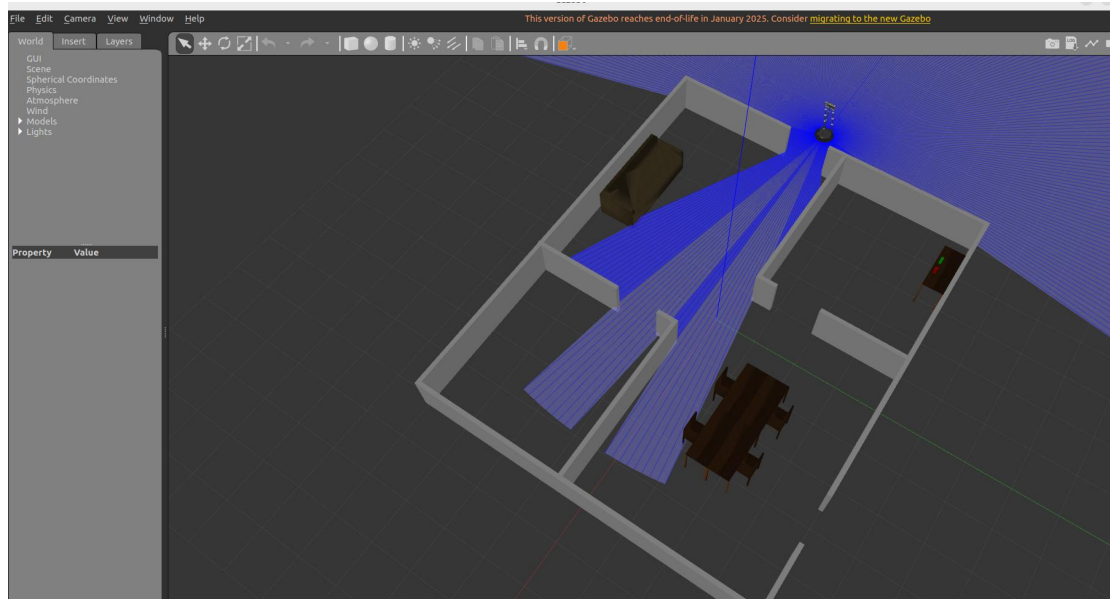


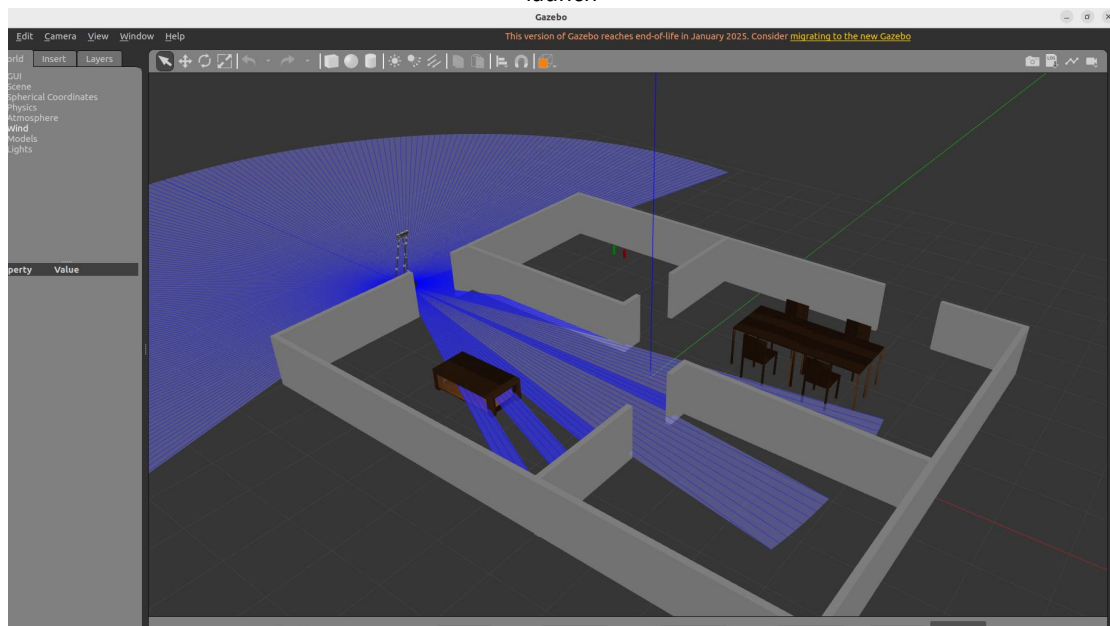
ROS2 Simulation Environment Loading Issue

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...

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

```

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     )

```

r than all the data in the transform cache' [rviz2-6] [INFO] [1763977432.645782668] [rviz2]: Message Filter dropping message: frame 'laser_link' at time 267.027 for reason 'the timestamp on th
 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 th

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

```

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

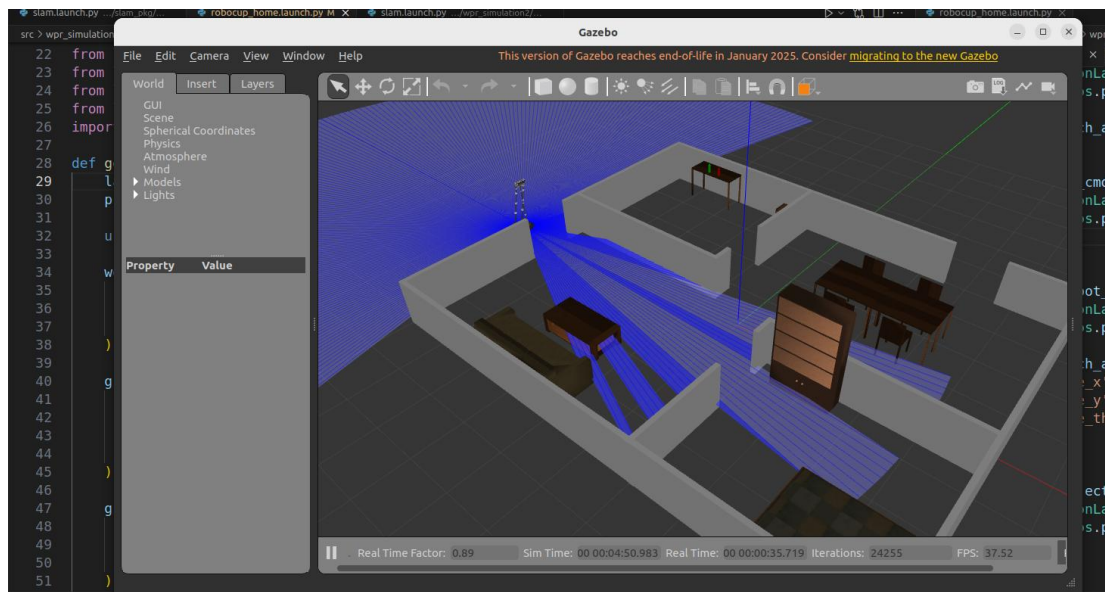
Check the two MD5 values — it's clear that they are unrelated to the map loading error...

```

two@two-virtual-machine: ~/ros2_ws
$ find ./ -name robocup_home.world
./src/wpr_simulation2/worlds/robocup_home.world
d = ./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 35a94f0b7ca975482e774287a46763ce ./src/wpr_simulation2/worlds/robocup_home.world
'ro 35a94f0b7ca975482e774287a46763ce ./install/wpr_simulation2/share/wpr_simulation2/worlds/robocup_home.world
two@two-virtual-machine:~/ros2_ws$
rver_r_cmd = IncludeLaunchDescription(
PythonLaunchDescriptionSource(
os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')

```

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



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

launch

ch docs

on API

ard Documents

umentation

elcome to launch's umentation!

Architecture of launch

ices and tables

x

Documentation / Welcome to launch's documentation! / Architecture of launch

View page source

Architecture of *launch*

launch is designed to provide core features like describing actions (e.g. executing a process or including another launch description), generating events, introspecting launch descriptions, and executing launch descriptions. At the same time, it provides extension points so that the set of things that these core features can operate on, or integrate with, can be expanded with additional packages.

Launch Entities and Launch Descriptions

The main object in *launch* is the `launch.LaunchDescriptionEntity`, from which other entities that are "launched" inherit. This class, or more specifically classes derived from this class, are responsible for capturing the system architect's (a.k.a. the user's) intent for how the system should be launched, as well as how *launch* itself should react to asynchronous events in the system during launch. A launch description entity has its `launch.LaunchDescriptionEntity.visit()` method called during "launching", and has any of the "describe" methods called during "introspection". It may also provide a `asyncio.Future` with the `launch.LaunchDescriptionEntity.get_asyncio_future()` method, if it has on-going asynchronous activity after returning from `visit`.

When visited, entities may yield additional entities to be visited, and this pattern is used from the "root" of the launch, where a special entity called `launch.LaunchDescription` is provided to start the

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>

https://docs.ros.org/en/foxy/Tutorials/Intermediate/Using-Event-Handlers.html

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

tr2

Testing

URDF

Advanced

Demos

Miscellaneous

How-to Guides

Concepts

Contact

The ROS 2 Project

Related Projects

Glossary

Citations

Using Event Handlers.html

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.

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

```
RegisterEventHandler(
    OnExecutionComplete(
        target_action=spawn_turtle,
        on_completion=[
            logInfo(msg="Spawn finished'"),
            change_background_r,
            TimerAction(
                period=2.0,
                actions=[change_background_r_conditioned],
            )
        ]
    )
),
```

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 `ExitEvent` 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"),
```

Try modifying `robocup_home.launch.py` by adding `from launch.actions import IncludeLaunchDescription, TimerAction` at line 26.

```

ROS2_WIS 16 # Fix By Maptnh
17
18 > python_pkg
19 import os
20 > slam_pkg
21 from ament_index_python.packages import get_package_share_directory
22 > include
23 from launch import LaunchDescription
24 > launch
25 from launch.actions import IncludeLaunchDescription
26 > slam.launch.py
27 from launch.launch_description_sources import PythonLaunchDescriptionSource
28 > src
29 from launch.substitutions import LaunchConfiguration
30 > CMakeLists.txt
31 from launch.actions import Node
32 > package.xml
33 from launch.actions import OpaqueFunction
34 > topic_pkg
35 from launch.actions import IncludeLaunchDescription, TimerAction
36 > vel_pkg
37
38 > wp_map_tools
39
40 > wpr_simulation2
41
42 > config
43 def generate_launch_description():
44     launch_file_dir = os.path.join(get_package_share_directory('wpr_simulation2'), 'launch')
45     pkg_gazebo_ros = get_package_share_directory('gazebo_ros')
46
47     use_sim_time = LaunchConfiguration('use_sim_time', default='true')
48
49     world = os.path.join(
50         get_package_share_directory('wpr_simulation2'),
51         'worlds',
52         'robocup_home.world'
53     )

```

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...

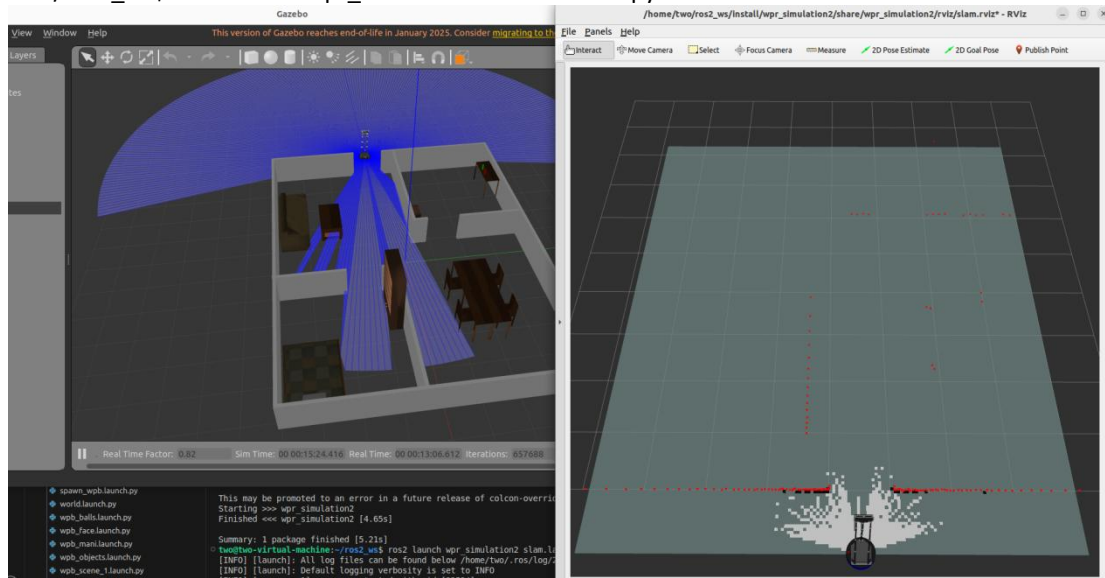

```

28 def generate_launch_description():
    pose_init = Pose(
        pose_init: 0.0
    ).items()
    spawn_objects = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(launch_file_dir, 'spawn_objects.launch.py')
        )
    )
    delayed_spawn_robot_cmd = TimerAction(
        period=20.0,
        actions=[spawn_robot_cmd]
    )
    delayed_spawn_objects = TimerAction(
        period=20.0,
        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
    
```

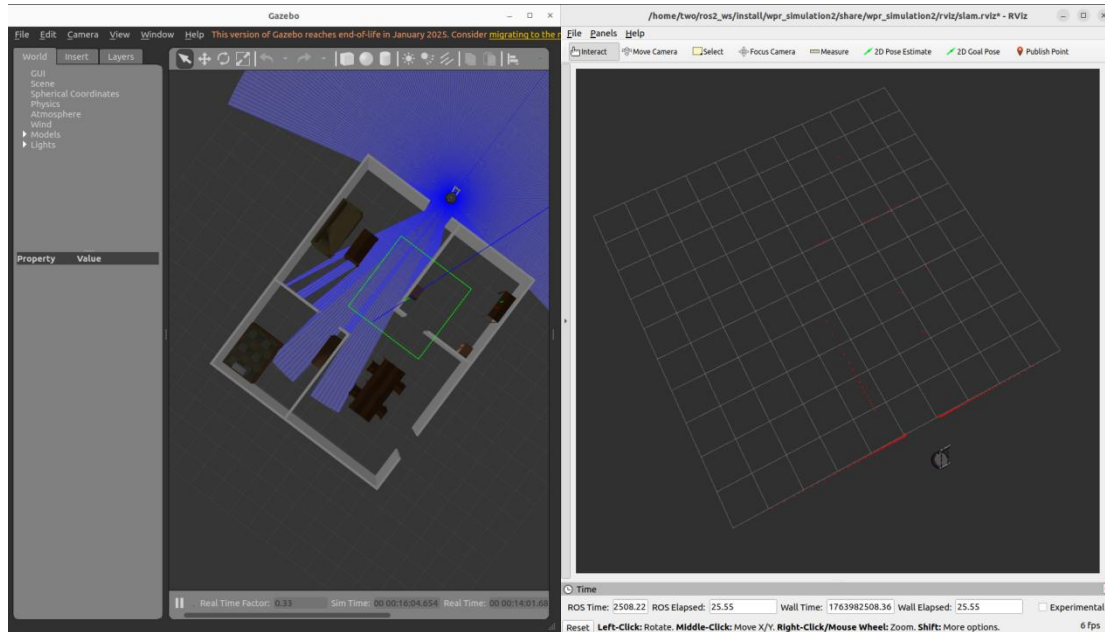
Compiling the code...

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

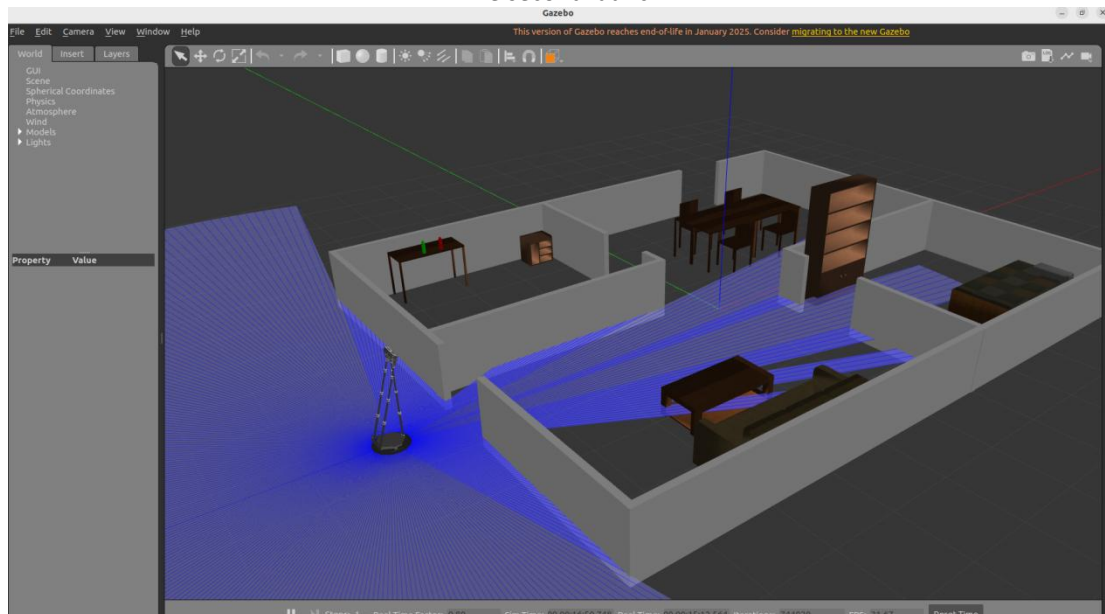
Xx:~/ros2_ws\$ ros2 launch wpr_simulation2 slam.launch.py



The first launch



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.
```

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'
    )

    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')
        )
    )

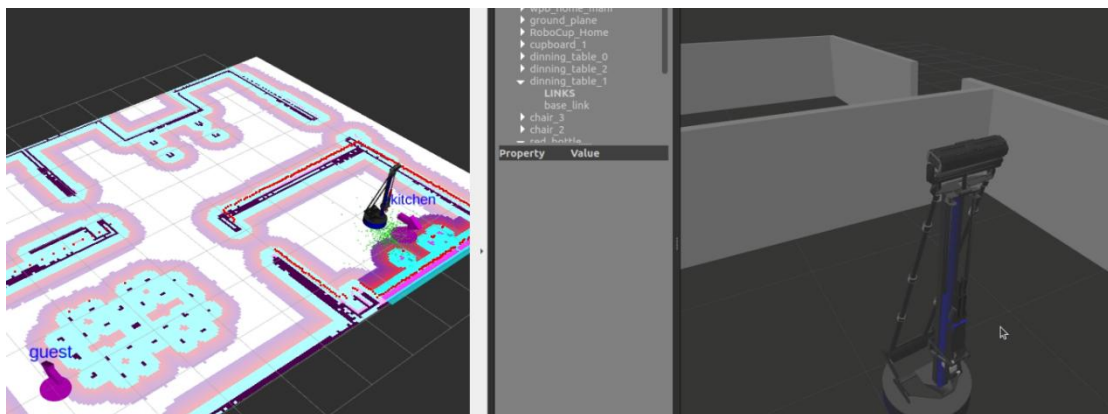
    delayed_spawn_robot_cmd = TimerAction(
        period=40.0,
        actions=[spawn_robot_cmd]
    )
```

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

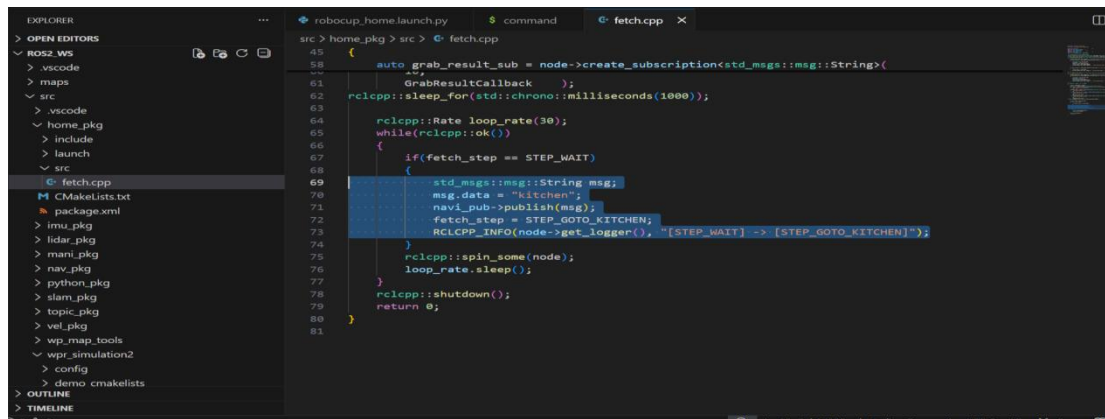
The original implementation overlooked that starting numerous programs at once could immediately trigger memory overflow, resulting in system failure.

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

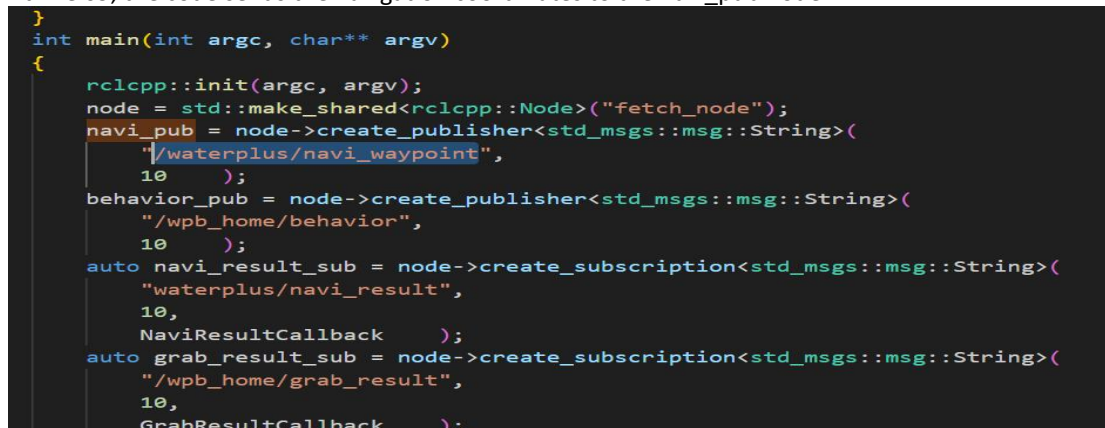


```

45 {
46     auto grab_result_sub = node->create_subscription<std_msgs::msg::String>(
47         GrabResultCallback );
48     rclcpp::sleep_for(std::chrono::milliseconds(1000));
49     rclcpp::Rate loop_rate(30);
50     while(rclcpp::ok())
51     {
52         if(fetch_step == STEP_WAIT)
53         {
54             std_msgs::msg::String msg;
55             msg.data = "kitchen";
56             navi_pub->publish(msg);
57             fetch_step = STEP_GOTO_KITCHEN;
58             RCLCPP_INFO(node->get_logger(), "[STEP_WAIT] -> [STEP_GOTO_KITCHEN]");
59         }
60         rclcpp::spin_some(node);
61         loop_rate.sleep();
62     }
63     rclcpp::shutdown();
64     return 0;
65 }

```

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



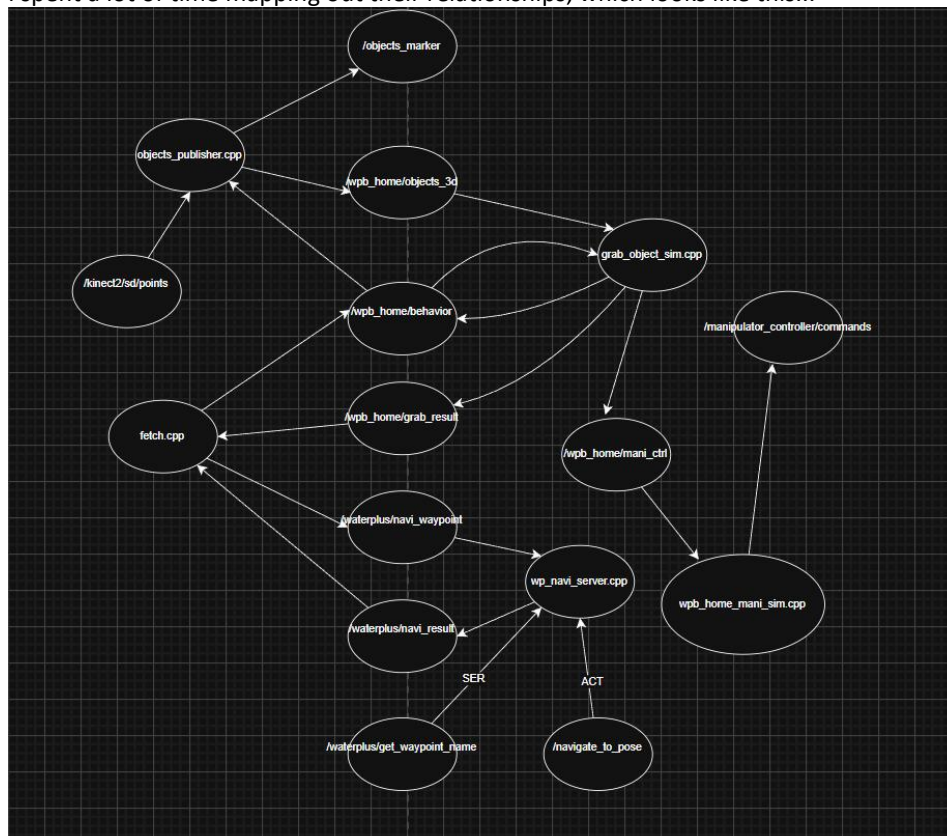
```

}
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 );
}

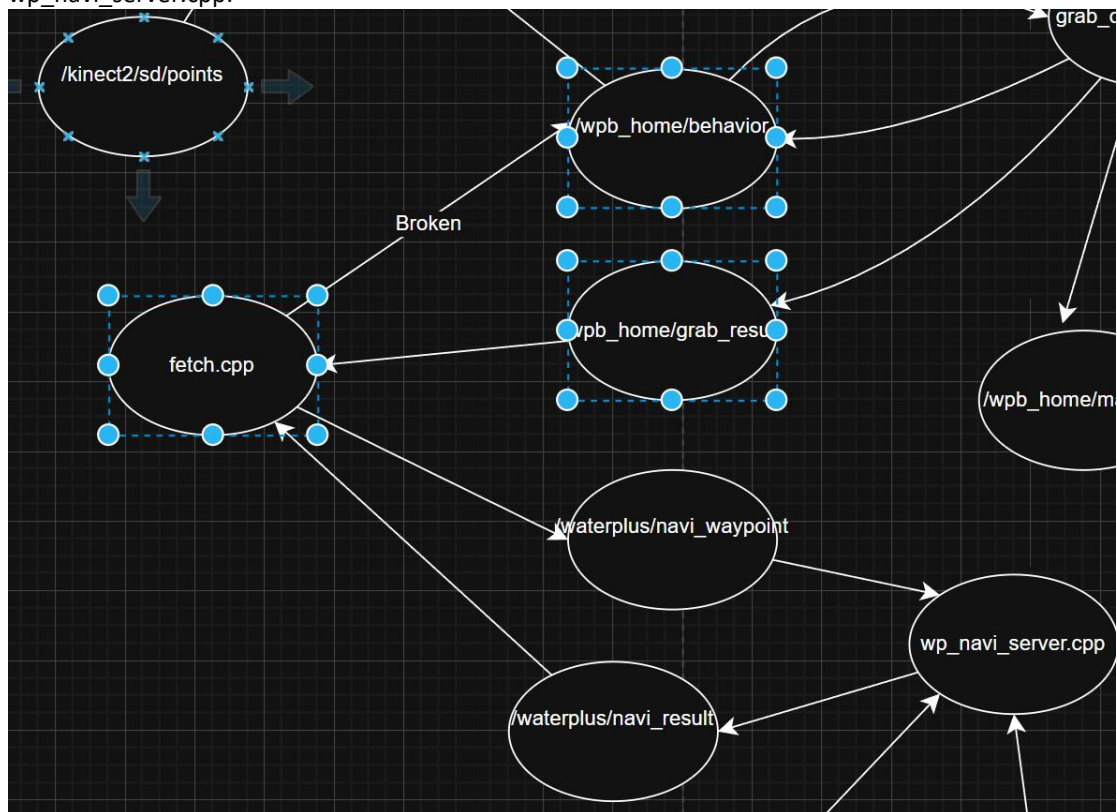
```

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

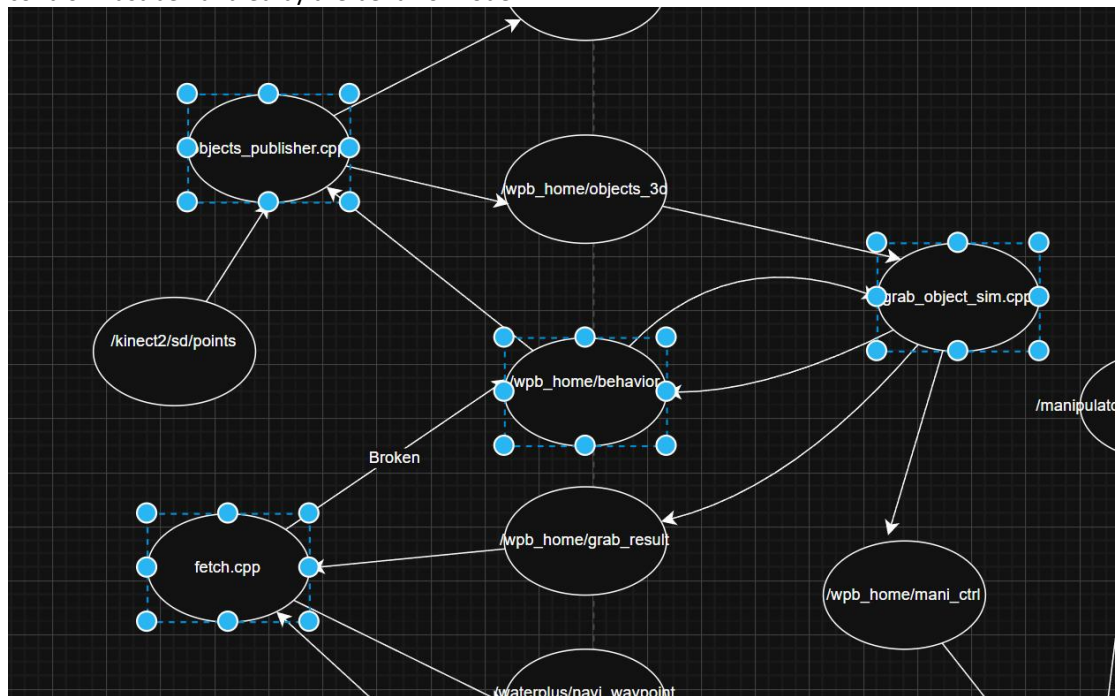
I spent a lot of time mapping out their relationships, which looks like this...



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...



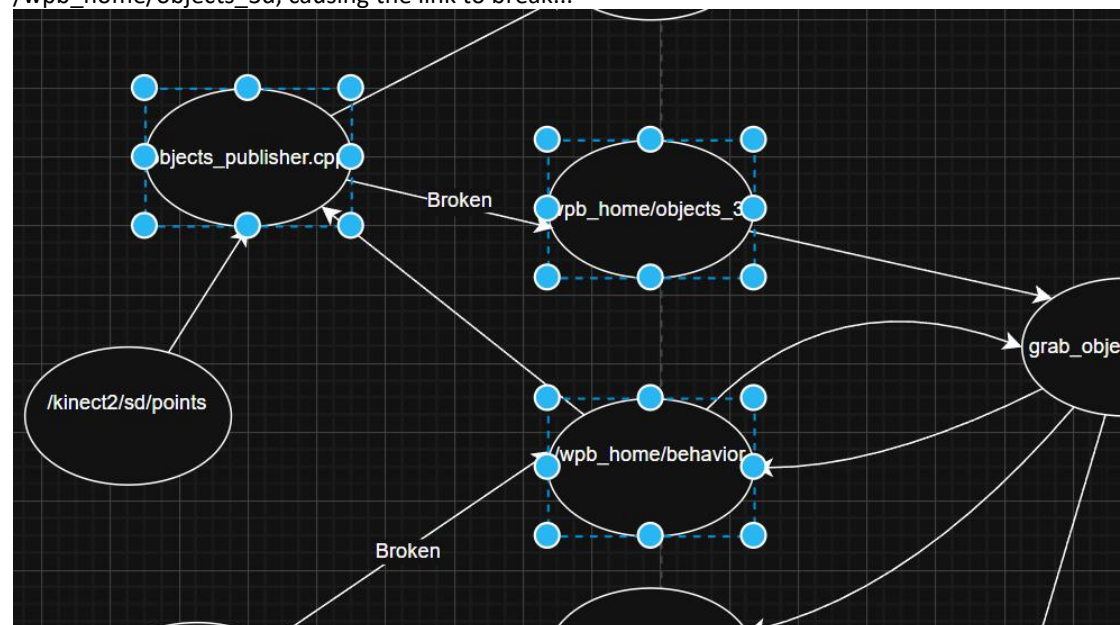
The `/wpb_home/behavior` node is linked to `objects_publisher.cpp` and `grab_object_sim.cpp`. `objects_publisher.cpp` handles the recognition of object point clouds.

```

OPEN EDITORS | 1 unsaved
ROS2_WS
src
  wpr_simulation2
    scripts
      src
        ball_random_move.cpp
        face_detector.py
        grab_object_sim.cpp
        keyboard_vel_cmd.cpp
        objects_publisher.cpp
        wpb_home_manip_sim.cpp
    worlds
      wpr_simulation2
    CMakeLists.txt
    package.xml
    README.md
    setup.cfg
    setup.py
    .data
    .posegraph
    bypass.py
    command
    waypoints.yaml
  OUTLINE
src > wpr_simulation2 > src > objects_publisher.cpp
61 private:
76 {
165 {
168 float points_y_sum = 0;
169 float points_z_sum = 0;
170 bool bFirstPoint = true;
171 for(int j = 0 ; j < point_num ; j++)
172 {
173   int point_index = cluster_indices[i].indices[j];
174   points_x_sum += cloud_src.points[point_index].x;
175   points_y_sum += cloud_src.points[point_index].y;
176   points_z_sum += cloud_src.points[point_index].z;
177 }
178
179 pcl::PointXYZRGB p = cloud_src.points[point_index];
180 if(bFirstPoint == true)
181 {
182   boxMarker.xMax = boxMarker.xMin = p.x;
183   boxMarker.yMax = boxMarker.yMin = p.y;
184   boxMarker.zMax = boxMarker.zMin = p.z;
185   bFirstPoint = false;
186 }
187
188 if(p.x < boxMarker.xMin) { boxMarker.xMin = p.x;}
189 if(p.x > boxMarker.xMax) { boxMarker.xMax = p.x;}
190 if(p.y < boxMarker.yMin) { boxMarker.yMin = p.y;}
191 if(p.y > boxMarker.yMax) { boxMarker.yMax = p.y;}
192 if(p.z < boxMarker.zMin) { boxMarker.zMin = p.z;}
193 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...



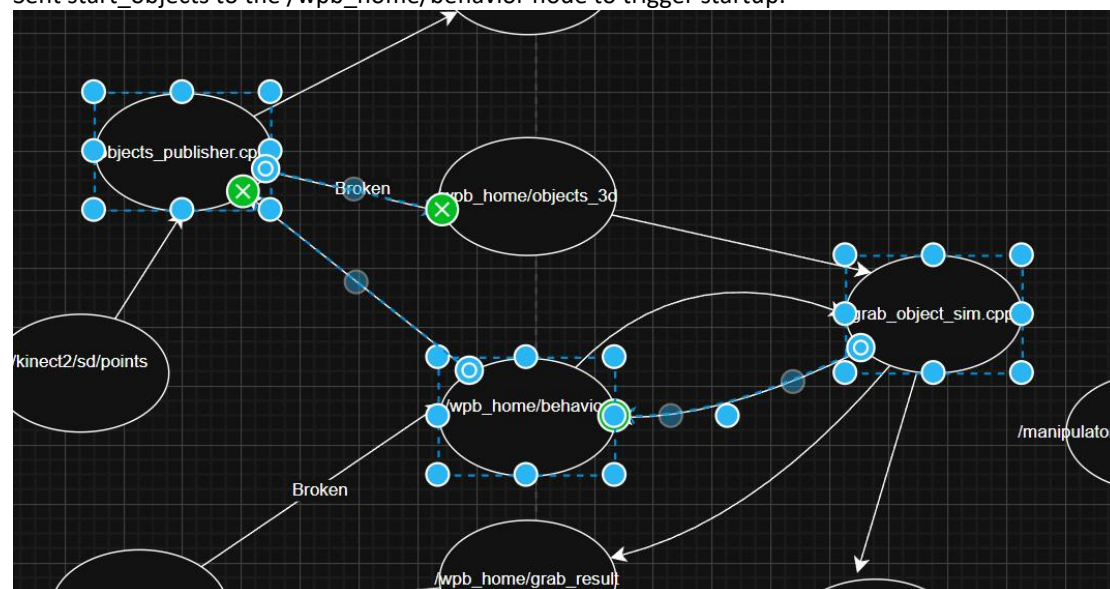
Follow through to grab_object_sim.cpp.


```

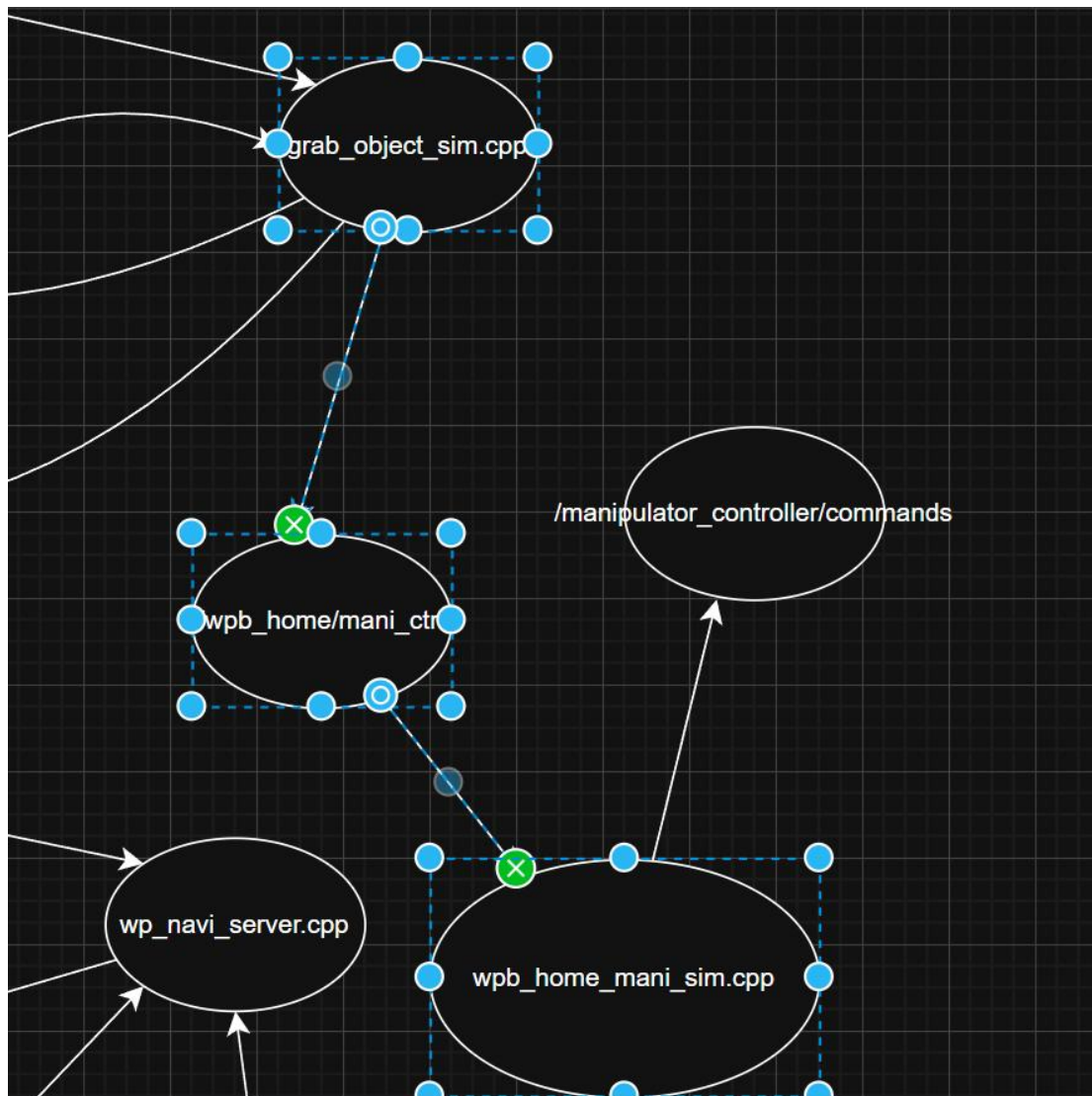
OPEN EDITORS 1 unsaved
ROS2_WS
src
  wpr_simulation2
    scripts
      src
        ball_random_move.cpp
        face_detector.py
        grab_object_sim.cpp
        keyboard_vel_cmd.cpp
        objects_publisher.cpp
        wpb_home_mani_sim.cpp
    worlds
      wpr_simulation2
        CMakeLists.txt
        package.xml
        README.md
        setup.cfg
        setup.py
        .data
        .posegraph
        bypass.py
        command
        waypoints.yaml
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::String::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 }

```

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...



Follow through to
wpb_home_manip_sim.cpp.

```

61  {
77  {
81  {
102  {
103  {
104  if (grab_step == STEP_HAND_UP)
105  {
106  RCLCPP_INFO(node->get_logger(), "[STEP_HAND_UP]");
107  sensor_msgs::msg::JointState mani_msg;
108  mani_msg.name.resize(2);
109  mani_msg.name[0] = "lift";
110  mani_msg.name[1] = "gripper";
111  mani_msg.position.resize(2);
112  mani_msg.position[0] = object_z;
113  mani_msg.position[1] = 0.15;
114  mani_pub->publish(mani_msg);
115  rclcpp::sleep_for(std::chrono::milliseconds(8000));
116  grab_step = STEP_FORWARD;
117  continue;
118  }
119  if (grab_step == STEP_FORWARD)
120  {
121  RCLCPP_INFO(node->get_logger(), "[STEP_FORWARD] object_x = %.2f", object_x);
122  geometry_msgs::msg::Twist vel_msg;
123  vel_msg.linear.x = 0.1;
124  vel_msg.linear.y = 0;
125  vel_pub->publish(vel_msg);
126  int forward_duration = (object_x - 0.65) * 20000;
127  rclcpp::sleep_for(std::chrono::milliseconds(forward_duration));
128  grab_step = STEP_GRAB;

```

Fortunately, we located the manipulator control logic...

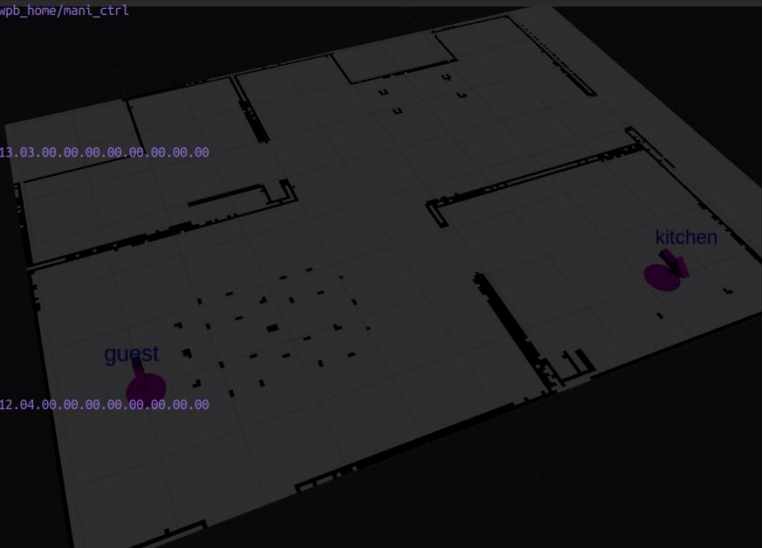
Now we inspect the node details.

\$ ros2 topic info -v /wpb_home/manip_ctrl

```

two@two-virtual-machine:~$ ros2 topic info -v /wpb_home/mani_ctrl
Type: sensor_msgs/msg/JointState
Publisher count: 1
Node name: grab_object_sim
Node namespace: /
Topic type: sensor_msgs/msg/JointState
Endpoint type: PUBLISHER
GID: 01.0f.f5.5f.e4.31.43.2b.00.00.00.00.13.03.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  History (Depth): UNKNOWN
  Durability: VOLATILE
  Lifespan: Infinite
  Deadline: Infinite
  Liveliness: AUTOMATIC
  Liveliness lease duration: Infinite
Subscription count: 1
Node name: wpb_home_manip_sim
Node namespace: /
Topic type: sensor_msgs/msg/JointState
Endpoint type: SUBSCRIPTION
GID: 01.0f.f5.5f.65.33.1a.38.00.00.00.00.12.04.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  History (Depth): UNKNOWN
  Durability: VOLATILE
  Lifespan: Infinite
  Deadline: Infinite
  Liveliness: AUTOMATIC
  Liveliness lease duration: Infinite

```



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).

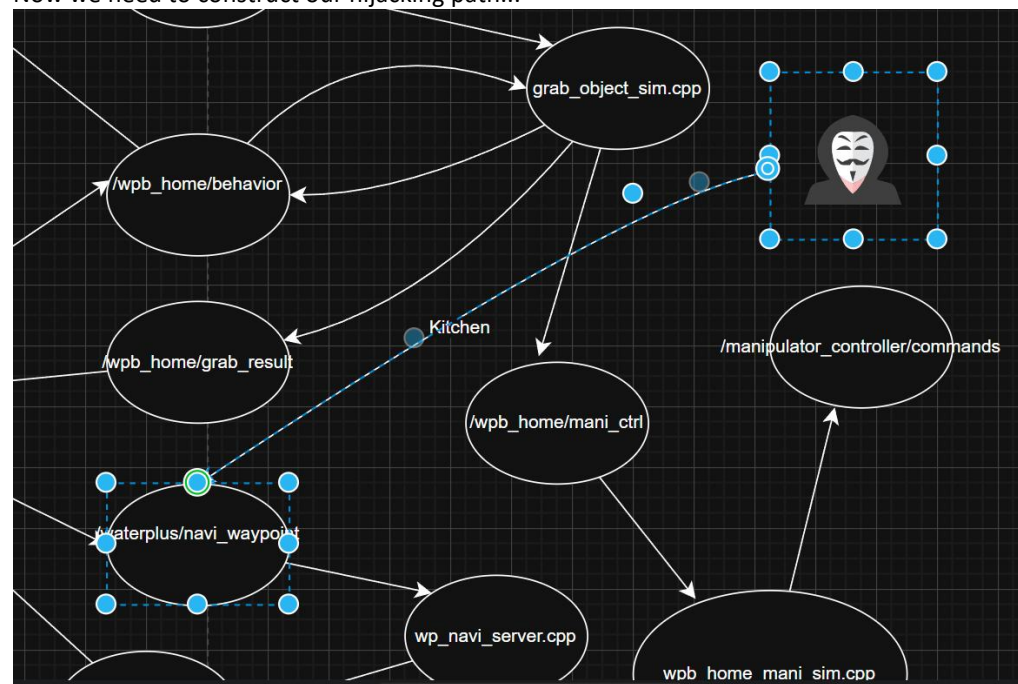
Can be left empty → use an empty array if not needed.

4. effort: float64[]

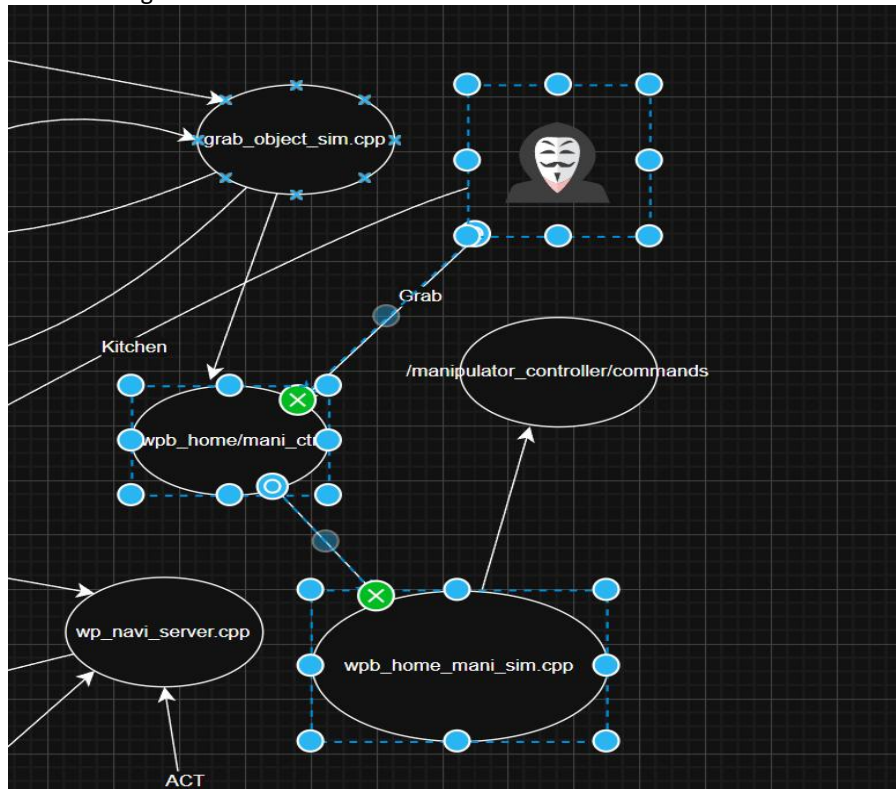
The torque or force of each joint (N·m or N).

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

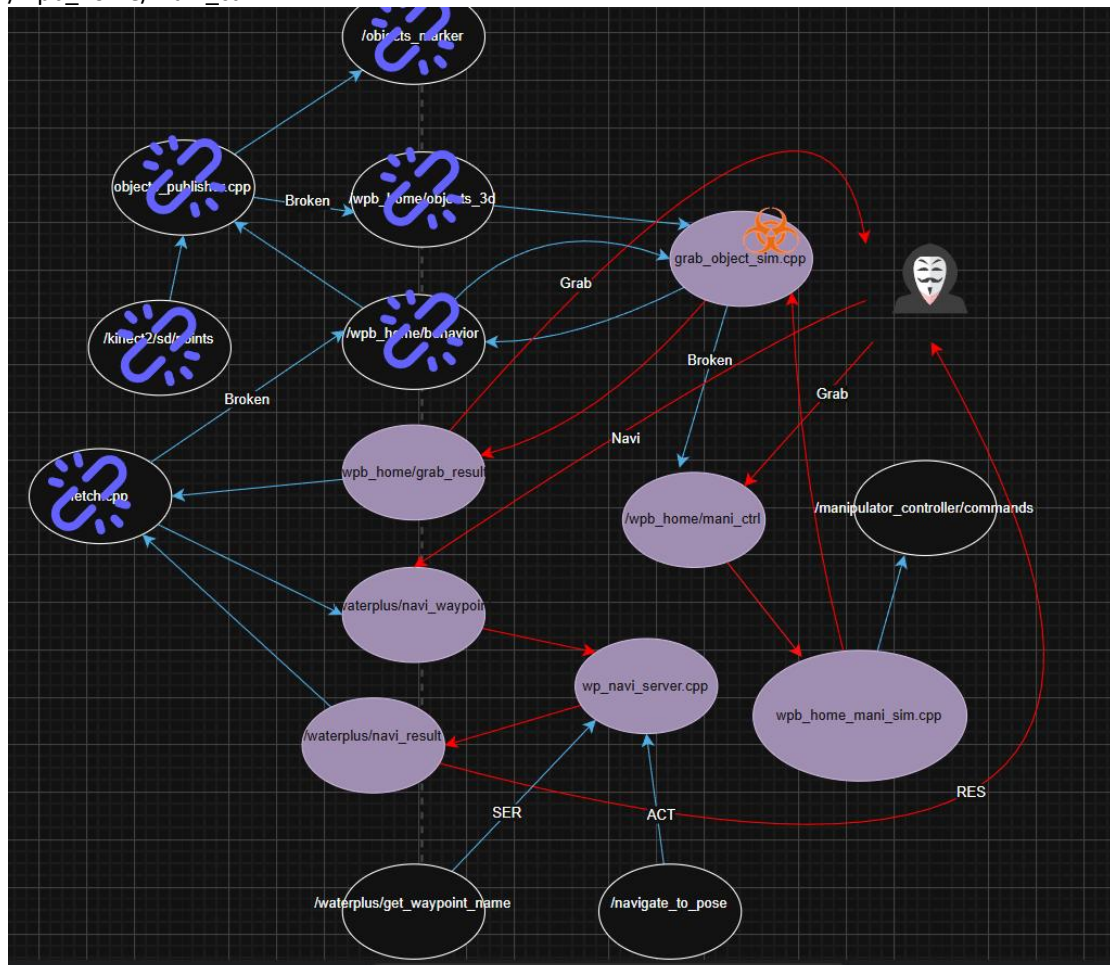
Now we need to construct our hijacking path...



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



Send control data directly to
/wpb_home/mani_ctrl.




```
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 ===')

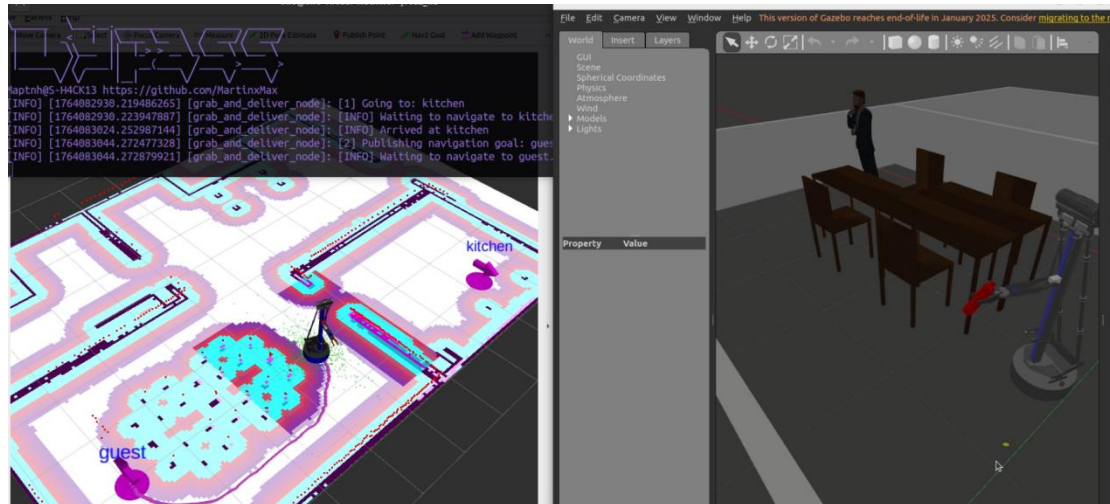
def main(args=None):
    print(LOGO)
    rclpy.init(args=args)
    node = GrabAndDeliver()
    node.run_sequence()
    node.destroy_node()
    rclpy.shutdown()

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...

