Introduction to Robot Operating System (ROS 1)

Dr. Essa Alghannam

RViz can visualize data:

1. Robot models (URDF/SDF): a robot model position and orientation can be visualized in RViz. need to launch the robot description and a `robot_state_publisher`.
2. Point clouds: Visualize 3D point clouds from sensors like lidar.
3. Laser scans: Visualize data from laser range finders.
4. Odometry: Show the estimated pose of the robot.
5. Camera images: Display images from cameras.

To visualize these DATA, we need to write ROS nodes that publish the appropriate ROS messages and configure RViz accordingly.

Commented [EA1]: URDF (Unified Robot Description Format) and SDF (Simulation Description Format) are both **XML-based file formats** used in robotics to describe robots and their environments.

URDF is used to describe the kinematic and dynamic properties of a **single robot** for use within the **ROS (Robot Operating System)**.

- **Kinematics:** Defines how joints connect links and the resulting degrees of freedom.
- **Dynamics:** Can include inertial properties (mass, inertia tensor) for basic physics calculations.

SDF: To describe **everything** in a **Gazebo simulation environment**, including robots, static objects, sensors, lights, and various physics properties.

First Exapmle: Running RViz and a node:

1. Open a terminal and run `roscore` to start the ROS master.
2. In another terminal, run `rqt_graph`
3. In another terminal, navigate to the workspace 'mycatkin_ws' and run `rviz`.

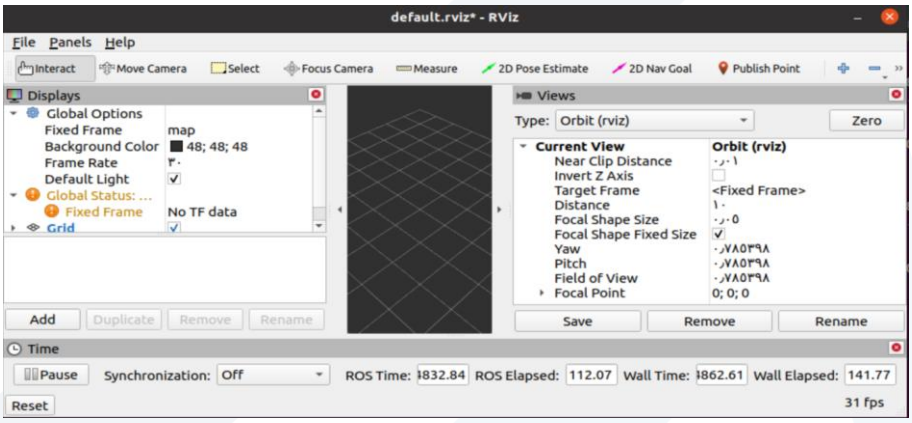
Output:

```
[ INFO] [1734184718.596195559]: rviz version 1.14.25
[ INFO] [1734184718.596349632]: compiled against Qt version 5.12.8
[ INFO] [1734184718.596412848]: compiled against OGRE version 1.9.0 (Ghadamon)
[ INFO] [1734184718.620233275]: Forcing OpenGl version 0.
```

[INFO] [1734184719.842820826]: Stereo is NOT SUPPORTED

[INFO] [1734184719.843075946]: OpenGL device: llvmpipe (LLVM 12.0.0, 256 bits)

[INFO] [1734184719.843144009]: OpenGl version: 3.1 (GLSL 1.4).



4. Navigate the **scr** folder of the **workspace 'mycatkin**_ws'. And run:

   essa@essa:~/mycatkin_ws$ catkin_create_pkg **rvizpkg** std_msgs rospy roscpp

5. go back to the workspace folder and run:

   essa@essa:~/mycatkin_ws$ catkin_make

6. Add the following code named "**markerpublish.py**" to the **src** folder of the package: ~/mycatkin_ws/src/**rvizpkg**/**src**

~/mycatkin_ws/src/rvizpkg/src/markerpublish.py

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

2  import rospy

3  from visualization_msgs.msg import Marker, MarkerArray
```

Commented [EA2]: ROS node that publishes a **red sphere** to the /visualization_marker topic, which RViz can then display.

Commented [EA3]: Shebang tells that Python 3 interpreter is used in the user's environment.

Commented [EA4]: This line imports the Marker and MarkerArray message types from the visualization_msgs package. These messages are used to define and publish visual markers for display in RViz.

```
4  def publish_marker():
5      rospy.init_node('rviz_marker_publisher', anonymous=True)
6      pub = rospy.Publisher('/visualization_marker', Marker, queue_size=10)
7      rate = rospy.Rate(10) # 10hz
8      marker = Marker()
9      marker.header.frame_id = "world" # Or robot's base frame
10     marker.header.stamp = rospy.Time.now()
11     marker.id = 0
12     marker.type = Marker.CUBE
13     marker.action = Marker.ADD
14     marker.pose.position.x = 0
15     marker.pose.position.y = 0
16     marker.pose.position.z = 0
17     marker.pose.orientation.x = 0.0
18     marker.pose.orientation.y = 0.7
19     marker.pose.orientation.z = 0.7
20
```

Commented [EA5]: `/visualization_marker` is a ROS topic for publishing markers visualized in RViz. It's common to display simple geometric shapes (spheres, cubes, cylinders, etc.), text, and other visual elements in a 3D environment.

Commented [EA6]: we can publish either `visualization_msgs/Marker` (single marker) using visualization_msgs/Marker or `visualization_msgs/MarkerArray` (multiple markers) using `visualization_msgs/MarkerArray`.

Commented [EA7]: This message contains all the information needed to render the marker in RViz: its position, orientation, shape, color, size, etc.

Commented [EA8]: frame_id: The coordinate frame that the marker is relative to ("world" in this example). This specifies the coordinate frame in which the marker's position and orientation are defined.

Commented [EA9]: action: Action to perform (Marker.ADD to create a new marker). Marker Action: The `action` field specifies whether to add, modify, or delete a marker.

```
21      marker.pose.orientation.w = 0.7
22      marker.scale.x = 0.5
23      marker.scale.y = 0.5
24      marker.scale.z = 0.5
25      marker.color.r = 1.0
26      marker.color.g = 0.0
27      marker.color.b = 0.0
        marker.color.a = 1.0


28      while not rospy.is_shutdown():
29          pub.publish(marker)
30          rate.sleep()


31  if __name__ == '__main__':
32      try:
33          publish_marker()
34      except rospy.ROSInterruptException:
35          pass
```

8 -----> 27 LINES

8. marker = Marker()

marker = Marker(): This line creates a **new Marker object**. This object will hold all the properties of the marker (sphere in this case) that will be visualized.

The following lines set the properties of the marker object:

9. marker.header.frame_id = "world" # Or robot's base frame

Frame ID: The `header.frame_id` field in the `Marker` message is crucial. This specifies the coordinate frame in which the marker's position and orientation are defined. It should match a frame known to RViz (e.g., `/map`, `/odom`, `/base_link`). Incorrect frame IDs will result in the marker not appearing correctly or at all.

10.marker.header.stamp = rospy.Time.now()

stamp: Timestamp.

Commented [EA10]: returns the current time according to the ROS clock.

- RViz uses the stamp field to determine if a piece of data is "fresh" enough to be displayed. If RViz receives a marker with a very old stamp, it might ignore it or display warnings because it considers the data stale.
- It ensures that different sensor readings or robot states displayed in RViz are synchronized to a common timeline.

11.marker.id = 0

id: Unique identifier for this marker. Marker ID: The `id` field is used to identify individual markers. If we're publishing multiple markers, each one should have a unique ID. This allows us to update or delete specific markers later.

12.marker.type = Marker.SPHERE

type: The shape of the marker (Marker.SPHERE). Marker Type: The `type` field determines the shape of the marker (sphere, cube, arrow, text, etc.).

13.marker.action = Marker.ADD

action: Action to perform (Marker.ADD to create a new marker). Marker Action: The `action` field specifies whether to add, modify, or delete a marker.

14.marker.pose.position.x = 0

15.marker.pose.position.y = 0

16.marker.pose.position.z = 0

17.marker.pose.orientation.x = 0.0

18.marker.pose.orientation.y = 0.0

19.marker.pose.orientation.z = 0.0

20.marker.pose.orientation.w = 1.0

pose: Position and orientation of the marker (set to origin in this example).

21.marker.scale.x = 0.5

22.marker.scale.y = 0.5

23.marker.scale.z = 0.5

scale: Size of the marker (0.5 x 0.5 x 0.5 meters).

For SPHERE:

- marker.scale.x: Defines the radius of the sphere along its local X-axis.
- marker.scale.y: Defines the radius of the sphere along its local Y-axis.
- marker.scale.z: Defines the radius of the sphere along its local Z-axis.

24.marker.color.r = 1.0

25.marker.color.g = 0.0

26.marker.color.b = 0.0

27.marker.color.a = 1.0

color: Color of the marker (red in this example: r=1.0, g=0.0, b=0.0, a=1.0 for full opacity). Four floating-point values:

- `r`: Red component (0.0 to 1.0)
- `g`: Green component (0.0 to 1.0)
- `b`: Blue component (0.0 to 1.0)
- `a`: Alpha (transparency) component (0.0 to 1.0)

These values represent the intensity of each color channel, where 0.0 means no intensity (absence of that color) and 1.0 means full intensity. The alpha value controls transparency:

- `a = 0.0`: Fully transparent (invisible)
- `a = 1.0`: Fully opaque (completely visible)
- `0.0 < a < 1.0`: Semi-transparent (partially visible)
- `marker.color.r = 0.5; marker.color.g = 0.5; marker.color.b = 0.5; marker.color.a = 0.5;` would create a semi-transparent gray marker.

NOTES

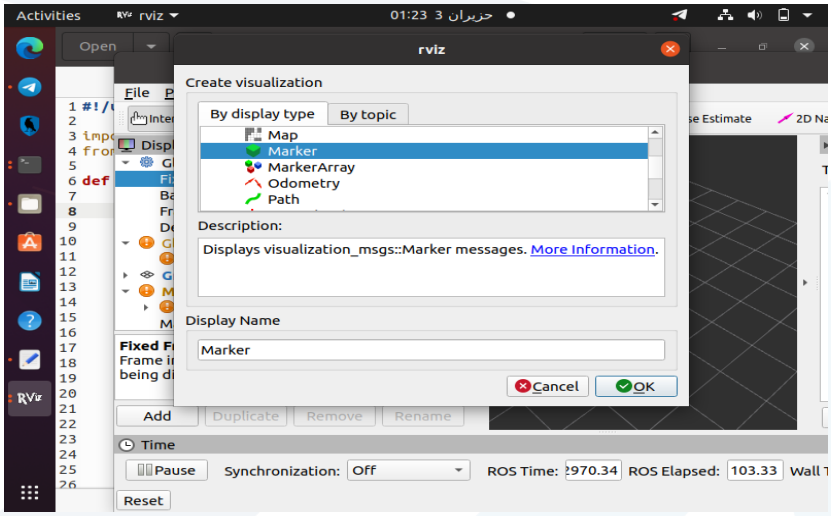
- Navigate the src folder of the package ~/mycatkin_ws/src/rvizpkg/src and to make the code executable:

`chmod +x markerpublish.py`

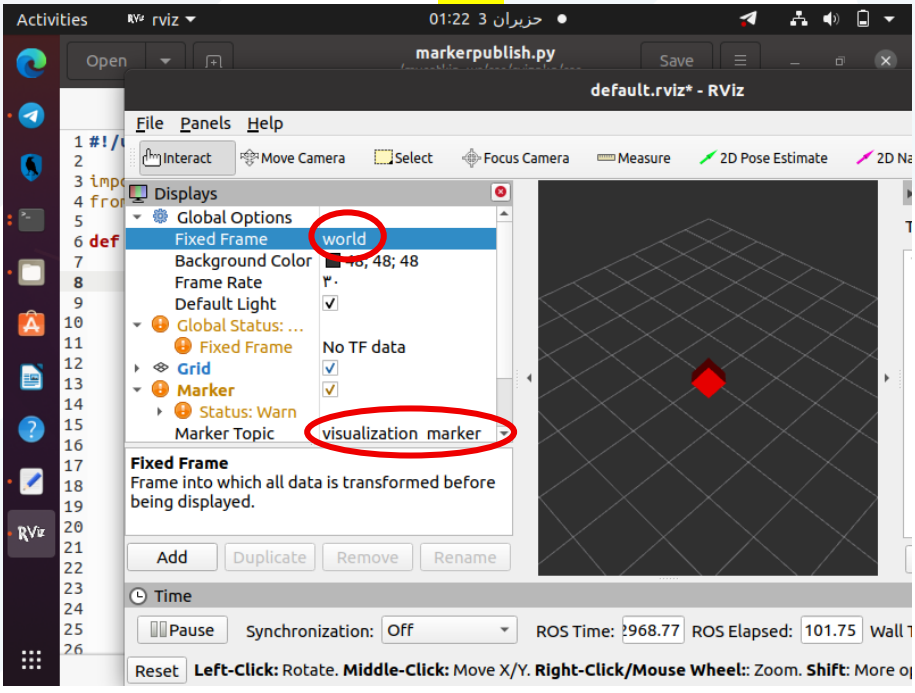
- Edit the package `CMakeLists.txt` by modifing the lines 162-164:

```
catkin_install_python(PROGRAMS
  src/markerpublish.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

- Run the node: In a third terminal, run: `**rosrun rvizpkg markerpublish.py** `
- In the RViz window, go to "Add" -> "By topic" and select `/visualization_marker`. RViz will then subscribe to this topic and display the published markers.



- Modify fixed frame name "map" to "world" **according to line 9 in the code**:



1.1 Saving RViz configurations

- From the "File" Menu in the RViz window, we select "Save Config As..." or Save Config to update a preloaded file.
- Choose a Location and Name: Save the .rviz file inside the ROS package, typically in a dedicated config folder:
  (e.g., **~/mycatkin_ws/src/rvizpkg/config/markers_view.rviz**). This makes it easy to manage and use in launch files later.
- Then we click Save.

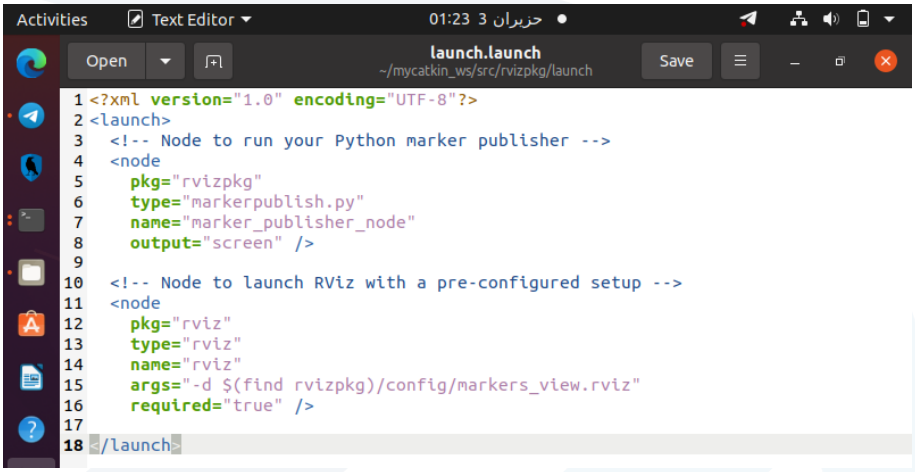
1.2 ADD Launch file:

~/mycatkin_ws/src/rvizpkg/launch/launch.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
        <!-- Node to run Python marker publisher -->
        <node
            pkg="rvizpkg"
            type="markerpublish.py"
            name="marker_publisher_node"
            output="screen" />
        <!-- Node to launch RViz with a pre-configured setup -->
        <node
            pkg="rviz"
            type="rviz"
            name="rviz"
            args="-d $(find rvizpkg)/config/markers_view.rviz"
          required="true" />
</launch>
```

roslaunch rvizpkg launch.launch

Commented [EA11]: XML Declaration
This is the standard and often mandatory first line of any XML document.



In RViz, `Marker`, and `MarkerArray` are both used to **visualize data** in **3D space**, but they **differ** in how they **handle multiple visualizations**:

Feature	**Marker**	**MarkerArray**
Number of Objects	Single	Multiple
Efficiency for many objects	Inefficient	Efficient
Message Size	Smaller **cube**, **sphere**, **arrow**, **text**, **line**, **etc**. **a robot**, **a single obstacle**, or a **specific point of interest**	Larger (potentially) **point clouds**, **trajectories**, or **collections of objects.** **lidar sensor**, **plotting a robot's path** over time, or **showing the positions of multiple detected objects**
Use Case	Single object visualization	Many objects visualization

The choice between using `**Marker**` and `**MarkerArray**` depends on the nature of the data we want to visualize:

- If we have a **few independent objects**, using individual `**Marker**` messages is simpler.
- If we have many similar objects, such as points in a cloud or waypoints in a path, using `**MarkerArray**` is much more efficient and cleaner. We can manage the visualization more effectively by publishing a single updated message rather than multiple individual messages.

Add a new python file to the scr folder of rvizpkg

~/mycatkin_ws/src/rvizpkg/src/publish_marker_array.py

```
#!/usr/bin/env python

import rospy
from visualization_msgs.msg import Marker, MarkerArray
from geometry_msgs.msg import Point
import math

def publish_marker_array():
    rospy.init_node('marker_array_publisher', anonymous=True)

    # Create a publisher for MarkerArray messages
    marker_array_pub = rospy.Publisher('/my_marker_array', MarkerArray,
queue_size=10)

    rate = rospy.Rate(2) # Publish at 2 Hz (2 times per second)

    rospy.loginfo("Starting MarkerArray publisher...")

    while not rospy.is_shutdown():
        marker_array = MarkerArray() # Create a new MarkerArray message
```

Commented [EA12]: # The topic name is crucial: /my_marker_array (we will use this in RViz)

```
# --- Marker 1: A moving cube ---
marker1 = Marker()
marker1.header.frame_id = "world"
marker1.header.stamp = rospy.Time.now()
marker1.ns = "moving_cube" # Namespace for this marker
marker1.id = 0          # Unique ID within "moving_cube" namespace
marker1.type = Marker.CUBE # Shape of the marker
marker1.action = Marker.ADD # Add or modify the marker

# Animate its position
time_elapsed = rospy.Time.now().to_sec()
marker1.pose.position.x = math.sin(time_elapsed) * 2.0
marker1.pose.position.y = math.cos(time_elapsed) * 2.0
marker1.pose.position.z = 0.5
marker1.pose.orientation.w = 1.0 # No rotation

marker1.scale.x = 0.3 # Size of the cube
marker1.scale.y = 0.3
marker1.scale.z = 0.3

marker1.color.a = 1.0 # Alpha (transparency)
marker1.color.r = 1.0 # Red
marker1.color.g = 0.0 # Green
marker1.color.b = 0.0 # Blue
```

```
marker_array.markers.append(marker1)

# --- Marker 2: A simple grid of spheres ---
# Using a loop to create multiple markers efficiently
for i in range(-5, 6): # -5 to 5
  for j in range(-5, 6): # -5 to 5
    marker2 = Marker()
    marker2.header.frame_id = "world"
    marker2.header.stamp = rospy.Time.now()
    marker2.ns = "grid_spheres" # Different namespace
    # Unique ID for each sphere in the grid
    marker2.id = (i + 5) * 11 + (j + 5) # (0 to 10) * 11 + (0 to 10)
    marker2.type = Marker.SPHERE
    marker2.action = Marker.ADD

    marker2.pose.position.x = float(i) * 1.0
    marker2.pose.position.y = float(j) * 1.0
    marker2.pose.position.z = 0.0
    marker2.pose.orientation.w = 1.0 # No rotation

    marker2.scale.x = 0.1 # Size of the sphere
    marker2.scale.y = 0.1
    marker2.scale.z = 0.1
```

```
        marker2.color.a = 0.7 # Slightly transparent
        marker2.color.r = 0.0
        marker2.color.g = 0.8
        marker2.color.b = 0.0



        marker_array.markers.append(marker2)

    # --- Marker 3: Line list (efficient for drawing many lines) ---
    marker3 = Marker()
    marker3.header.frame_id = "world"
    marker3.header.stamp = rospy.Time.now()
    marker3.ns = "line_square"
    marker3.id = 0 # Unique ID
    marker3.type = Marker.LINE_LIST # Important: LINE_LIST type
    marker3.action = Marker.ADD

    marker3.scale.x = 0.05 # Line width (only x component is used for line
types)

    marker3.color.a = 1.0
    marker3.color.r = 0.0
    marker3.color.g = 0.0
    marker3.color.b = 1.0
```

```
# Define points for lines (each pair of points forms a line)
marker3.points.append(Point(1.0, 1.0, 0.0))
marker3.points.append(Point(1.0, 2.0, 0.0))

marker3.points.append(Point(1.0, 2.0, 0.0))
marker3.points.append(Point(2.0, 2.0, 0.0))

marker3.points.append(Point(2.0, 2.0, 0.0))
marker3.points.append(Point(2.0, 1.0, 0.0))

marker3.points.append(Point(2.0, 1.0, 0.0))
marker3.points.append(Point(1.0, 1.0, 0.0))

marker_array.markers.append(marker3)



# --- Marker 4: center sphere ---
marker4 = Marker()
marker4.header.frame_id = "world"
marker4.header.stamp = rospy.Time.now()
marker4.ns = "center" # Namespace for this marker
```

```
marker4.id = 0          # Unique ID within "moving_cube" namespace
marker4.type = Marker.SPHERE # Shape of the marker
marker4.action = Marker.ADD # Add or modify the marker

marker4.pose.position.x = 0.0
marker4.pose.position.y = 0.0
marker4.pose.position.z = 0.0
marker4.pose.orientation.w = 1.0 # No rotation

marker4.scale.x = 0.2 # Size of the cube
marker4.scale.y = 0.2
marker4.scale.z = 0.2

marker4.color.a = 1.0 # Alpha (transparency)
marker4.color.r = 1.0 # Red
marker4.color.g = 1.0 # Green
marker4.color.b = 0.0 # Blue

marker_array.markers.append(marker4)



# Publish the MarkerArray
marker_array_pub.publish(marker_array)
```

```
        rospy.loginfo("Published MarkerArray with %d markers",
len(marker_array.markers))

        rate.sleep()

if __name__ == '__main__':
    try:
        publish_marker_array()
    except rospy.ROSInterruptException:
        pass
```

- In the RViz window, go to "Add" -> "By topic" and select `/my_marker_array`. RViz will then subscribe to this topic and display the published markers.
- Modify fixed frame name "map" to "world" **according to the code**:

Saving RViz configurations

- From the "File" Menu in the RViz window, we select "Save Config As..." or Save Config to update a preloaded file.
- Choose a Location and Name: Save the .rviz file inside the ROS package, typically in a dedicated config folder:
  (e.g., **~/mycatkin_ws/src/rvizpkg/config/markers_view2.rviz**). This makes it easy to manage and use in launch files later.
- Then we click Save.

Add a new launch file:

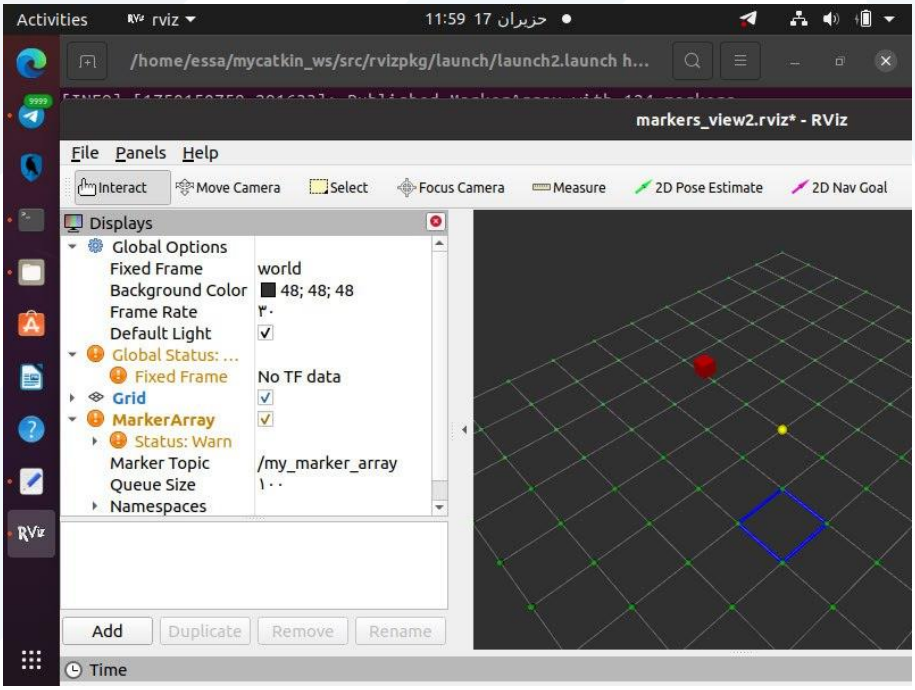
~/mycatkin_ws/src/rvizpkg/launch/launch2.launch

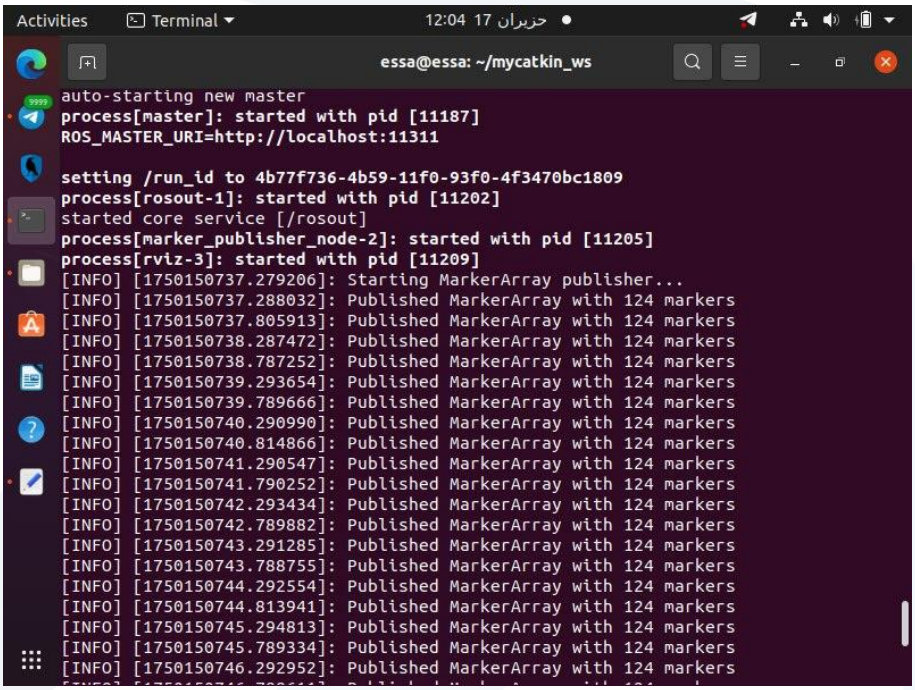
```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <!-- Node to run the Python marker publisher -->
  <node
    pkg="rvizpkg"
    type="publish_marker_array.py"
    name="marker_publisher_node"
    output="screen" />

  <!-- Node to launch RViz with a pre-configured setup -->
  <node
    pkg="rviz"
    type="rviz"
    name="rviz"
    args="-d $(find rvizpkg)/config/markers_view2.rviz"
    required="true" />

</launch>
```

```
roslaunch rvizpkg launch2.launch
```




Activities
Terminal
12:04 17 حزيران
essa@essa: ~/mycatkin_ws
auto-starting new master
process[master]: started with pid [11187]
ROS_MASTER_URI=http://localhost:11311
setting /run_id to 4b77f736-4b59-11f0-93f0-4f3470bc1809
process[rosout-1]: started with pid [11202]
started core service [/rosout]
process[marker_publisher_node-2]: started with pid [11205]
process[rviz-3]: started with pid [11209]
[INFO] [1750150737.279206]: Starting MarkerArray publisher...
[INFO] [1750150737.288032]: Published MarkerArray with 124 markers
[INFO] [1750150737.805913]: Published MarkerArray with 124 markers
[INFO] [1750150738.287472]: Published MarkerArray with 124 markers
[INFO] [1750150738.787252]: Published MarkerArray with 124 markers
[INFO] [1750150739.293654]: Published MarkerArray with 124 markers
[INFO] [1750150739.789666]: Published MarkerArray with 124 markers
[INFO] [1750150740.290990]: Published MarkerArray with 124 markers
[INFO] [1750150740.814866]: Published MarkerArray with 124 markers
[INFO] [1750150741.290547]: Published MarkerArray with 124 markers
[INFO] [1750150741.790252]: Published MarkerArray with 124 markers
[INFO] [1750150742.293434]: Published MarkerArray with 124 markers
[INFO] [1750150742.789882]: Published MarkerArray with 124 markers
[INFO] [1750150743.291285]: Published MarkerArray with 124 markers
[INFO] [1750150743.788755]: Published MarkerArray with 124 markers
[INFO] [1750150744.292554]: Published MarkerArray with 124 markers
[INFO] [1750150744.813941]: Published MarkerArray with 124 markers
[INFO] [1750150745.294813]: Published MarkerArray with 124 markers
[INFO] [1750150745.789334]: Published MarkerArray with 124 markers
[INFO] [1750150746.292952]: Published MarkerArray with 124 markers