May 23, 2024 Intel Unnati Technical Challenge

Creating 2D Occupancy Grid Map using overhead infrastructure cameras

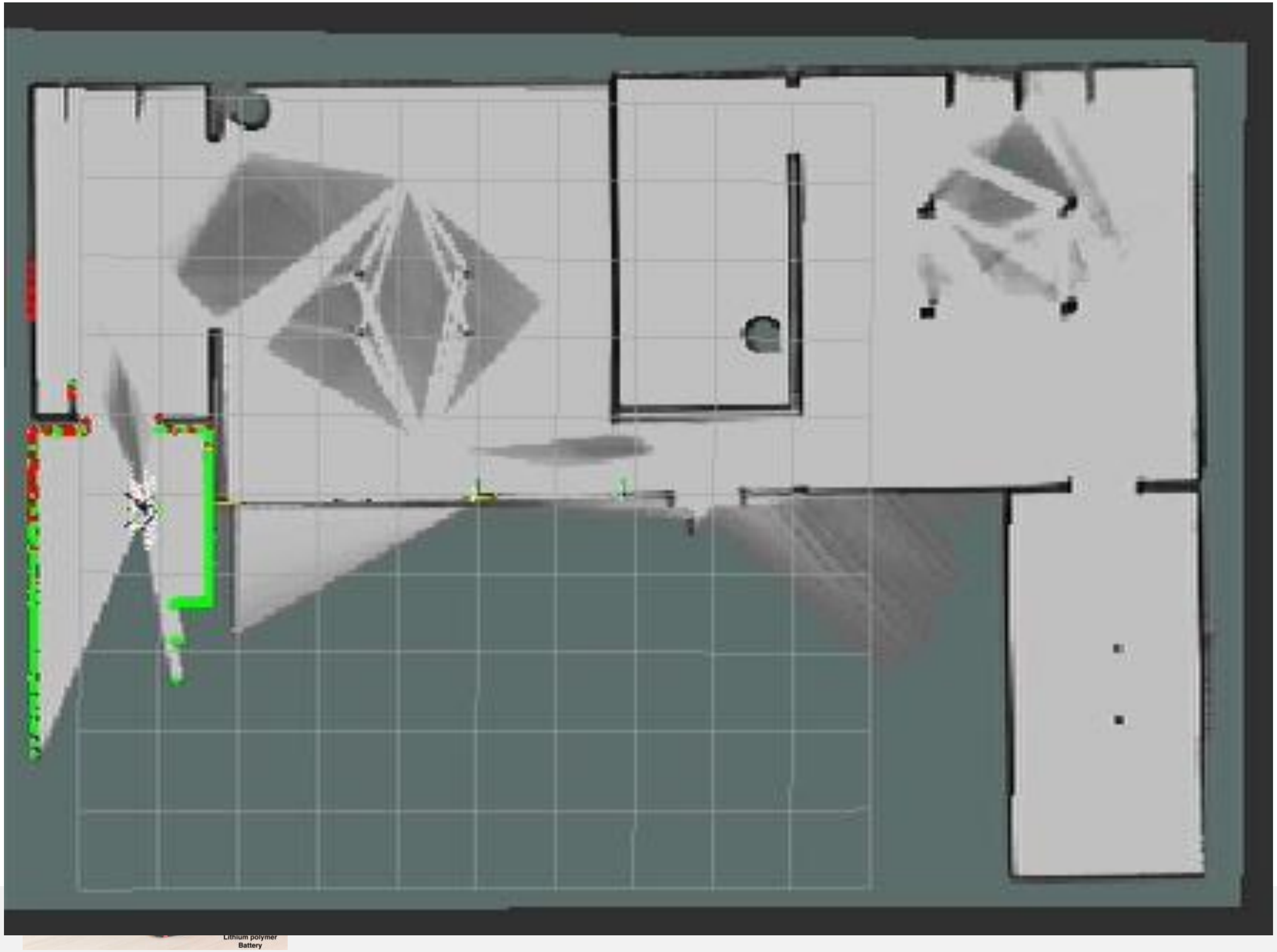
Arvind Raju, Amit Baxi, Ajay Kumar Sandula

raju.arvind@intel.com
amit.s.baxi@intel.com
ajay.kumar.sandula@intel.com

Agenda

- Problem, Concept and Motivation
- Project scope and expected deliverables
- Intro to ROS & Gazebo
- Primer on
  - How to set up simulation environment?
  - How to acquire data from simulated cameras?
  - How to measure the map accuracy?
- Questions

Problem: Environment Mapping for AMR navigation



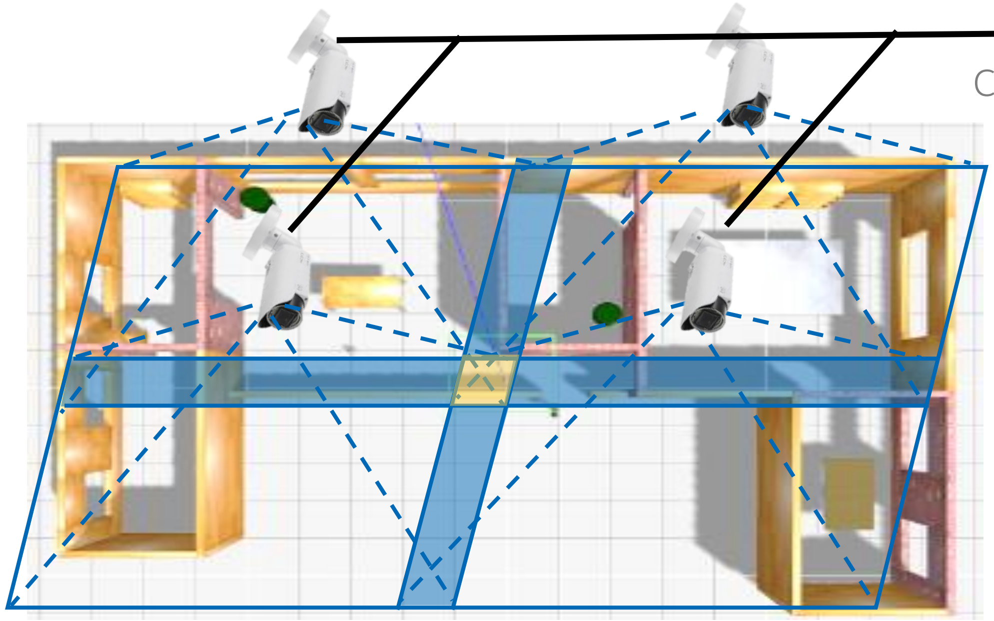
- Autonomous Mobile robots (AMRs) use on-board sensors to map the environment using SLAM
- Map is then used for path-planning, obstacle avoidance and navigation
- <u>Limitations of SLAM mapping:</u>
  - **Limited FoV:** can only map area in front of AMR – cannot map entire environment in one shot, in real-time
  - **Dynamic changes or obstacles not tracked** until they come in view of the AMR
  - Each AMR maps separately – no common map

SLAM – Simultaneous localization and mapping

Concept & Motivation: Mapping with infrastructure cameras

Demonstrate (in simulation) accurate and fast mapping (2D occupancy grid map) of indoor environments for AMR navigation using a network of RGB cameras in the infrastructure.

Concept

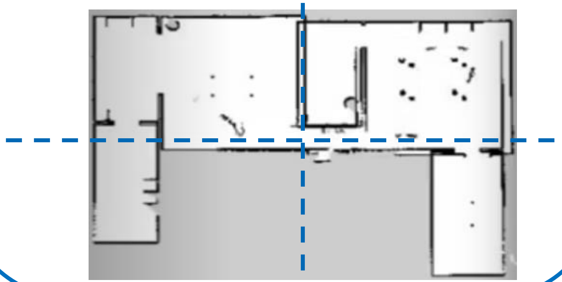


Camera Images

Cameras arranged in a (2x2) matrix

Edge Server

- Multi-camera data acquisition
- Depth estimation
- Individual FoV map generation
- Fusion of maps from multiple cams
- Composite Environment map



Black pixels – occupied / obstacles
White – empty space for navigation
Dark Grey – unexplored space

Motivation:

- Map entire environment in one shot
- Map moving obstacles in real-time
- Reduce AMR cost – don't need expensive LiDAR or depth cams
- Provide non line of sight to AMRs
- Better multi-robot path planning and coordination

Project Scope & Expected deliverables

- Project Scope
  - **Add the 4 RGB cameras in Gazebo simulation environment** [link] in a **2x2 matrix** (birds eye view) covering the environment with marginal overlap of FoVs at a height of **~ 8 meters** (at specified locations as on slide no.9)
  - Use **640x480 image resolution** and acquire images from the simulated cameras
  - Explore techniques for **multi-camera calibration** to automatically calibrate and align FoVs or all overhead cameras
  - Create a composite 2D occupancy grid map with accurate physical dimensions by fusing images from 4 cameras. **The map should be in a form (.pgm and 512x512 pixel resolution) such that it can be used in ROS navigation stack.**
  - Benchmark accuracy/error of infra-cam derived (dimensionally accurate) composite map v/s the Gazebo environment (refer slides 12-14) and Measure the computing latency to run the algorithm.
- Expected deliverables
  - **Word document describing your solution**, approach, novelty, pros/cons/limitations of your approach and comparison to state-of –the-art. Detailed block diagram and writeup describing your algorithm and approach.
  - **Fused map of the environment with detailed dimensions** (derived via your infra-cam fusion algorithm) v/s the ground truth map from Gazebo (provided)
  - **Error estimates:** Measure positions/distances between various key points in the composite map and identify % absolute avg, min and max errors. Provide details on how the algorithm was rigorously tested and the map accuracy results.
  - **Computing latency:** Characterize computing latency of the algorithm on an Intel i5 (10^{th} Gen) computer. Provide measurement procedure and detailed test results.
  - **Source code and algos to be shared** for evaluation.

Evaluation criteria

1. **Accuracy** (absolute avg, min, max error) of generated fused map v/s distances of key-point in simulation. **The expectation is that the avg error should be < 10% (lesser the better)**.
   - The algo and code will be tested on a modified map to verify accuracy of the algorithm

2. **Computational Complexity** (Computational latency in milliseconds to process each set of images from camera and convert them to a composite map) as measured on an Intel Core i5 (10^{th} gen CPU, iGPU can be used). **The expectation it that the overall computational latency should be < 1000ms (lesser the better).**

3. **Novelty, practicality and efficiency of the solution** – this is a subjective metric

Weightage will be given to the above criteria

Source: [TurtleBot3 (robotis.com)]

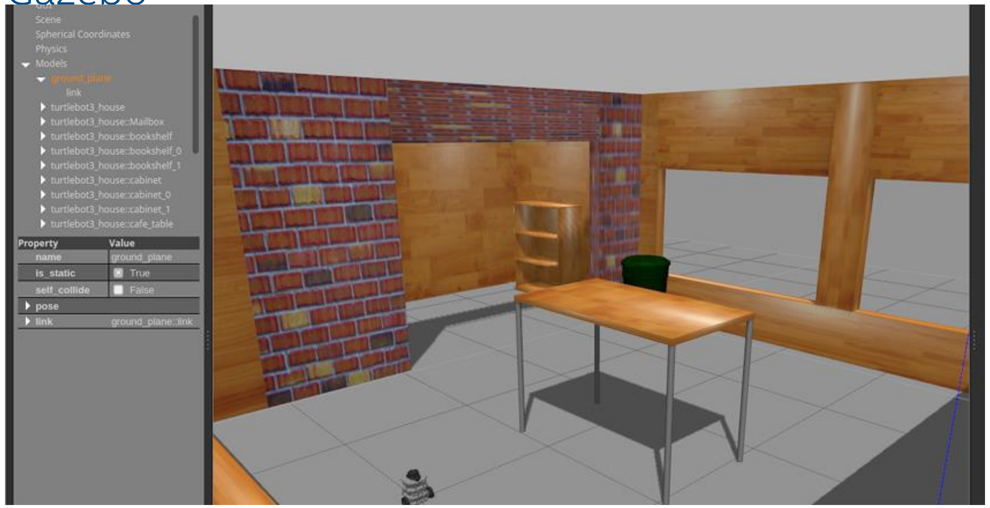
Intro to ROS-Gazebo

Simulating and Testing Robots with ROS and Gazebo

Gazebo is a powerful robot simulation tool that integrates with ROS to create realistic simulations for testing and development.

- **Realistic Physics**: Simulate real-world physics with high accuracy.
- **Sensor Models**: Test and integrate various sensor models like Lidar, cameras, and GPS.
- **Environment Modeling**: Create complex and dynamic environments for testing.
- **ROS Integration**: Seamlessly integrate with ROS for communication and control.
- **Safe Testing**: Test algorithms and robot behaviors in a risk-free environment.
- **Accelerated Development**: Speed up development cycles by using simulation for continuous testing.

Top camera view House simulation in ROS2-Gazebo

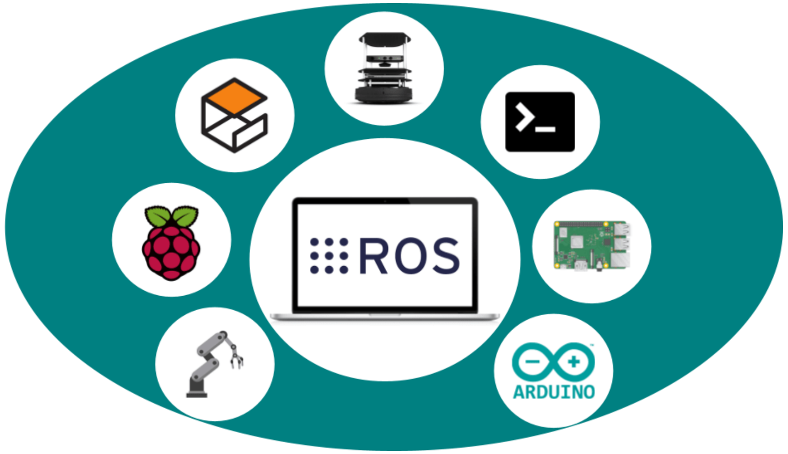


Turtlebot3 robot simulation in house environment

ROS2-Gazebo simulation stack - [ROS2 Robotics Developer Course - Using ROS2 In Python | Udemy]

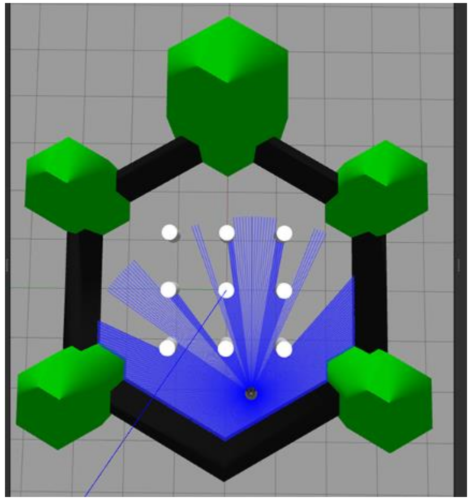
What is ROS?
Robot Operating System

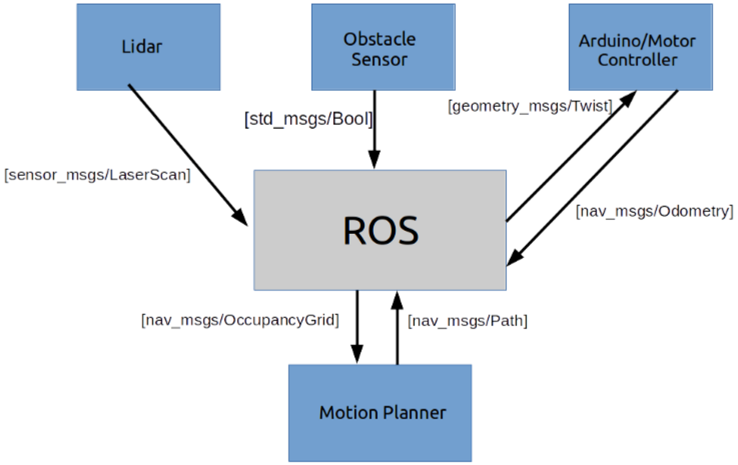
Introduction to ROS2- https://www.udemy.com/course/ros2-for-beginners/



It allows developers to focus on high-level functionality by offering a structured approach to robotics software development, simulation, navigation, and control.

- Open-source
- Works like a publisher/subscriber data exchange model
- Extensive libraries and tools
- Community support





Setting up the ROS2 environment

Ubuntu 20.04 – ROS2 Foxy environment setup

- Ubuntu 20.04
- Install ROS2 foxy, packages – 3.1.2-3.1.4
- Check “ros2 topic list”
- export TURTLEBOT3_MODEL=waffle_pi
- Steps to create ros2 workspace
  - “mkdir –p ~/turtlebot3_ws/src”
  - Install turtlebot3 packages – 6.1.1,6.1.2
  - Run the house simulation
- Check “ros2 topic list”
- Locations Overhead cameras
  [C1: (-5,-2,8) C2:(-5,3,8) C3:(1,-2,8) C4:(1,3,8)]

```
adityam@adityam-NUC8i7BEH:~$ ros2 topic list
/parameter_events
/rosout
adityam@adityam-NUC8i7BEH:~$
```

Before launching simulation

```
adityam@adityam-NUC8i7BEH:~$ ros2 topic list
/clock
/imu
/joint_states
/parameter_events
/performance_metrics
/robot_description
/rosout
/scan
/tf
/tf_static
```

Active topics after launch of simulation

Setting up the ROS2 environment

Adding overhead cameras

- Download the [infraCam.zip]
- Extract the folder “turtlebot3_camera_house” and place in the “……/turtlebot3_ws/src/turtlebot3_simulations /turtlebot3_gazebo/models” directory.
- Replace the waffle_pi.model in “…../turtlebot3_ws/src/turtlebot3_simulations/ turtlebot3_gazebo/worlds/turtlebot3_houses/ ” with the downloaded version.
- Change path in lines 6,11,16,21
- Source and build the workspace
- Run house simulation: “ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py”
- Topics related overhead camera are now visible.

```
adityam@adityam-NUC8i7BEH:~$ ros2 topic list
/camera/camera_info
/camera/image_raw
/clock
/cmd_vel
/imu
/joint_states
/odom
/overhead_camera/overhead_camera1/camera_info
/overhead_camera/overhead_camera1/image_raw
/overhead_camera/overhead_camera2/camera_info
/overhead_camera/overhead_camera2/image_raw
/overhead_camera/overhead_camera3/camera_info
/overhead_camera/overhead_camera3/image_raw
/overhead_camera/overhead_camera4/camera_info
/overhead_camera/overhead_camera4/image_raw
/parameter_events
/performance_metrics
/robot_description
/rosout
/scan
/tf
/tf_static
adityam@adityam-NUC8i7BEH:~$
```

Active topics after launch of simulation with overhead cameras

[Download]

Setting up the environment

How to acquire data from cameras?

- ROS works as a pub-sub model
- The images are being streamed on the topics
- The images can be accessed through the python script
- Always “source /opt/ros/foxy/setup.bash” before running the python script
- 1 – Initiates the nodes (script will be given unique name) and waits for user to interrupt the program
- 2 – Creates subscription to camera topic
- 3 – Every time the camera image is published, listener_callback function is executed.

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

import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class ImageListener(Node):

    def __init__(self):
        super().__init__('image_listener')
        self.subscription = self.create_subscription(
            Image,
            '/overhead_camera/overhead_camera1/image_raw',
            self.listener_callback,
            10)
        self.bridge = CvBridge()

    def listener_callback(self, msg):
        self.get_logger().info('Receiving image')
        cv_image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
        cv2.imshow("Camera Image", cv_image)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args)
    node = ImageListener()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

2

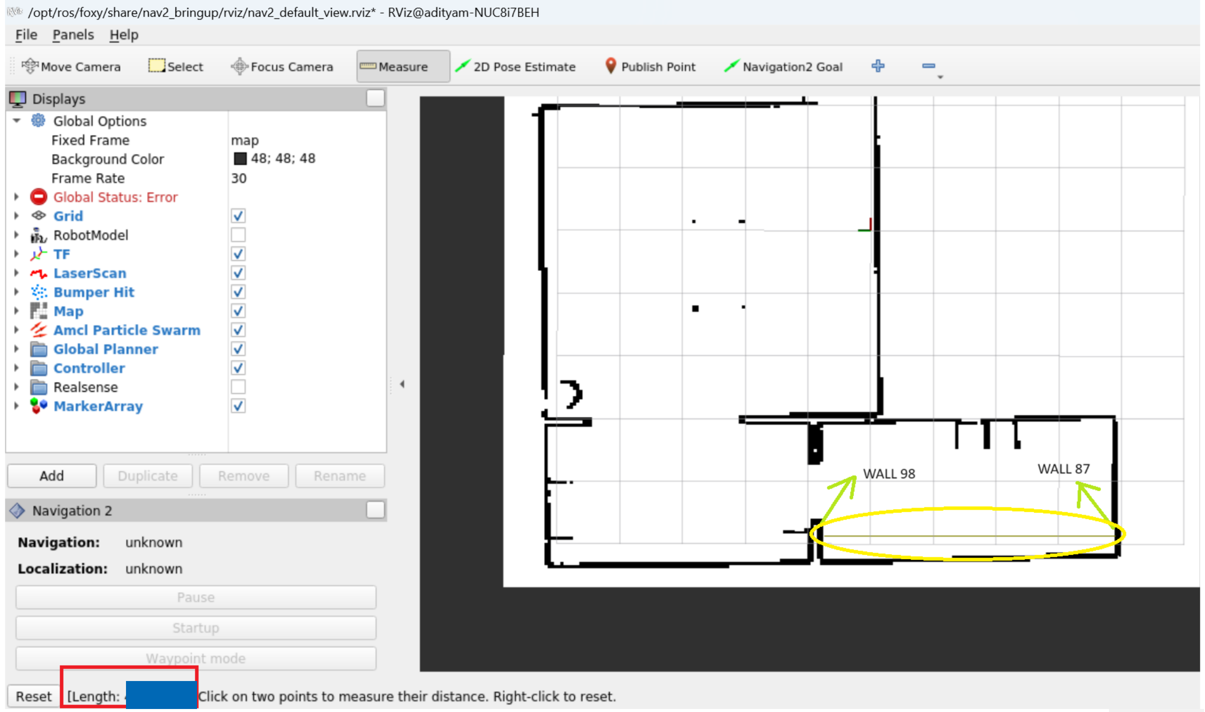
3

1

How to evaluate the generated map?

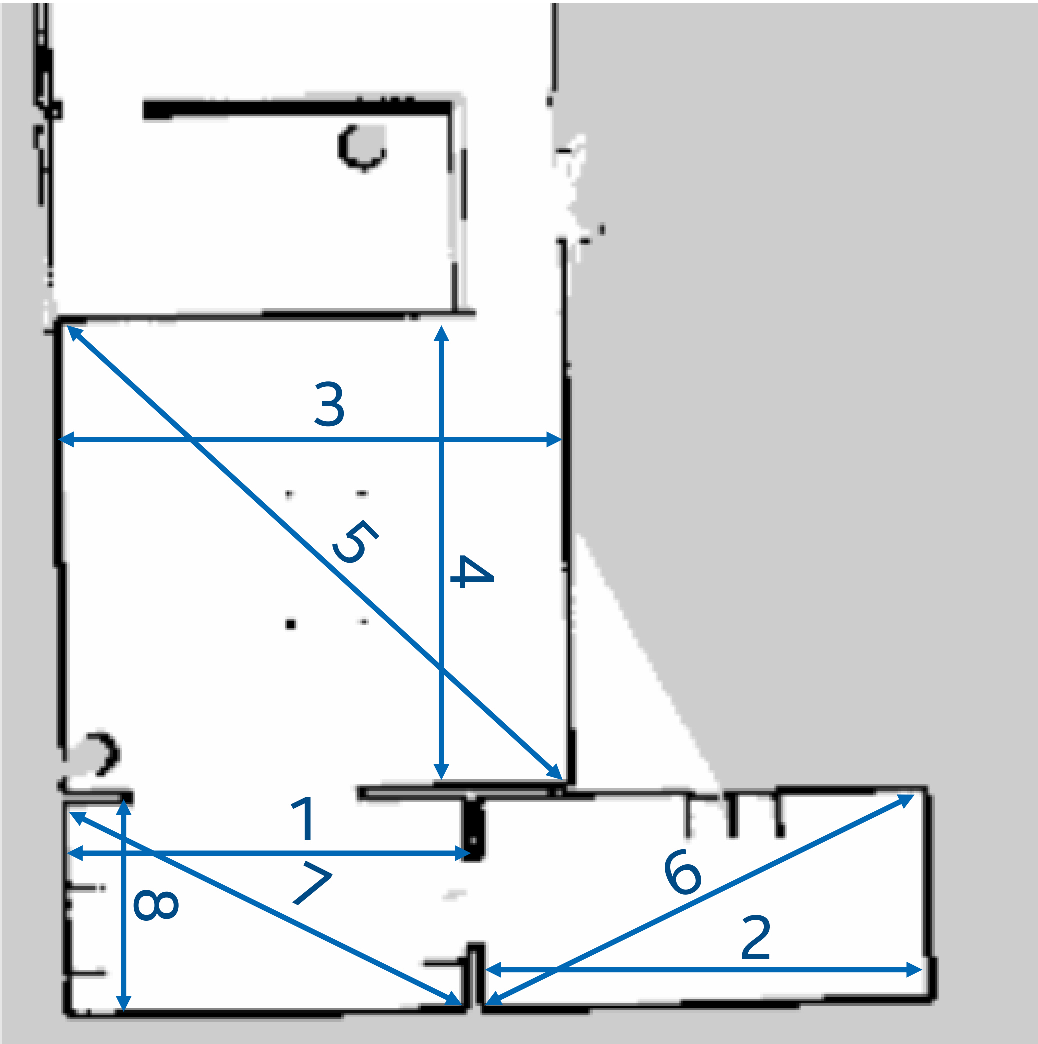
- Run command below (replace path in yaml file)

```
ros2 launch turtlebot3_navigation2
navigation2.launch.py use_sim_time:=True
map:=$HOME/map_house.yaml
```



- Rviz: a visualization tool for ROS
- We have 8 key point to point measurements as benchmarks.
- A map generated with cartographer will be provided for reference.
- [Download]

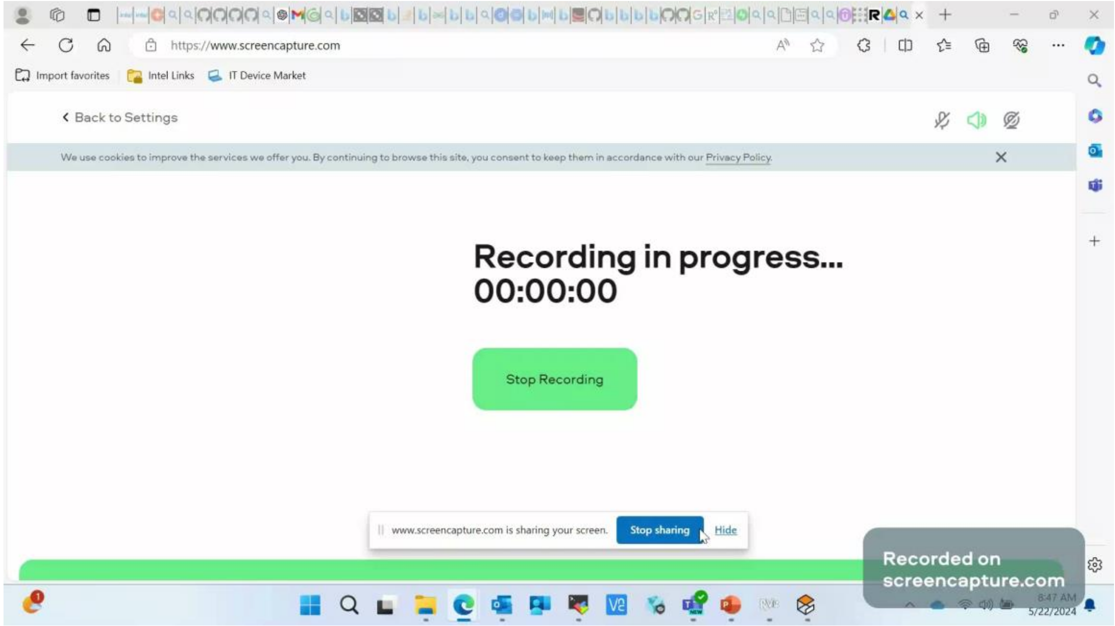
Reference map and 8 Key-points for measurement

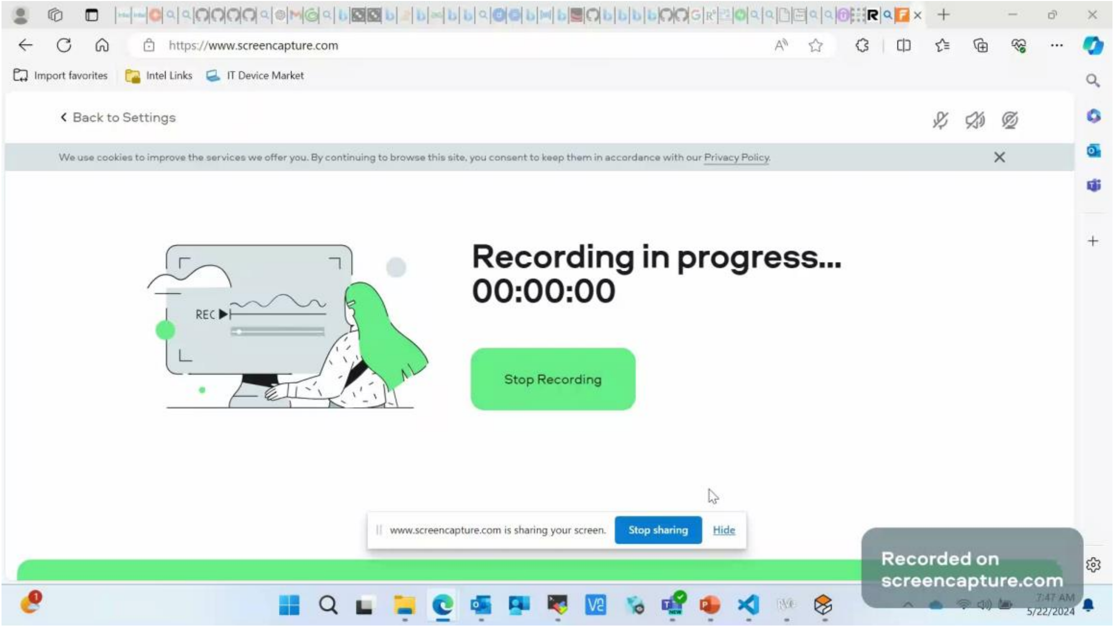


1 – ? m
2 – ? m
3 – ? m
4 – ? m
5 – ? m
6 – ? m
7 – ? m
8 – ? m

How to evaluate the generated map?

Using Rviz to find distance between key points





Questions?