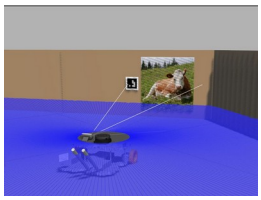
Robot Art Gallery

Project Report



ARAMS – Advanced Robotics and Autonomous Mobile Systems

By:

Bipeen Dhakal (3282119)
Rushikesh Pingat (3300656)

Prof. Dr. Ing. Stephan Kallweit
Mr. Patrick Wiesen

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Functionality:

The main purpose of this project is to learn how to navigate autonomously inside a certain unknown environment using SLAM toolbox. The additional goal is to find certain objects or pieces of art hanging in the walls of that environment using image processing tools like OpenCV. The main project is subdivided into following topics according to their functions and goals.

1. Spawn and Localization:

- After launching the world, the Robot localizes in the world by the help of the map generated and saved through SLAM toolbox.

    -ros2 launch tb3_gazebo arams.launch.py

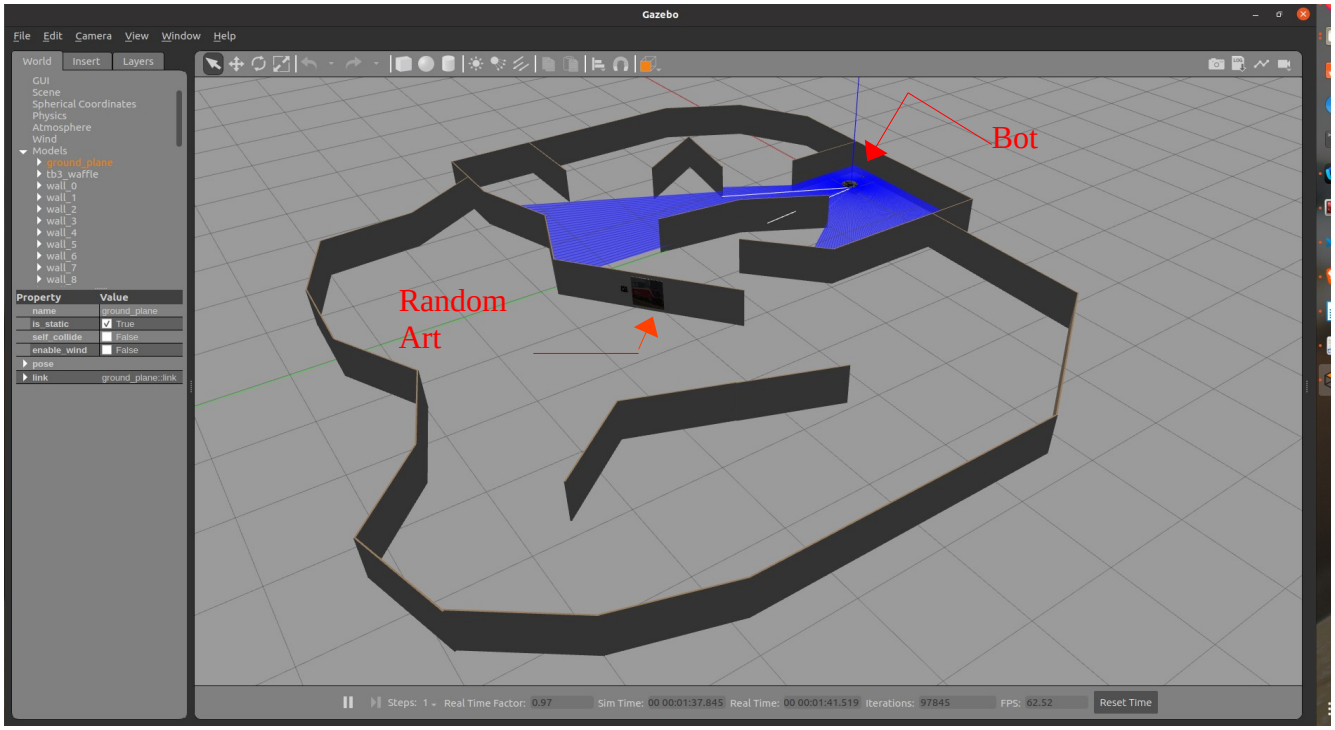


Figure 1: Art Gallery Arena

- We are using Slam toolbox and AMCL localization to map the environment, localize the bot itself and navigating through environment autonomously as per our requirement.

- The initial step is to run the simulation world and slam toolbox and control the bot with teleop for initial mapping process.

- After generating the whole occupancy grid of environment save the map files in *.pgm* and *.yaml* file types.

- for localization and navigation we load this map and robot is able to localize itself.

    -ros2 launch my_robot_slam clean_initiate.launch.py
    -ros2 launch my_robot_slam clean_begin.launch.py

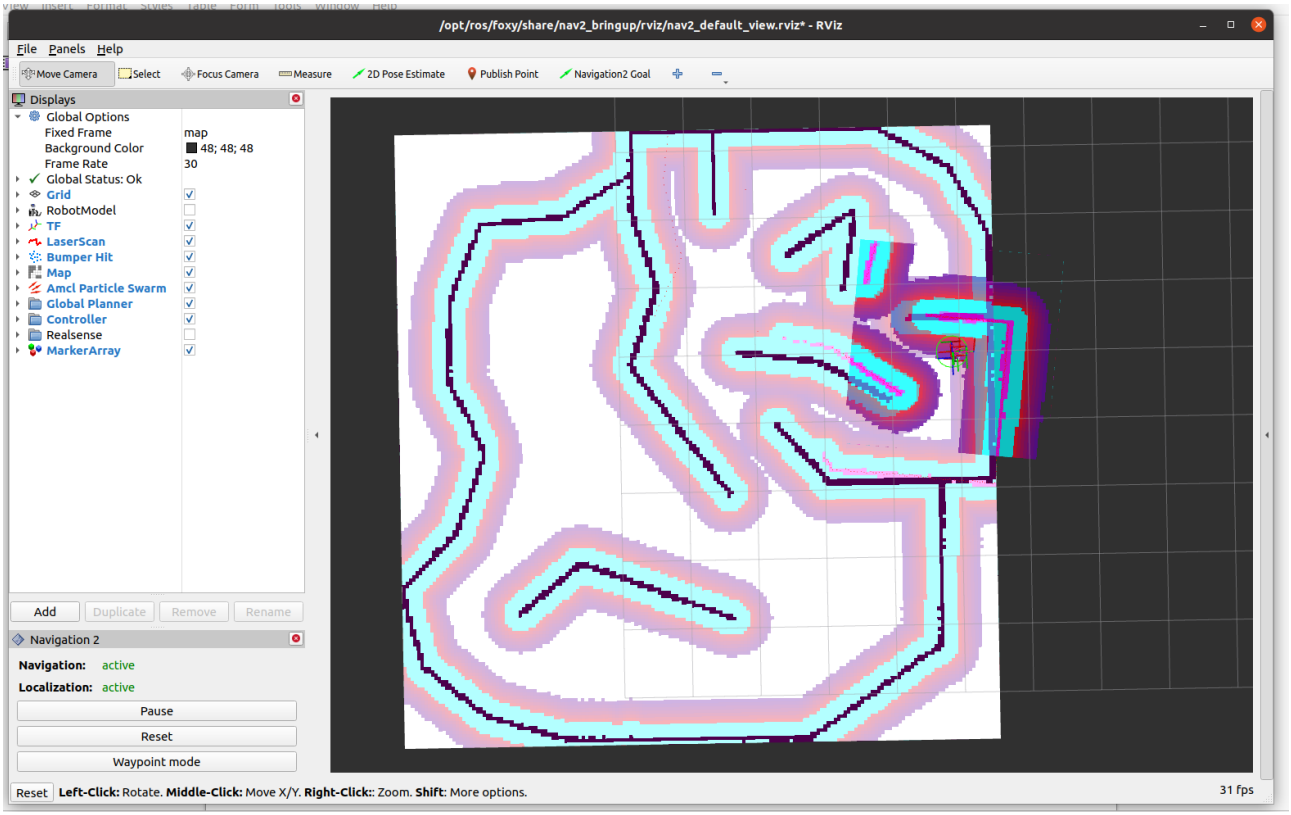


Figure 2: Arena map after Localization

2. Autonomous Navigation:

- With the *auto_explorer* node we make a bot go to various locations in the map where all the faces of walls can be visible and Images and Tags can be scanned and detected.
  - ros2 run my_robot_slam auto_explorer

- Here we publish to *cmd_vel* topic of turtlebot with linear and angular position and velocities through Twist() msg types.

```
  goal.pose.header.stamp = auto_chaos.get_clock().now().to_msg()
  if not Prev_position == position :
      goal.pose.pose.position = position
      #goal.pose.pose.orientation = orientation
      print("Received new goal => X: " + str(goal.pose.pose.position.x) + " Y: " + str(goal.pose.pose.position.y))
      send_goal_future = nav_to_pose_client.send_goal_async(goal)
      rclpy.spin_until_future_complete(auto_chaos, send_goal_future)
      goal_handle = send_goal_future.result()
      Prev_position = position


  if not goal_handle.accepted:
    print("Goal was rejected")
    nav_to_pose_client.destroy()
    auto_chaos.destroy_node()
    rclpy.shutdown()
    sys.exit(0)

  print("Goal Accepted!")

  return goal_handle

def checkResult(goal_handle):
  get_result_future = goal_handle.get_result_async()
  rclpy.spin_until_future_complete(auto_chaos, get_result_future)
  status = get_result_future.result().status
  if status == GoalStatus.STATUS_SUCCEEDED:
    print("Reached Goal !")
  return status
```

```
def generatePosition1():
  position = Point()
  position.x = 0.5
  position.y = 1.5
  position.z = 0.0
  return position
def generatePosition2():
  position = Point()
  position.x = 1.5
  position.y = 1.0
  position.z = 0.0
  return position
def generatePosition3():
  position = Point()
  position.x = 2.5
  position.y = 2.5
  position.z = 0.0
  return position
def generatePosition4():
  position = Point()
  position.x = 1.5
  position.y = 4.0
  position.z = 0.0
  return position
def generatePosition5():
  position = Point()
  position.x = -0.5
  position.y = 3.5
  position.z = 0.0
  return position
def generatePosition6():
  position = Point()
  position.x = -1.0
  position.y = 2.0
  position.z = 0.0
  return position
def generatePosition7():
  position = Point()
  position.x = -3.0
  position.y = 3.0
  position.z = 0.0
  return position
def generatePosition8():
```

Figure 3: WayPoints and Goals for auto exploration

- We are making the bot yaw at all those specific points in the map to detect the tags and images using if else statement, counter and time sleep tools of python as API.

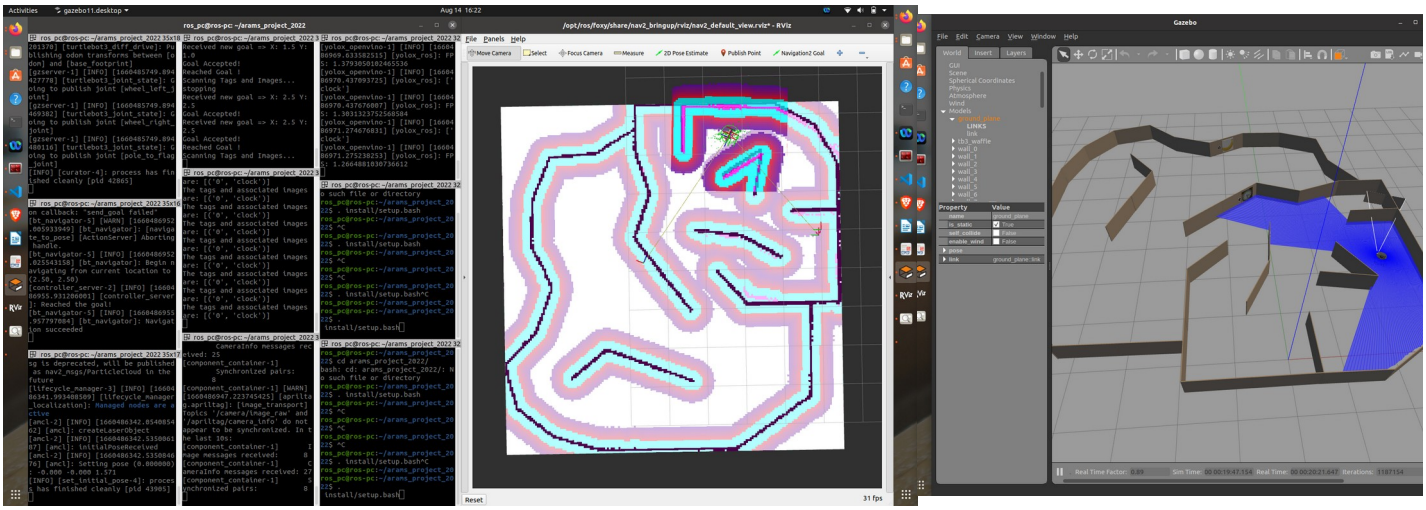


Figure 4: Autonomous Navigation with visualization in RVIZ2

3. Tag Detector:

- While exploring our bot can detect the AprilTags using tag_16h5_all.launch.py file in which we are subscribing to the */detections* topic of *aprilTag\msgs.*

  - ros2 launch apriltag-ros tag_16h5_all.launch.py

- The *follower.py* node takes the ID from topic and prints it into the terminal as an array.

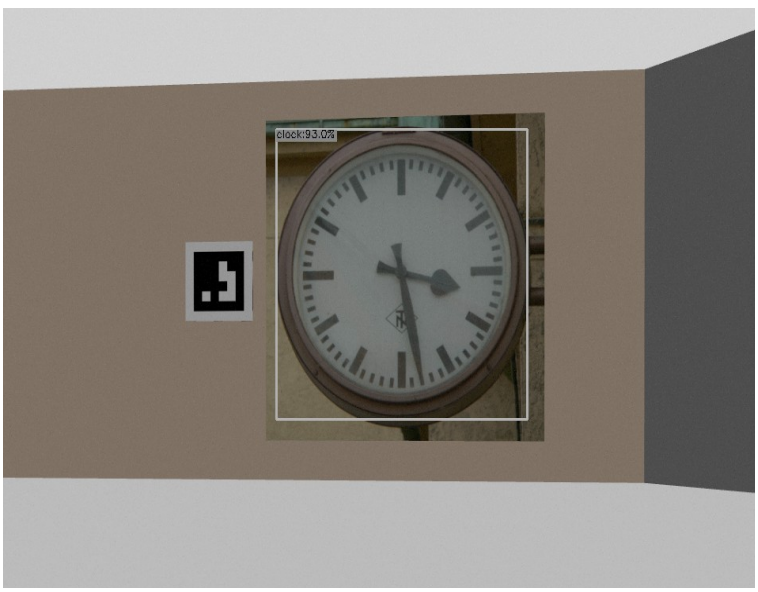
4. Object Detection:

- We are using YOLOX which is an anchor-free version of YOLO, with a simpler design but better performance.

- There are different sizes of YOLOX depending upon the user requirement ranging from nano to Larger versions. Inside it, we tested different versions and found M version best to our usage scenario. In the M version too, we used *Openvino* type because of its smooth usage of CPU and better performance.

  - ros2 launch yolox_ros_py yolox_m_openvino.launch.py

- It publishes the topic called bounding_boxes: Output BoundingBoxes like darknet_ros_msgs
  (bboxes_ex_msgs/BoundingBoxes)



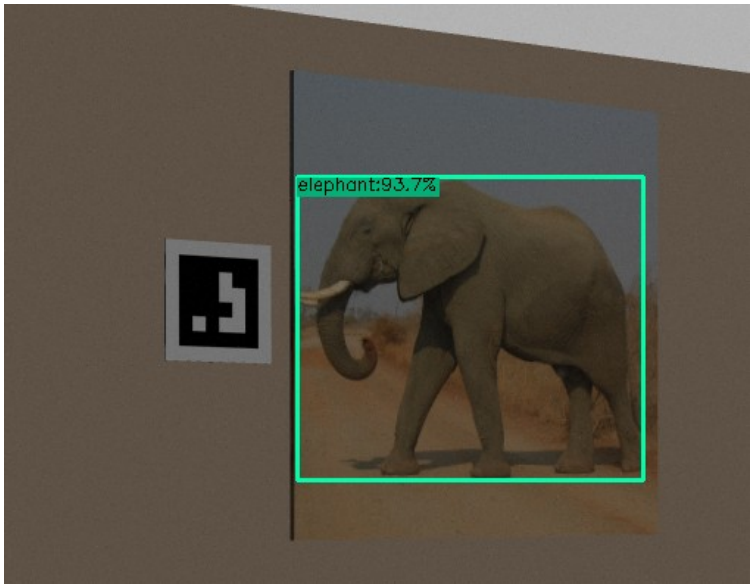


Figure 5: Object class detected with YOLOX and the confidence level in %

5. Final Results:

- At the end, we get detected tags and images inside an array printed in terminal window and as a text file *OutputR.txt* at root of the workspace next to *curator_debug.txt*.

- Side Note: occasionally, YOLOX may detect the wall at certain angle and proximity as a stop sign(as an anomaly). Apart then this glitch, the performance of YOLOX is highly accurate and satisfactory.