GOOD MORNING!
早上好!
안녕하세요!

DAY 3

DAY 1

- Welcome

- Project Introduction

- Introduction to Project Development Process

- Business Requirement Development

- System Requirement Development

- Time Management

- System(High Level) Design

DAY 2 (MINI PROJECT)

- Yolo객체 인식 모델 활용과 성능 평가 방법 이해
  - Custom Dataset과 Fine Tuning으로 자체 객체 인식 모델 구현 및 평가
  - (Optional)경량화 모델 등 개별 요구사항에 적합한 모델 탐색 및 성능 검증

DAY 2 (MINI PROJECT)

WEB-CAM 기반 객체 인식 (IF NEEDED)

- YOLOv8 기반 데이터 수집/학습/deploy (Detection Alert)
  - 감시용 데이터 수집(rc_car, dummy, 등)
  - 감시용 데이터 라벨링
  - YOLOv8 기반 학습
  - YOLOv8 Object Detection

AMR-CAM 기반 객체 인식

- AMR(Autonomous Mobile Robot) Turtlebot4 개발 환경 구축
- 로봇 개발 환경에 완성 모델 서빙 및 테스트 / 로봇 H/W, 제반 환경의 한계점 도출
  - Tracking 데이터 수집((rc_car, dummy, 등)
  - Tracking 데이터 라벨링
  - YOLOv8 기반 학습
  - YOLOv8 Object Tracking

DAY 3 (MINI PROJECT)

- Auto. Driving 시스템 학습
  - Digital Mapping of environment
  - Operate AMR (Sim. & Real)
  - Tutorial 실행
  - Detection, Depth and AMR 주행
  - 로봇 개발 환경에 적용 및 테스트 / 로봇 H/W, 제반 환경의 한계점 도출

TURTLEBOT4 시뮬레이션

- 환경 구축
- SLAM과 AutoSLAM으로 맵 생성
- Sim. Tutorial 실행
- Detection, Depth and AMR 주행 example

DAY 3 (MINI PROJECT)

REAL ROBOT

- Manually operating the AMR (Teleops)

- autonomous driving 시스템 with obstacle avoidance
  - Digital Mapping of environment
  - Launching Localization, Nav2, and using Rviz to operate a robot
  - Goal Setting and Obstacle Avoidance using Navigation

TUTORIAL

- Turtlebot4 API를 활용한 Initial Pose Navigate_to Pose 구현

- Turtlebot4 API를 활용한 Navigate_Through_pose, Follow Waypoints 구현

DAY 4

- Business Requirement Development

- System Requirement Development

- Time Management

- System(High Level) Design

- Begin Detail Design to Acceptance - Agile Development (SPRINTs)

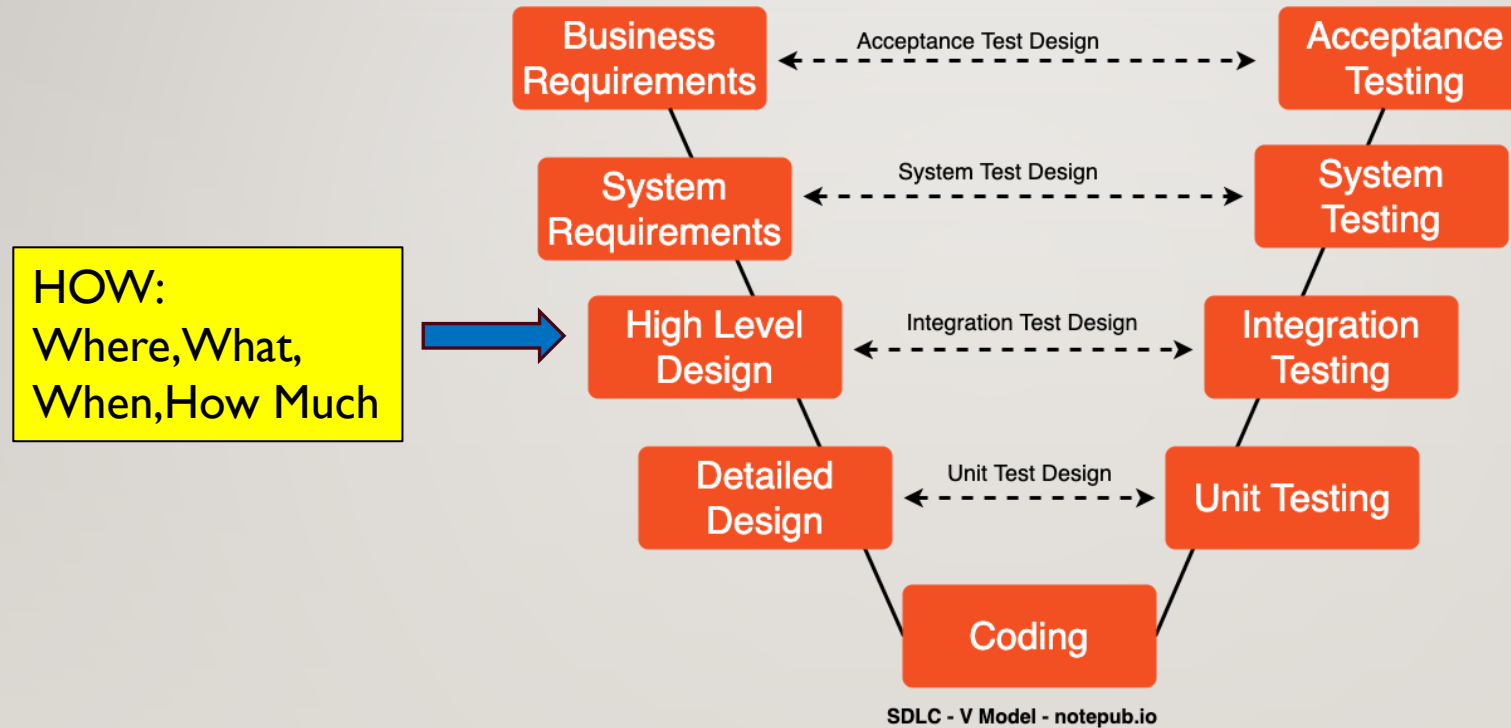# DAY 4 (MINI PROJECT)

SYSTEM DESIGN

- Mini Project

DETAIL DESIGN

- Detection
- AMR Control

프로젝트 RULE NUMBER ONE!!!

Have Fun Fun Fun!

SW DEVELOPMENT PROCESS

HOW:
Where,What,
When,How Much

Business Requirements — Acceptance Test Design — Acceptance Testing

System Requirements — System Test Design — System Testing

High Level Design — Integration Test Design — Integration Testing

Detailed Design — Unit Test Design — Unit Testing

Coding

SDLC - V Model - notepub.io

VISUALIZATION – SYSTEM FUNCTIONAL PROCESS FLOW (ARCHITECTURE) DIAGRAMS

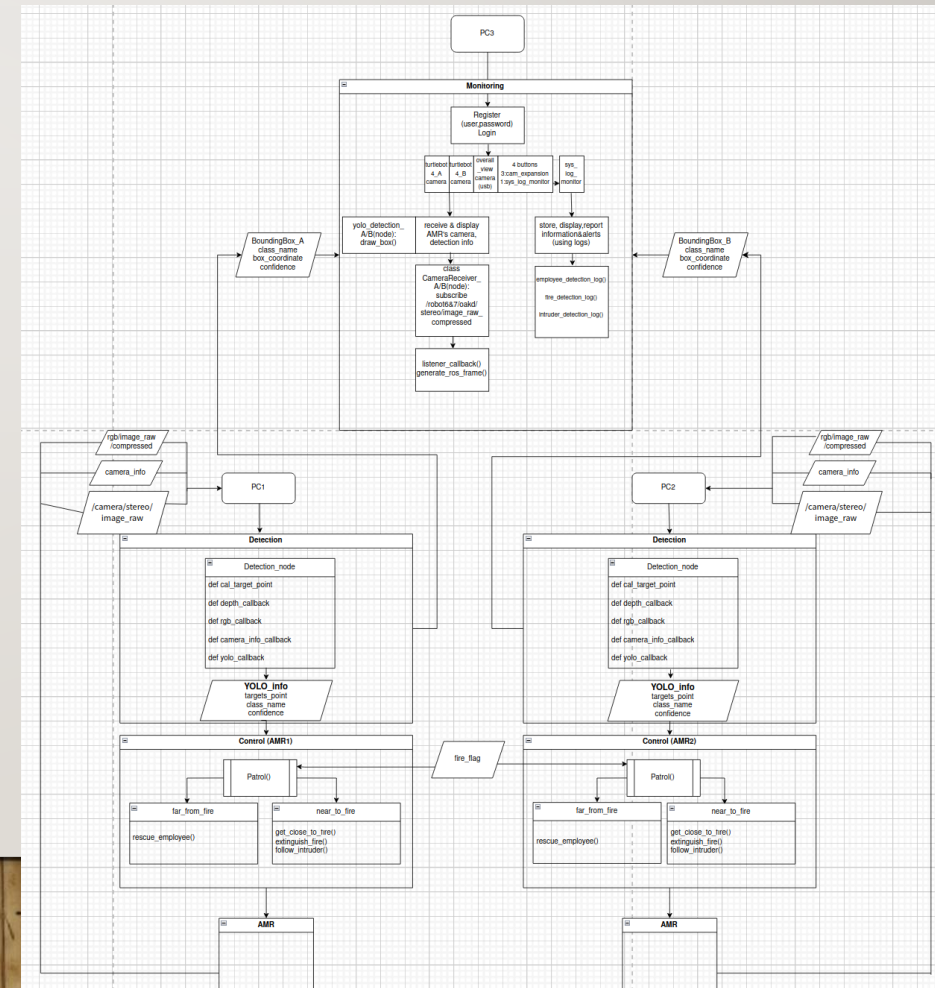- To-Be Functional Process Flow Diagram

  Detection Alert

  AMR Controller

- Functions

- Interfaces

  Dataflow

- Testing

  Error and Exception Handling

EXAMPLE SYSTEM DESIGN DOCUMENT

System Design Document (SDD)

Project Title: Autonomous Mobile Robot (AMR) Security System
Version: 1.1
Date: [Insert Date]

---

1. Overview

The Autonomous Mobile Robot (AMR) Security System is designed to provide autonomous patrolling, threat detection, and alerting within a secure area using a single AI-enabled robot. The system consists of one AMR equipped with necessary hardware and software components to operate independently, processing data on-board without the need for a central server.

---

2. System Architecture

Since the system consists of a single AMR, data processing, navigation, threat detection, and alerting are all performed locally on the AMR itself. The AMR communicates directly with a user interface on a PC via a local network (Wi-Fi) for monitoring, alerts, and manual override if required.

시스템 설계 문서 (SDD)

프로젝트 제목: 자율 이동 로봇(AMR) 보안 시스템
버전: 1.1
날짜: [날짜 삽입]

---

1. 개요

자율 이동 로봇(AMR) 보안 시스템은 단일 AI 기반 로봇을 사용하여 보안 구역 내에서 자율 순찰, 위협 탐지 및 경고를 제공하도록 설계되었습니다. 시스템은 단일 AMR이 독립적으로 작동할 수 있도록 필요한 하드웨어 및 소프트웨어 구성 요소로 구성되며, 중앙 서버 없이 데이터를 현장에서 처리합니다.

---

2. 시스템 아키텍처

이 시스템은 단일 AMR으로 구성되므로 데이터 처리, 네비게이션, 위협 탐지 및 경고가 모두 AMR에서 로컬로 수행됩니다. AMR은 모니터링, 알림 및 수동 제어를 위해 PC의 사용자 인터페이스와 로컬 네트워크(Wi-Fi)를 통해 직접 통신합니다.
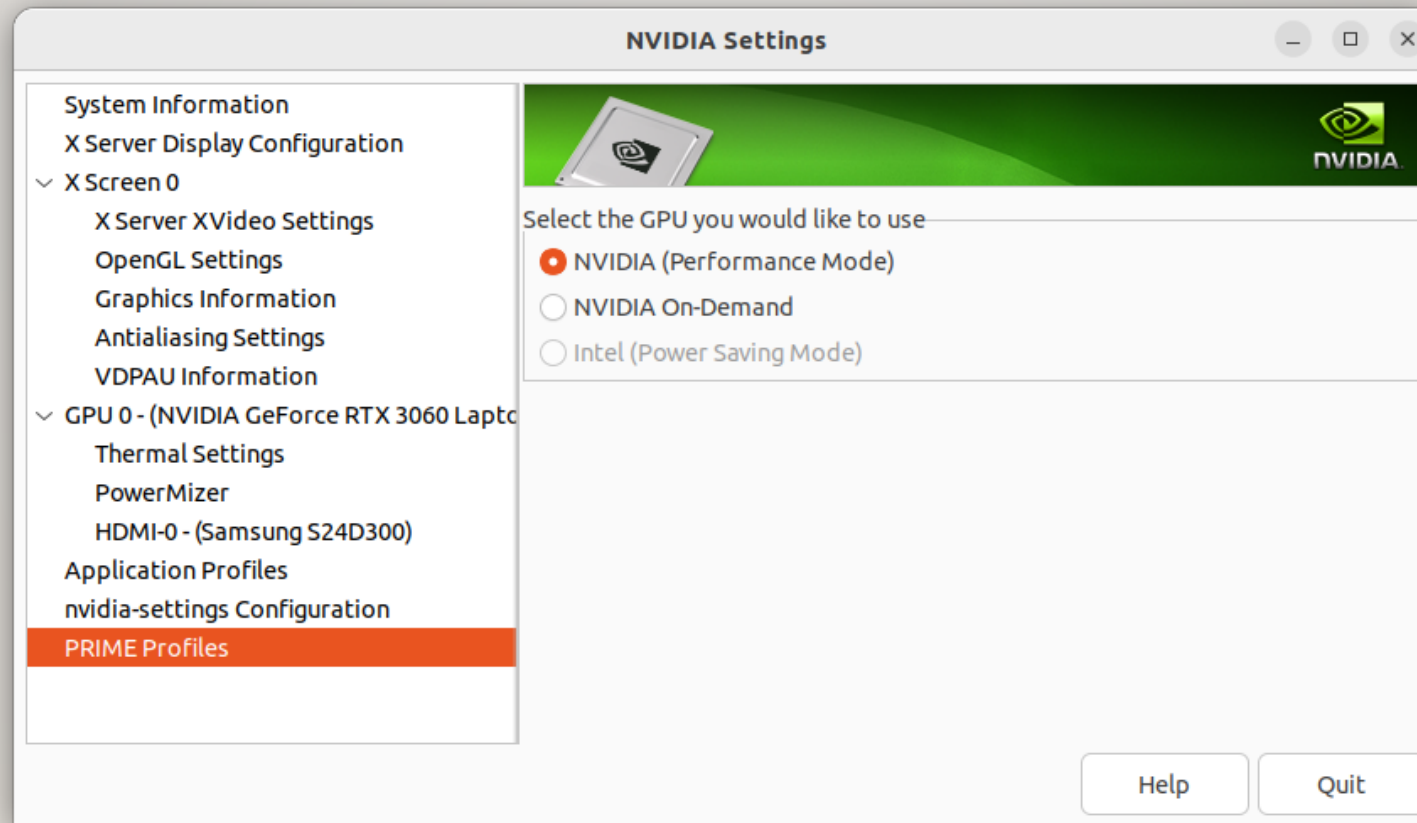
CONTROLLING AMR

AMR (TURTLEBOT4)

- [Features](#) · [User Manual](#)

- https://turtlebot.github.io/turtlebot4-user-manual/overview/features.html

- Review the content

RUNNING IN SIMULATION

SETUP NVIDIA GPU FOR SLAM

SETUP BASH

- Make sure bashrc has:
  - ROS_DOMAIN_ID = 0

- Make sure discovery setup.bash is <mark>**not**</mark> sourced!

- source ~/.bashrc

OPERATING A ROBOT(SIM) – GAZEBO

TERM1

- ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py **rviz:=true**

TERM2

- ros2 topic list

- ros2 topic echo <topic> --once
  - /oakd/rgb/preview/image_raw
  - /oakd/rgb/preview/depth
  - ….

OPERATING A ROBOT(SIM)

- Dock/Undock

- Manual Driving

  - Teleops

- Camera Display

  - RGB/Depth

- Navigation with rviz

  - 2D_Pose_Estimate (initial position)

  - Nav2_Goal

DIGITAL MAPPING USING SLAM (SIM)

TERM1

- ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py **nav2:=true slam:=true rviz:=true**

ON GAZEBO

- Undock the robot

- Use keyboard to operate and complete the map

DIGITAL MAPPING USING SLAM (SIM)

TERM1

- ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py nav2:=true slam:=true rviz:=true

TERM2 (SAVE MAP AFTER MAPPING FINISHES)

- ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name: data: 'map_name'''

  Ex:: ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name: data:'my_map'''
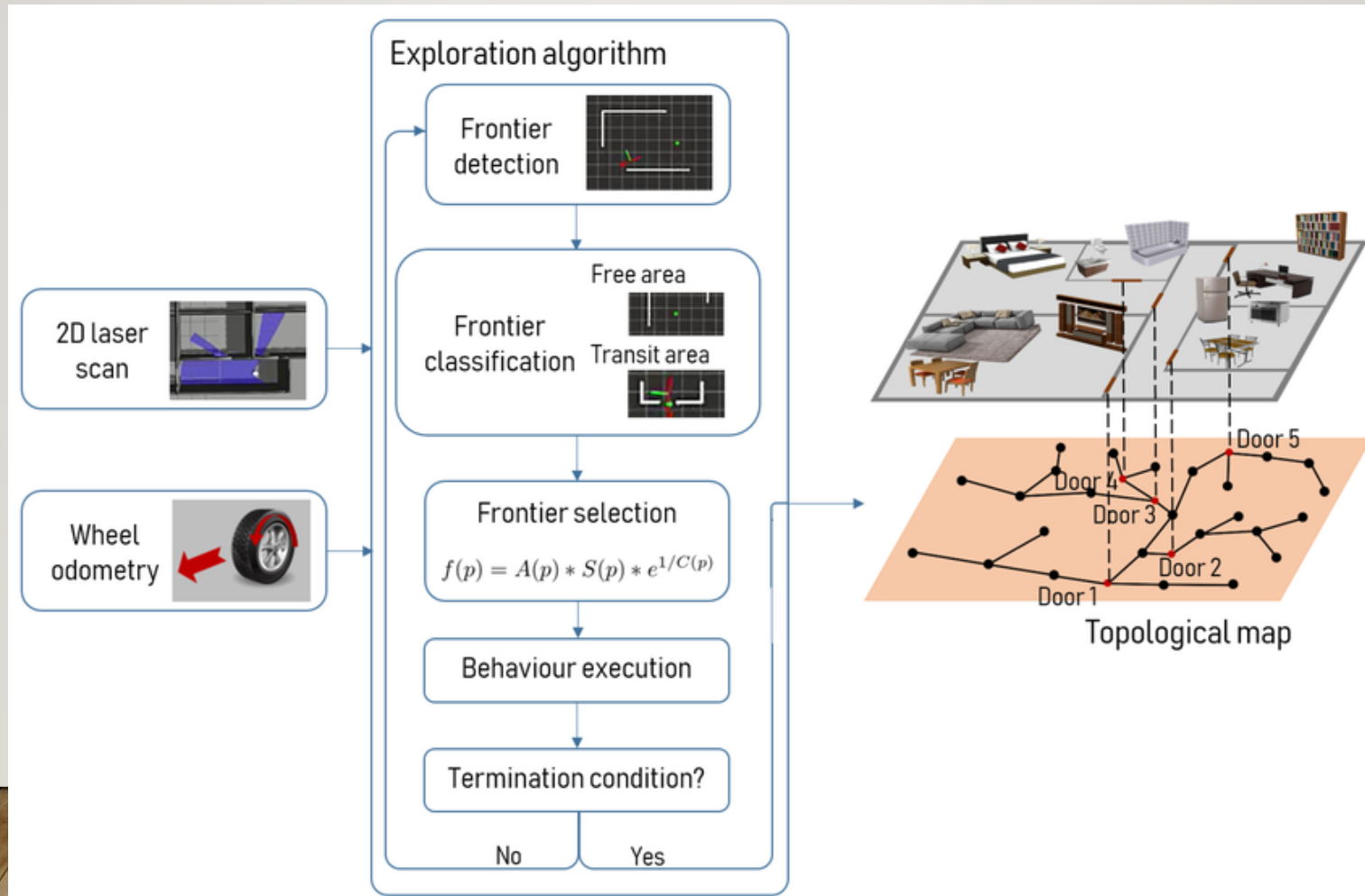
DIGITAL MAPPING WITH AUTO – SLAM (SIM)

TERM1

- ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py **nav2:=true slam:=true rviz:=true**

- Undock the robot

- Set init pose from rviz

TERM2

- ros2 launch explore_lite explore.launch.py

AUTO SLAM CONCEPT/ALGORITHM

Exploration algorithm

Frontier detection

Frontier classification — Free area — Transit area

Frontier selection

$$f(p) = A(p) * S(p) * e^{1/C(p)}$$

Behaviour execution

Termination condition?

No   Yes

2D laser scan

Wheel odometry

Door 1, Door 2, Door 3, Door 4, Door 5

Topological map

ALGORITHM DETAIL

- **Map Subscription**

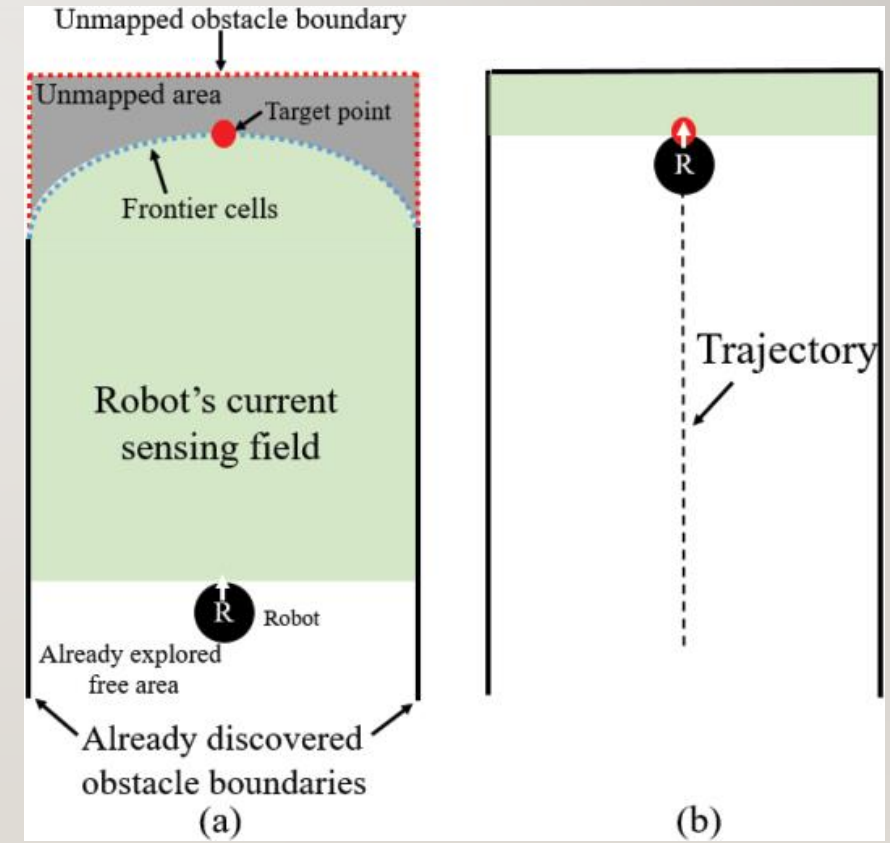explore_lite subscribes to the SLAM-generated occupancy grid (/map topic) and identifies:

- Free space: known, unoccupied areas
- Occupied space: obstacles
- Unknown space: unexplored

- **Frontier Detection**

The map is scanned for cells that:

- Are free, and
- Are adjacent to at least one unknown cell.

These are marked as frontier cells.

ALGORITHM DETAIL

- Frontier Grouping
  - Frontier cells are clustered into connected regions.
  - Each group represents a potential exploration target.

- Goal Selection
  - For each frontier group, a representative point (typically the centroid or closest point) is selected.
  - The robot scores each group based on:
    - Distance from the robot
    - Information gain (how much new area might be revealed)
  - The best-scoring frontier is chosen as the next goal.
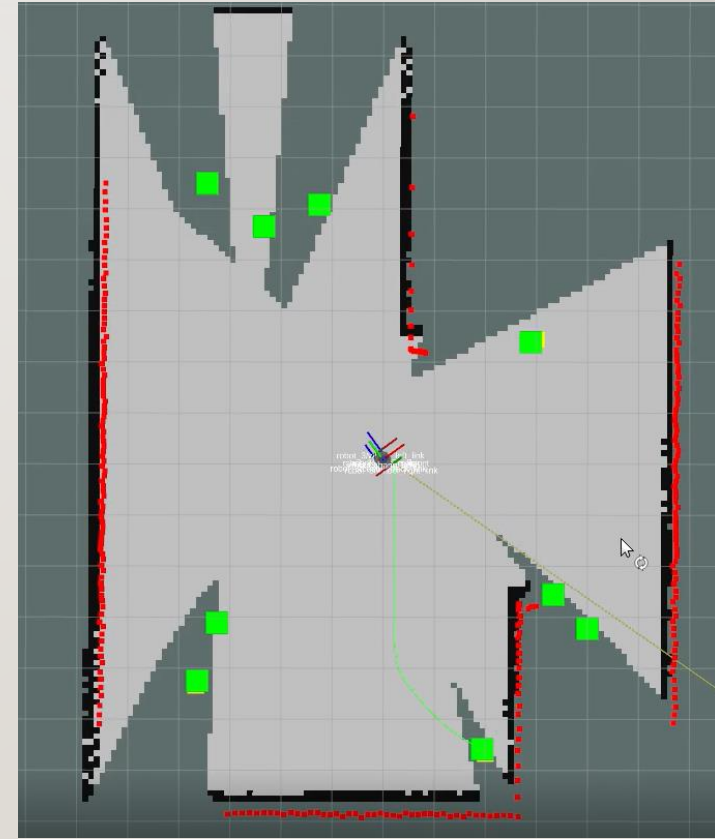
ALGORITHM DETAIL

- Termination

    While (frontiers exist and reachable)

      Select best frontier

      Send as goal

      If goal fails → blacklist

    If (no frontiers or all blacklisted)

      Terminate exploration

DIGITAL MAPPING WITH AUTO – SLAM (SIM)

TERM3 (SAVE MAP AFTER MAPPING FINISHES)

$ ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name: data: 'map_name'''

Ex:: ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name: data:'my_map'''

CHECKING DIGITAL MAP (SIM)

CHECK IF CORRECT

$ xdg-open <map-path>/map.pgm

Or,

$ eog <map-path>/map.pgm

TUTORIAL(SIM)

- [TurtleBot 4 Navigator · User Manual](https://turtlebot.github.io/turtlebot4-user-manual/tutorials/turtlebot4_navigator.html)

  https://turtlebot.github.io/turtlebot4-user-manual/tutorials/turtlebot4_navigator.html

TUTORIAL(SIM)

TERMINAL 1

$ ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py **nav2:=true slam:=false localization:=true rviz:=true**

• Undock and set init pose

TERMINAL 2

$ ros2 run turtlebot4_python_tutorials nav_to_pose

$ ros2 run turtlebot4_python_tutorials nav_through_poses

$ ros2 run turtlebot4_python_tutorials follow_waypoints

$ ros2 run turtlebot4_python_tutorials create_path

$ ros2 run turtlebot4_python_tutorials mail_delivery

$ ros2 run turtlebot4_python_tutorials patrol_loop

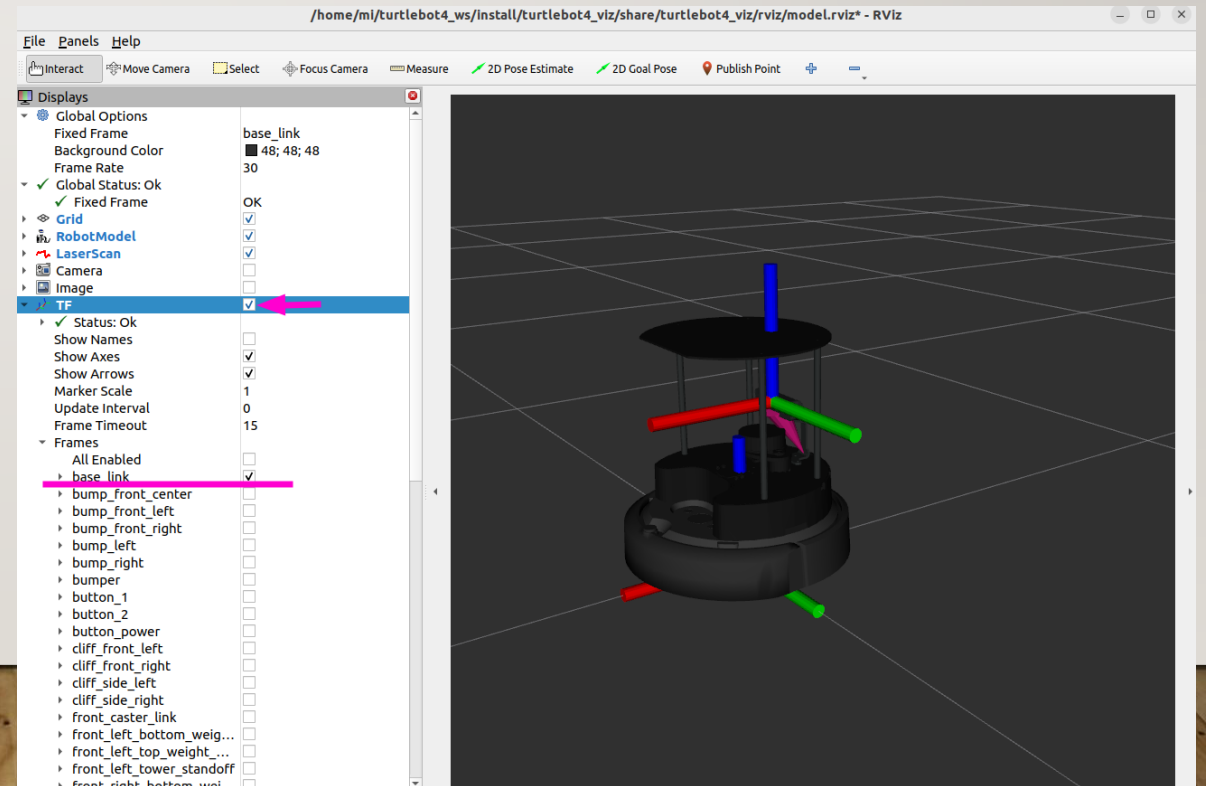# USING DEPTH (TF TRANSFORM)

CAMERA INTRINSIC AND REPROJECTION
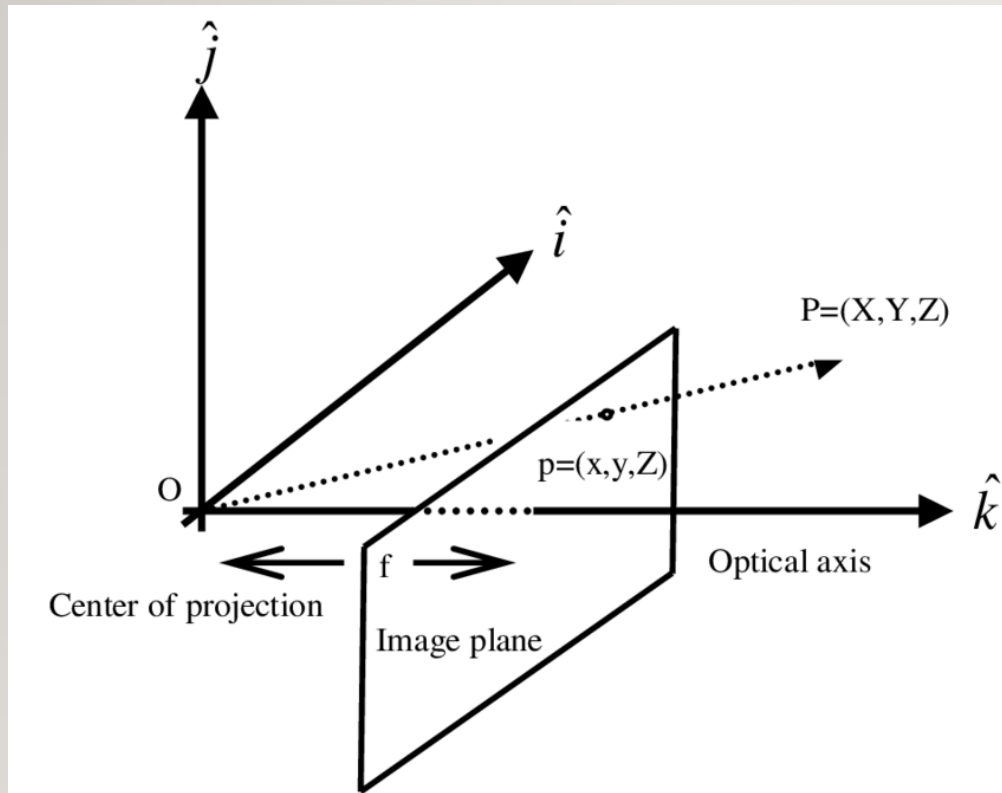
$$X = \frac{(u - c_x) \cdot Z}{f_x}, \quad Y = \frac{(v - c_y) \cdot Z}{f_y}, \quad Z = Z$$

ROS TRANSFORM EXPLAINED

- **Transform Explained (1)**

- https://indecisive-freedom-6e8.notion.site/Transform-Explained-1-22f8e215779c809eb2d4cb0be5538056

USING DEPTH (SIM)

TERMINAL 1

- ros2 launch turtlebot4_ignition_bringup turtlebot4_ignition.launch.py **nav2:=true slam:=false localization:=true rviz:=true**

TERMINAL 2


```
3_1_a_depth_checker.py
3_1_b_depth_to_3d.py
3_1_c_depth_to_nav_goal.py
3_1_d_nav_to_person.py
```

*create a necessary ROS2 package and run

OPERATING THE REAL ROBOT

SETUP BASH

- Make sure bashrc has:

  - ROS_DOMAIN_ID = 0

- echo "alias **ros-restart**='ros2 daemon stop; ros2 daemon start'" >> ~/.bashrc

- Make sure discovery setup.bash **is** sourced!

- source ~/.bashrc

CONTROLLING THE AMR MOVEMENT

- Teleop with keyboard

  - [Driving your TurtleBot 4 · User Manual](#)

  - https://turtlebot.github.io/turtlebot4-user-manual/tutorials/driving.html

- Navigation with SLAM

UNDOCKING AND DOCKING THE ROBOT

TERMINAL1

- Undocking

$ ros2 action send_goal /robot<n>/undock irobot_create_msgs/action/Undock "{}"

TERMINAL 1

- Docking

$ ros2 action send_goal /robot<n>/dock irobot_create_msgs/action/Dock "{}"

DRIVING YOUR ROBOT

TERMINAL 2

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r /cmd_vel:=/robot<n>/cmd_vel

DIGITAL MAPPING (SLAM)

- [Generating a map · User Manual](https://turtlebot.github.io/turtlebot4-user-manual/tutorials/generate_map.html)
  https://turtlebot.github.io/turtlebot4-user-manual/tutorials/generate_map.html

DIGITAL MAPPING (MANUAL SLAM)

- Undock your robot

TERMINAL 1

$ ros2 launch turtlebot4_navigation slam.launch.py **namespace:=/robot <n>**

TERMINAL 2

$ ros2 launch turtlebot4_viz view_robot.launch.py **namespace:=/robot <n>**

- Undock and set init pose

TERMINAL 3

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r /cmd_vel:=/robot**<n>**/cmd_vel

DIGITAL MAPPING (MANUAL SLAM)

- Terminal 1

$ ros2 launch turtlebot4_navigation slam.launch.py namespace:=/robot <n>

- Terminal 2

$ ros2 launch turtlebot4_viz view_robot.launch.py namespace:=/robot <n>

- Terminal 3

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r /cmd_vel:=/robot<n>/cmd_vel

TERMINAL 4

$ cd <map_directory>

$ ros2 run nav2_map_server map_saver_cli -f "<map_name>" --ros-args -p map_subscribe_transient_local:=true -r __ns:=/robot <n>

DIGITAL MAPPING (AUTO – SLAM)

- Undock your robot

TERMINAL 1

$ ros2 launch turtlebot4_navigation slam.launch.py **namespace:=/robot<n>**

TERMINAL 2

$ ros2 launch turtlebot4_viz view_robot.launch.py **namespace:=/robot<n>**

- Undock and set init pose

TERMINAL 3

$ ros2 launch turtlebot4_navigation nav2.launch.py **namespace:=robot<n>**

$ **ros-restart** #execute if fails/as needed

TERMINAL 4

$ ros2 launch explore_lite explore.launch.py **namespace:=/robot<n>**

DIGITAL MAPPING (AUTO – SLAM)

TERMINAL 5

$ cd <map_directory>

$ ros2 run nav2_map_server map_saver_cli -f "<map_name>" --ros-args -p map_subscribe_transient_local:=true -r __ns:=/robot<n>

NAVIGATION W/ MAP

- [Navigation · User Manual](#)

- https://turtlebot.github.io/turtlebot4-user-manual/tutorials/navigation.html

TERMINAL 1

$ ros2 launch turtlebot4_navigation localization.launch.py namespace:=robot<n> map:=$HOME/Documents/room/room_map.yaml

TERMINAL 2

$ ros2 launch turtlebot4_viz view_robot.launch.py namespace:=/robot<n>

- Undock and set init pose

TERMINAL 3

$ ros2 launch turtlebot4_navigation nav2.launch.py namespace:=/robot<n>

$ ros-restart #execute if fails/as needed

TUTORIAL EXERCISE

Make copy and Update the tutorial code to successfully execute in the project environment

TUTORIAL

TERMINAL 1

$ ros2 launch turtlebot4_navigation localization.launch.py namespace:=/robot<n> map:=$HOME/Documents/room/room_map.yaml

TERMINAL 2

$ ros2 launch turtlebot4_viz view_robot.launch.py namespace:=/robot <n>

- Undock and set init pose

TERMINAL 3

$ ros2 launch turtlebot4_navigation nav2.launch.py namespace:=/robot <n>

SIMPLE NAV2 PARAM ADJUSTMENT

$ cd ~/turtlebot4_ws/src/turtlebot4/turtlebot4_navigation/config

- Change/adjust "inflation_radius" to fit your environment

```
139
140 local_costmap:
141   local_costmap:
142     ros__parameters:
143       update_frequency: 5.0
144       publish_frequency: 2.0
145       global_frame: odom
146       robot_base_frame: base_link
147       use_sim_time: True
148       rolling_window: true
149       width: 3
150       height: 3
151       resolution: 0.06
152       robot_radius: 0.175
153       plugins: ["static_layer", "voxel_layer", "infl
154       inflation_layer:
155         plugin: "nav2_costmap_2d::InflationLayer"
156         cost_scaling_factor: 4.0
157
158         #inflation_radius: 0.45
159         inflation_radius: 0.25
160         #changed by aak
161
162       voxel_layer:
163         plugin: "nav2_costmap_2d::VoxelLayer"
164         enabled: True
```

TUTORIAL

```
3_2_a_nav_to_pose.py
3_2_b_nav_through_poses.py
3_2_c_follow_waypoints.py
3_2_d_create_path.py
3_2_e_mail_delivery.py
3_2_f_patrol_loop.py
```

TERMINAL 4

$ ros2 run day3 create_path --ros-args -r __ns:=/robot<n>

$ ros2 run day3 nav_to_poses --ros-args -r __ns:=/robot<n>

$ ros2 run day3 follow_waypoints --ros-args -r __ns:=/robot<n>

$ ros2 run day3 nav_through_poses --ros-args -r __ns:=/robot<n>

$ ros2 run day3 mail_delivery --ros-args -r __ns:=/robot<n>

$ ros2 run day3 patrol_loop --ros-args -r __ns:=/robot<n>

USING DEPTH (ROBOT)

SETUP BASH(ROBOT)

- Make sure bashrc has:
    - ROS_DOMAIN_ID = 0

- Make sure discovery setup.bash **is** sourced!

- source ~/.bashrc

USING DEPTH (ROBOT)

TERMINAL 1

$ ros2 launch turtlebot4_navigation localization.launch.py namespace:=/robot<n> map:=$HOME/Documents/room/room_map.yaml

TERMINAL 2

$ ros2 launch turtlebot4_viz view_robot.launch.py namespace:=/robot<n>

- Set Init Pose using 2D_PoseEstimate

TERMINAL 3

$ ros2 launch turtlebot4_navigation nav2.launch.py namespace:=/robot <n>

USING DEPTH (ROBOT)

```
3_1_a_depth_checker.py
3_1_b_depth_to_3d.py
3_1_c_depth_to_nav_goal.py
3_1_d_nav_to_person.py
3_2_a_nav_to_pose.py
3_2_b_nav_through_poses.py
3_2_c_follow_waypoints.py
3_2_d_create_path.py
3_2_e_mail_delivery.py
3_2_f_patrol_loop.py
3_3_a_depth_checker.py
3_3_b_depth_to_3d.py
3_3_c_depth_to_nav_goal.py
3_3_d_nav_to_car.py
```

```
3_3_a_depth_checker.py
3_3_b_depth_to_3d.py
3_3_c_depth_to_nav_goal.py
3_3_d_nav_to_car.py
```

Exercise: using the simulation code develop the code for the actual robot

USING DEPTH (ROBOT)

3_3_a_depth_checker.py
3_3_b_depth_to_3d.py
3_3_c_depth_to_nav_goal.py
3_3_d_nav_to_car.py

$ ros2 run day4 depth_checker **--ros-args -r __ns:=/robot<n>**

$ ros2 run day4 depth_to_3d **--ros-args -r __ns:=/robot<n> -r /tf:=/robot<n>/tf –r /tf_static:=/robot<n>/tf_static**

$ ros2 run day4 depth_to_goal **--ros-args -r __ns:=/robot<n> -r /tf:=/robot<n>/tf –r /tf_static:=/robot<n>/tf_static**

프로젝트 RULE NUMBER ONE!!!

Are we still having FUN!