forward_duration = (object_x - 0.65) * 20000;
87        rclcpp::sleep_for(std::chrono::milliseconds(forward_duration));
88        grab_step = STEP_GRAB;
89    }
90}
```

Fortunately, we located the manipulator control logic...

Now we inspect the node details.

\$ ros2 topic info -v /wpb_home/mani_ctrl

Maptnh@S-H4CK13



1. name: string[]

The names of all robot joints.

Length = number of joints.

Example: ["shoulder_pan", "shoulder_lift", "elbow", ...]

You can find the corresponding joint names in `grab_object_sim.cpp` (lines 76–191).

2. position: float64[]

The angles or linear displacements of each joint.

Units: usually radians or meters (for linear sliders).

Length must match name[].

Do not leave empty → this will cause errors or the robot won't move.

3. velocity: float64[]

The velocity of each joint (rad/s or m/s).

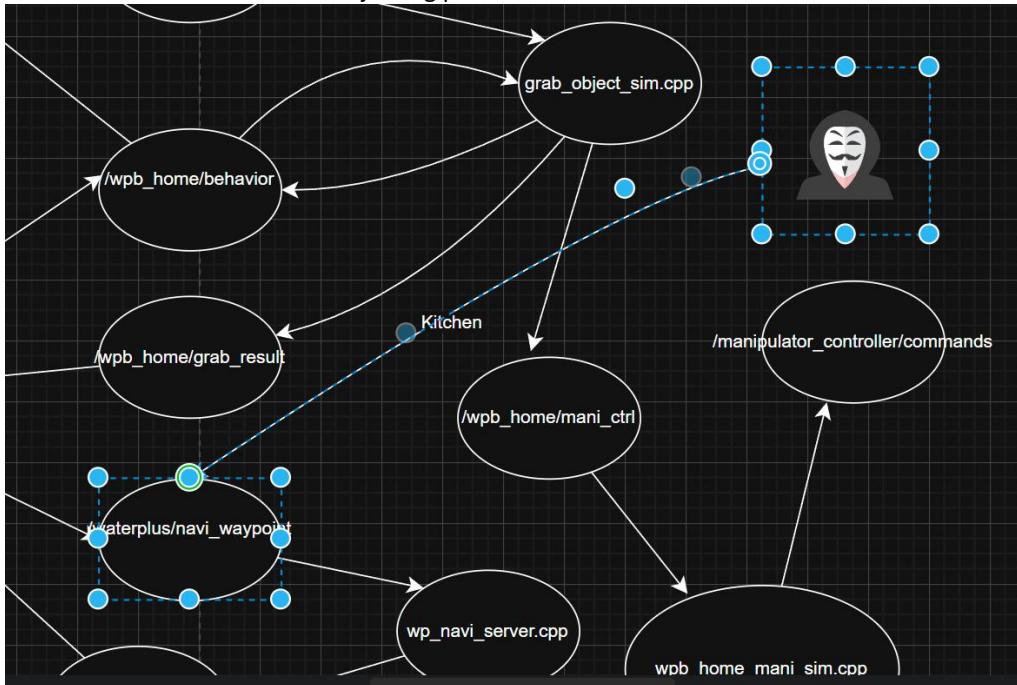
Can be left empty → use an empty array if not needed.

4. effort: float64[]

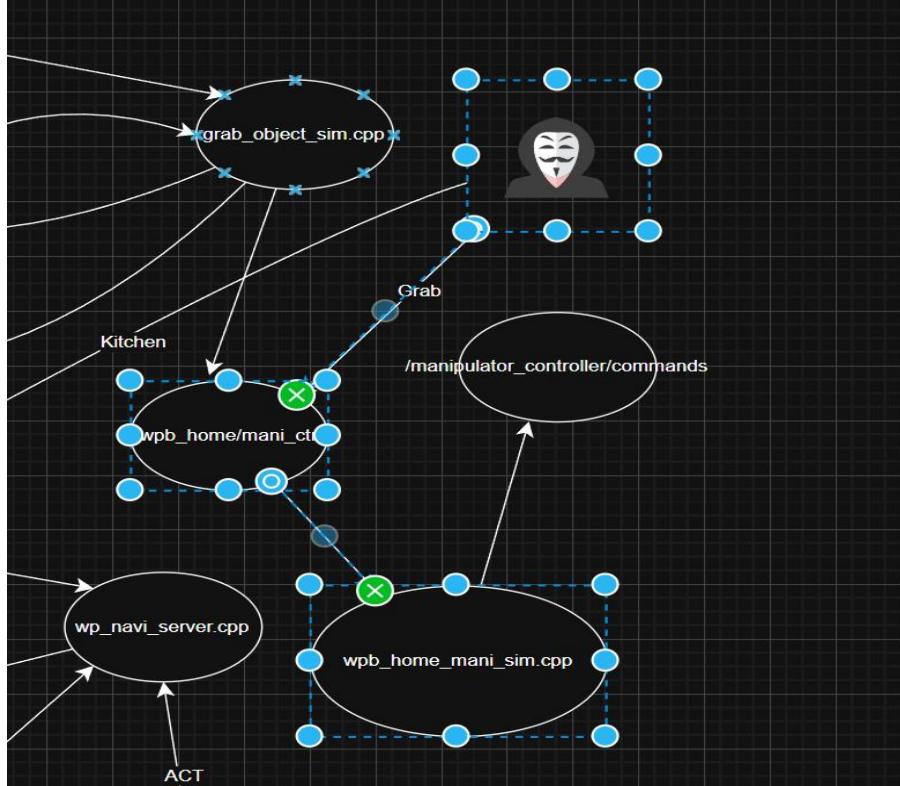
The torque or force of each joint (N·m or N).

Usually not needed in simulation → can be an empty array.

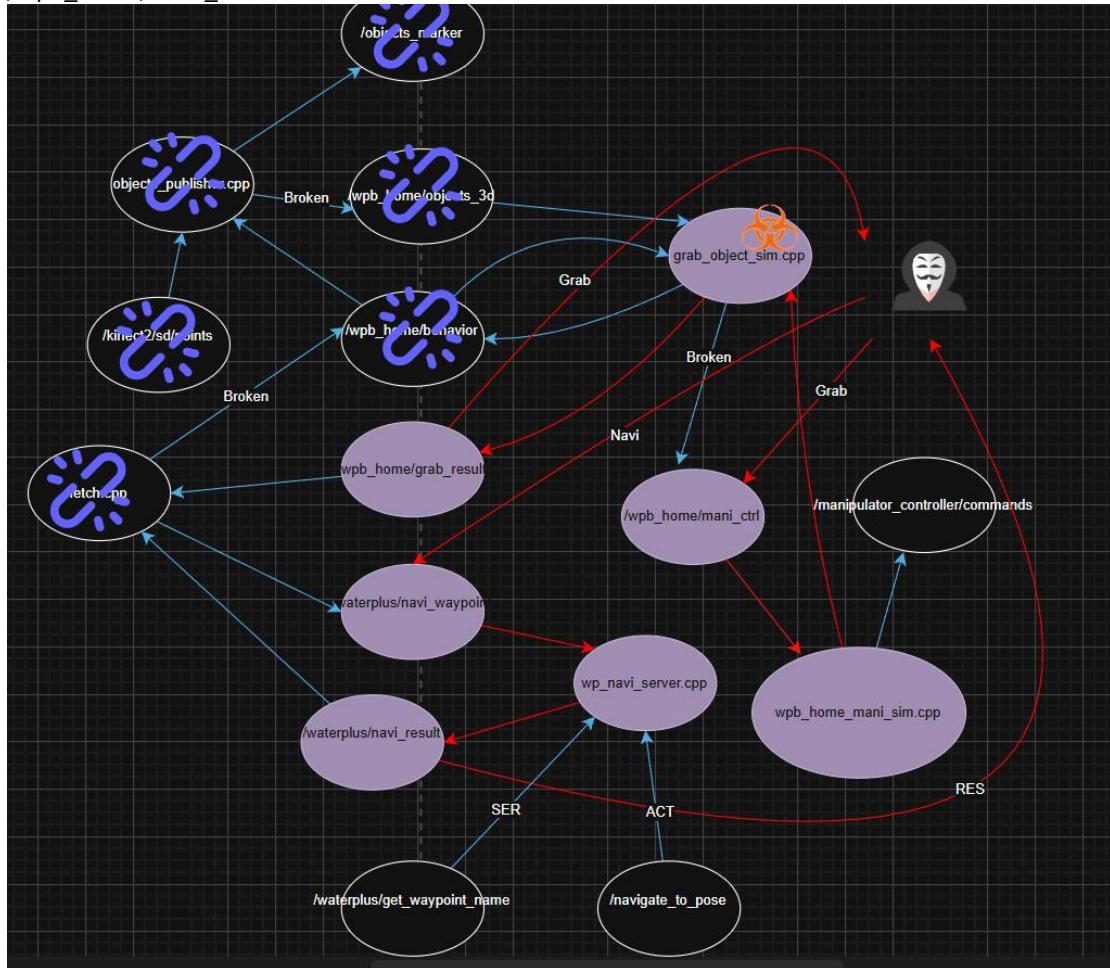
Now we need to construct our hijacking path...



Send the target location and let the robot move to the kitchen...



Send control data directly to
`/wpb_home/mani_ctrl`.



Maptnh@S-H4CK13

A complete hijacking chain....executed through scripts.

```
-----  
#!/usr/bin/env python3
```

```
# Maptnh
```

```
import rclpy
```

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

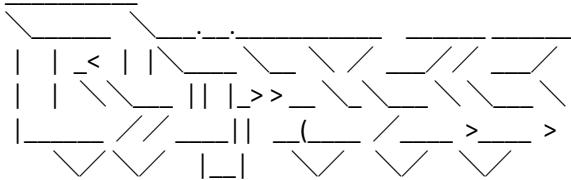
```
from std_msgs.msg import String
```

```
from sensor_msgs.msg import JointState
```

```
from geometry_msgs.msg import Twist
```

```
import time
```

```
LOGO = ""
```



```
Maptnh@S-H4CK13 https://github.com/MartinxMax""
```

```
class GrabAndDeliver(Node):
```

```
    def __init__(self):
```

```
        super().__init__('grab_and_deliver_node')
```

```
        self.target_lift = 0.80
```

```
        self.gripper_open = 0.15
```

```
        self.gripper_close = 0.07
```

```
        self.lift_up_delta = 0.05
```

```
        self.forward_duration = 3.0
```

```
        self.backward_duration = 3.0
```

```
        self.sleep_short = 1.0
```

```
        self.sleep_long = 4.0
```

```
        self.navi_pub = self.create_publisher(String, '/waterplus/navi_waypoint', 10)
```

```
        self.mani_pub = self.create_publisher(JointState, '/wpb_home/mani_ctrl', 10)
```

```
        self.cmd_vel_pub = self.create_publisher(Twist, '/cmd_vel', 10)
```

```
        self.navi_done = False
```

```
        self.sub = self.create_subscription(String, '/waterplus/navi_result', self.navi_callback, 10)
```

```
    def navi_callback(self, msg):
```

```
        if msg.data == 'navi done':
```

```
            self.navi_done = True
```

```
    def wait_for_navi_done(self, target_name):
```

```
        self.get_logger().info(f'[INFO] Waiting to navigate to {target_name}...')
```

```
        while rclpy.ok() and not self.navi_done:
```

```
            rclpy.spin_once(self, timeout_sec=0.1)
```

```
            self.navi_done = False
```

```
            self.get_logger().info(f'[INFO] Arrived at {target_name}')
```

```
    def publish_mani(self, lift, gripper):
```

```
        msg = JointState()
```

```
        msg.name = ['lift', 'gripper']
```

```
        msg.position = [lift, gripper]
```

```
        self.mani_pub.publish(msg)
```

```
    def publish_cmd_vel(self, linear_x):
```

```
        msg = Twist()
```

Maptnh@S-H4CK13

```
msg.linear.x = linear_x
msg.linear.y = 0.0
msg.linear.z = 0.0
msg.angular.x = 0.0
msg.angular.y = 0.0
msg.angular.z = 0.0
self.cmd_vel_pub.publish(msg)

def run_sequence(self):
    self.get_logger().info('[1] Publishing navigation goal: kitchen')
    self.navi_pub.publish(String(data='kitchen'))
    self.wait_for_navi_done('kitchen')

    self.publish_mani(self.target_lift, self.gripper_open)
    time.sleep(self.sleep_long)

    self.publish_cmd_vel(0.1)
    time.sleep(self.forward_duration)

    self.publish_cmd_vel(0.0)
    time.sleep(self.sleep_short)

    self.publish_mani(self.target_lift, self.gripper_close)
    time.sleep(self.sleep_long)

    target_lift_up = self.target_lift + self.lift_up_delta
    self.publish_mani(target_lift_up, self.gripper_close)
    time.sleep(self.sleep_long)

    self.publish_cmd_vel(-0.1)
    time.sleep(self.backward_duration)

    self.publish_cmd_vel(0.0)
    time.sleep(self.sleep_short)

    self.get_logger().info('[2] Publishing navigation goal: guest')
    self.navi_pub.publish(String(data='guest'))
    self.wait_for_navi_done('guest')

    self.publish_mani(target_lift_up, self.gripper_open)
    time.sleep(self.sleep_long)

    self.get_logger().info('== Sequence Completed ==')

def main(args=None):
    print(LOGO)
    rclpy.init(args=args)
    node = GrabAndDeliver()
    node.run_sequence()
    node.destroy_node()
    rclpy.shutdown()

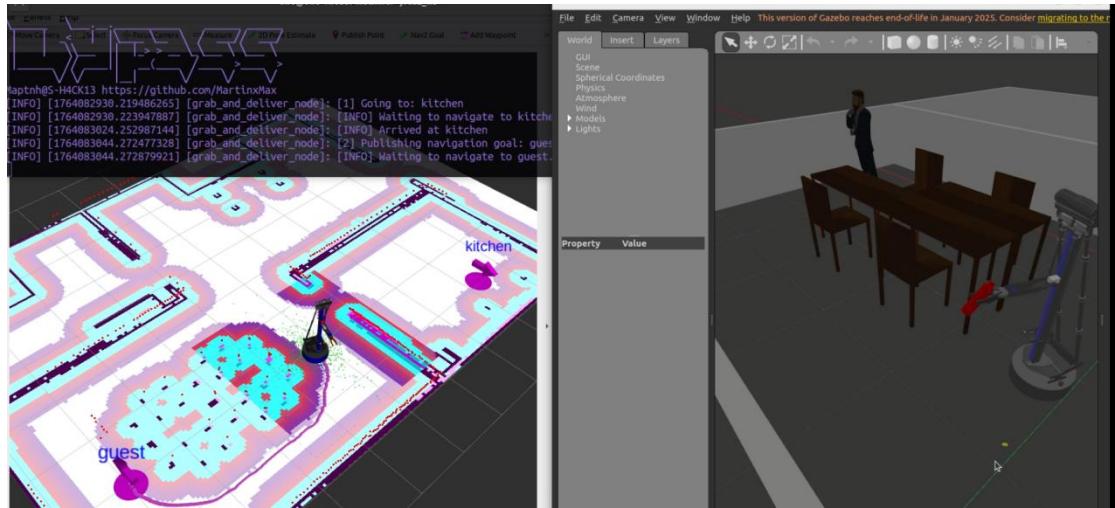
if __name__ == '__main__':
    main()


---


$ source install/setup.bash;ros2 launch home_pkg home.launch.py
$ python3 bypass.py
```

Maptnh@S-H4CK13

The parameters in the script are all pre-calculated. For grasping objects in different positions, you usually need to adjust the waypoints.yaml accordingly. The object's position should align with the robot's X-axis.



It started to behave exactly as we scripted it...

