



# Parking Robot Based on 3D LiDAR

XJEL3875 Individual Engineering project

Author: Hu, Yadong

Supervisor: Sun, Yongkui

# CONTENTS

---

01

## Background & Motivation

Development of Automated Vehicle;  
Problems; Literature Review; Objectives.

02

## Method & Improvement

SLAM+; SLAM+ Performance; RRT+;  
Speed Bumps & Arch Gates Recognition.

03

## System Architecture

Architecture; Gazebo Setup; Simulations  
and Octomap; Protocols; Server; App.

04

## Result & Conclusion

Requirements; Scenario 1: Outdoor  
Parking Lot; Scenario 2; Conclusion



# 01

## Background & Motivation

Development of Automated Driving,  
Problems and significance, literature review and objectives

# Automated Driving Development

Mar. 2018

## First Self-driving Road Test

- Level 2 AV is already on the ground
- Mostly based on the computer vision

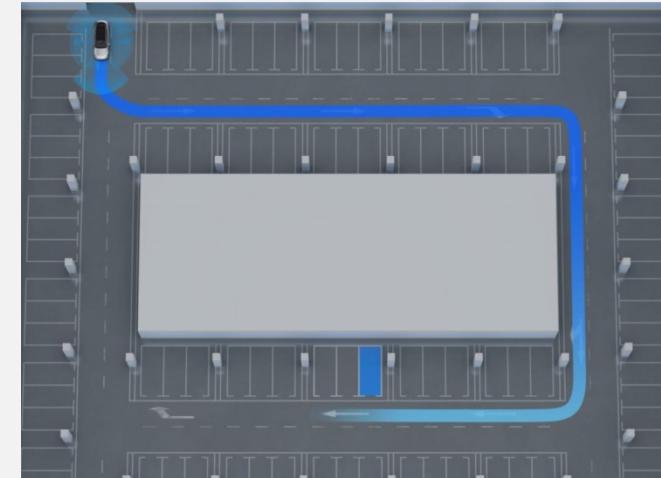


Source: <https://www.gigabyte.com/jp/Article/constructing-the-brain-of-a-self-driving-car>

Jun. 2021

## First Automatic Parking Release

- To reduce the parking time
- To ensure the parking safety
- .....



Source: <https://www.pingwest.com/a/243965>

Early 2025

## IoT-based L4 Self-driving

- Hybrid solution of 3D LiDAR and camera
- Driver attention is not required for safety
- Fully automatic control of the terminal
- .....



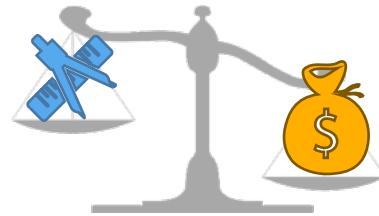
Source: <https://www.walleniuswilhelmsen.com/insights/the-future-of-mobility-whats-the-road-ahead-for-self-driving-vehicles>

# Problems & Significance

1

## 2D or 3D LiDAR?

- High-definition 3D LiDAR is very expensive (>\$15k) but can provide multi-dimensional information without depth camera and supercomputing platforms.
- Normal 2D LiDAR is cheap (~\$0.5k) but other sensors are needed to work in cooperation, and 2D LiDAR only plays a supporting role.



2

## No Remote Control?

- The existing automatic parking system can only be controlled by the car display.
- However, by remote control, driver gets off the car at the entrance of parking lot; or the car goes to the designated place to pick the driver.
- This saves up to 107 hours a year according to INRIX.



# Literature Review



2016

2017

2018

2019

2020

2021

## Cartographer

Developed by Google. Still in maintenance.

## LOAM

Developed by Dr. Zhang Ji at CMU.

**Low speed** but able to **auto optimization**.

## LeGO-LOAM

Developed by Dr. Shan Tixiao at MIT.

**Improved speed and accuracy** of LOAM.  
Middle speed. **Not suitable for navigation**.

## LIO-SAM

Developed by Dr. Shan Tixiao at MIT.

Add IMU & GPS for **high precision and efficiency**.  
**IMU data is not easy to be configured**.

# Objectives

To save **MONEY** and **TIME!**

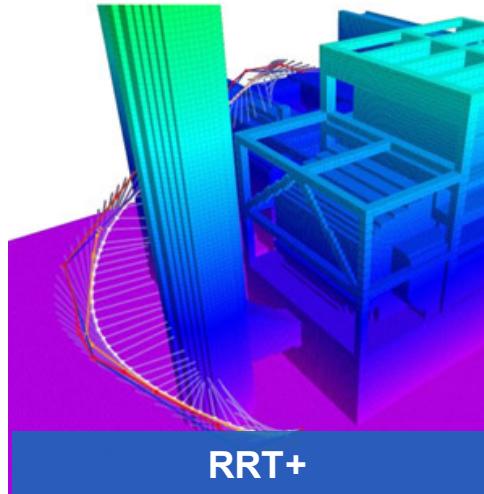


Simultaneous Localization and Mapping +

Inexpensive 3D LiDAR still has a satisfied precision and accuracy with improved method (**IMU**).

Source: <https://geoawesomeness.com/velodyne-500-lidar/>

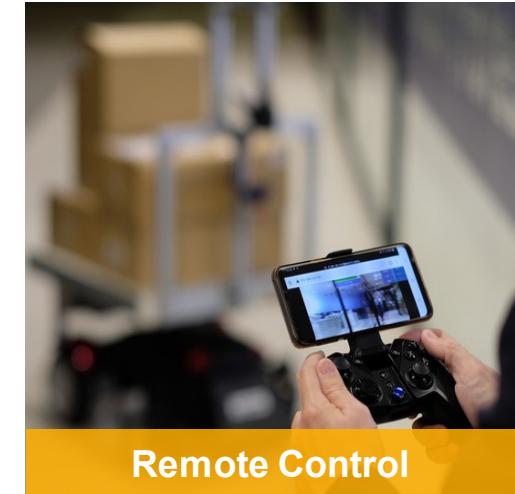
Running in **Simulation Environment**



Rapid Random-explore Tree +

Real-time and dynamic fast navigation **added height info detection** for 3D Navi.

Source: <https://onlinelibrary.wiley.com/doi/10.1002/rob.21863>



Client-Server-Vehicle Arch.  
Able to control and have visualizations through smartphone.

Source: <https://re-how.net/all/1150145>



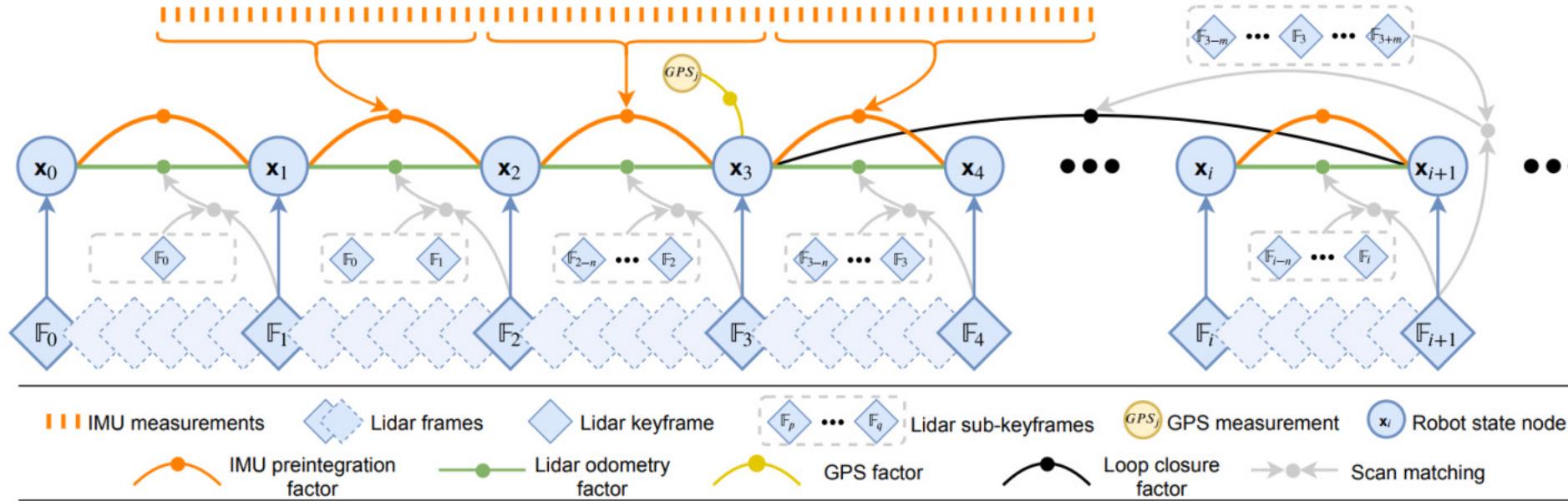
# 02

## **Method & Improvement**

SLAM+; SLAM+ Performance;

RRT+; Speed Bumps & Arch Gates Recognition

# SLAM+: Added IMU for Better Map Building



**Added:**  
**Inertial Measurement Unit**

Pre-integration Factor

$$v_{t+\Delta t} = v_t + g\Delta t + R_t(\hat{a}_t - b_t^a - n_t^a)\Delta t$$

$$p_{t+\Delta t} = p_t + v_t\Delta t + \frac{1}{2}g\Delta t^2 + \frac{1}{2}R_t(\hat{a}_t - b_t^a - n_t^a)\Delta t^2$$

$$R_{t+\Delta t} = R_t \exp((\hat{\omega}_t - b_t^\omega + n_t^\omega)\Delta t)$$

$$\Delta v_{ij} = R_i^T(v_j - v_i - g\Delta t_{ij})$$

$$\Delta p_{ij} = R_i^T(p_j - p_i - v_i \Delta t_{ij} - \frac{1}{2}g\Delta t_{ij}^2)$$

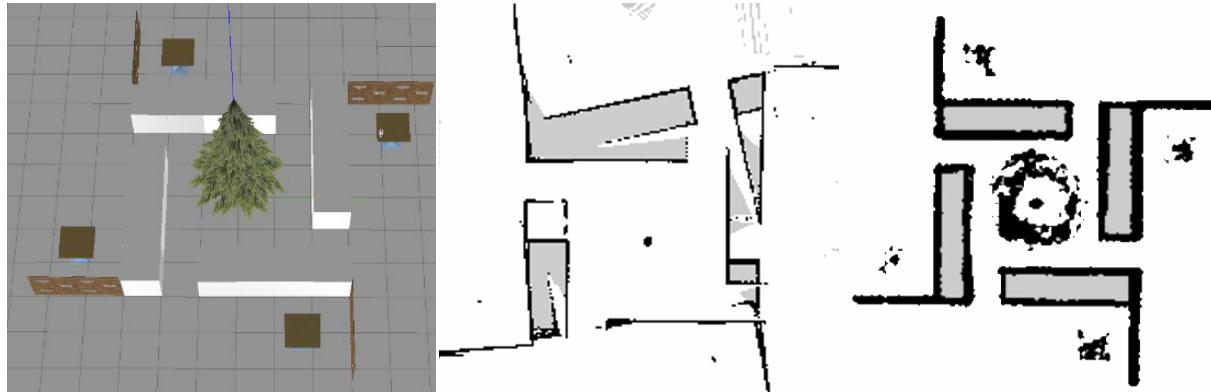
$$\Delta R_{ij} = R_i^T R_j.$$

# SLAM+ Performance

## 2D vs. 3D LiDAR

2D LiDAR identified the tree as a tiny dot.

3D LiDAR correctly restored the leaves and branches.

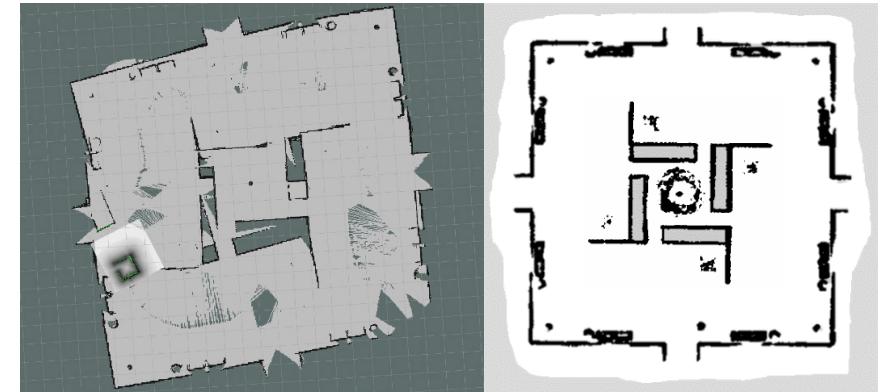


Left: Real-world scene. Middle: Cartographer result. Right: SLAM+ result.

## Cartographer vs. SLAM+

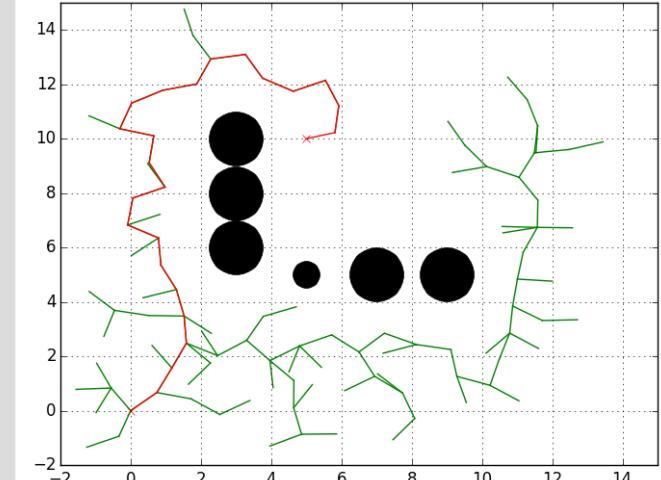
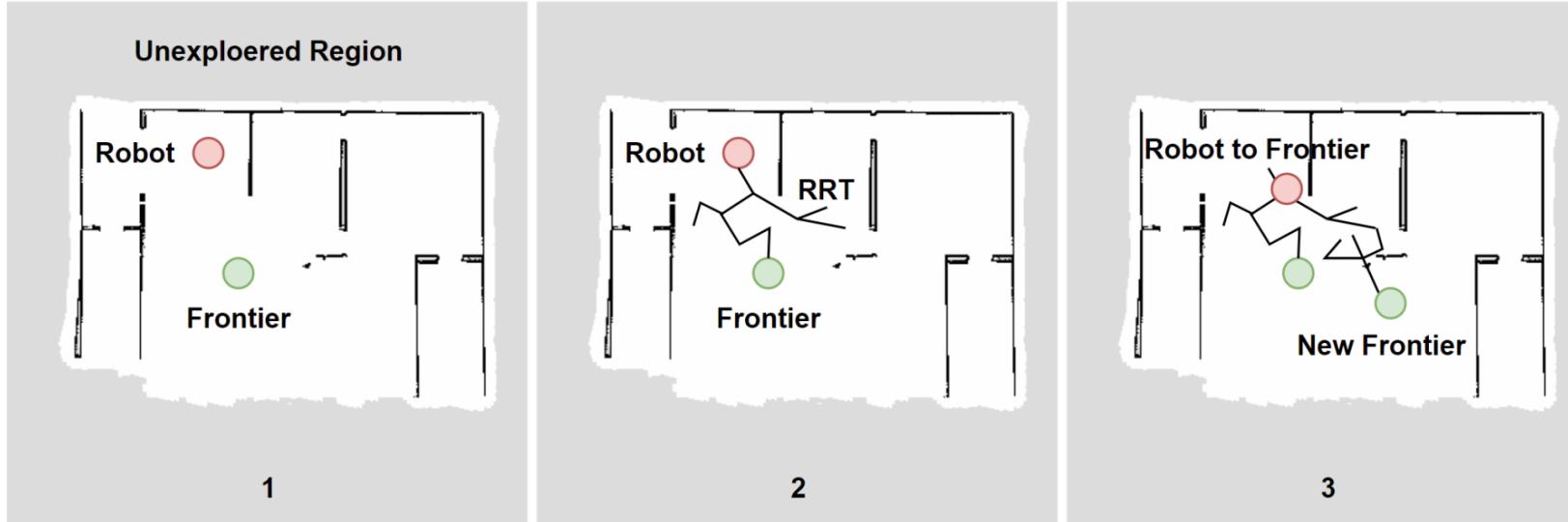
Cartographer had distorted walls.

SLAM+ kept precision and scale well.



Left: Cartographer map building process. The map cannot be used for navigation. Right: SLAM+ whole map result.

# RRT+: An Improved 3D Navigation



Source: <https://myenigma.hatenablog.com/entry/2016/03/23/092002>

**Added Detection for floating PCDs on the path**

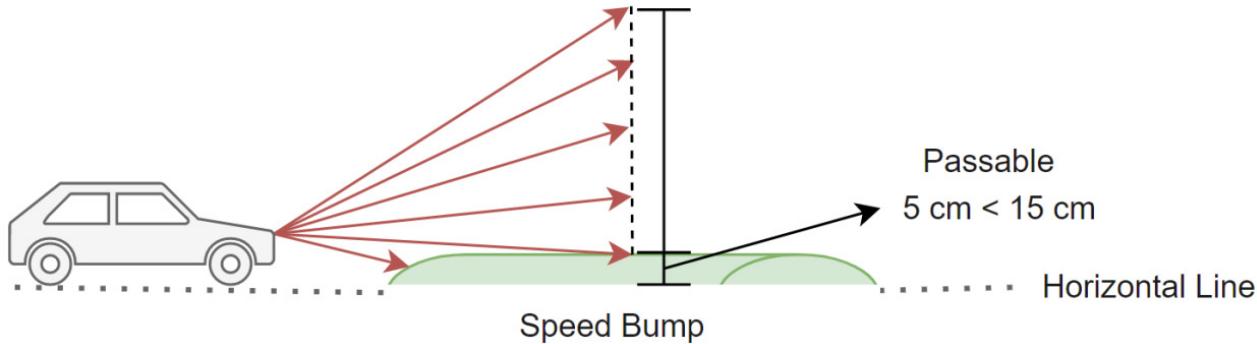
**Rapidly-exploring Random Tree**

In RRT's path checking functions, they only check the new front points' X and Y positions. By adding the **Z-axis PCD-free check**, the robot can detect real-time dynamic obstacles.

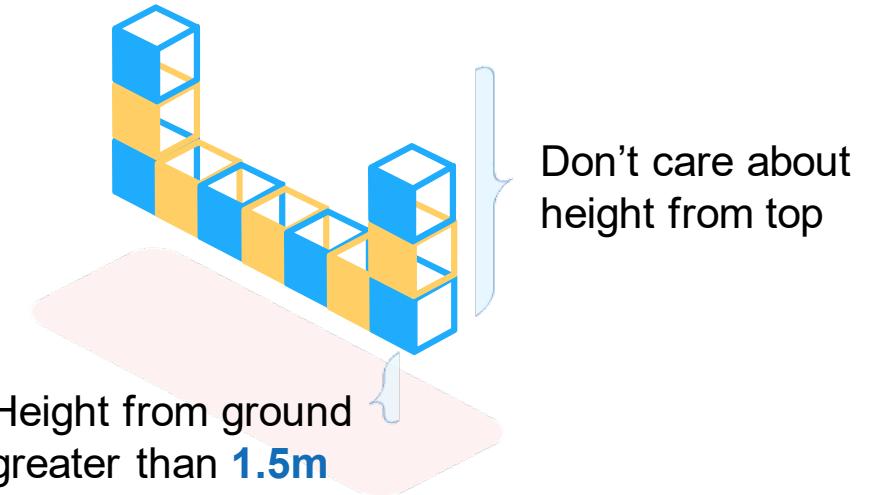
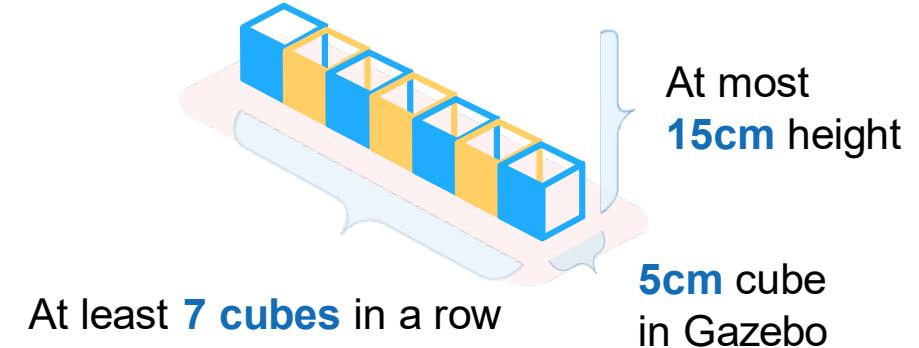
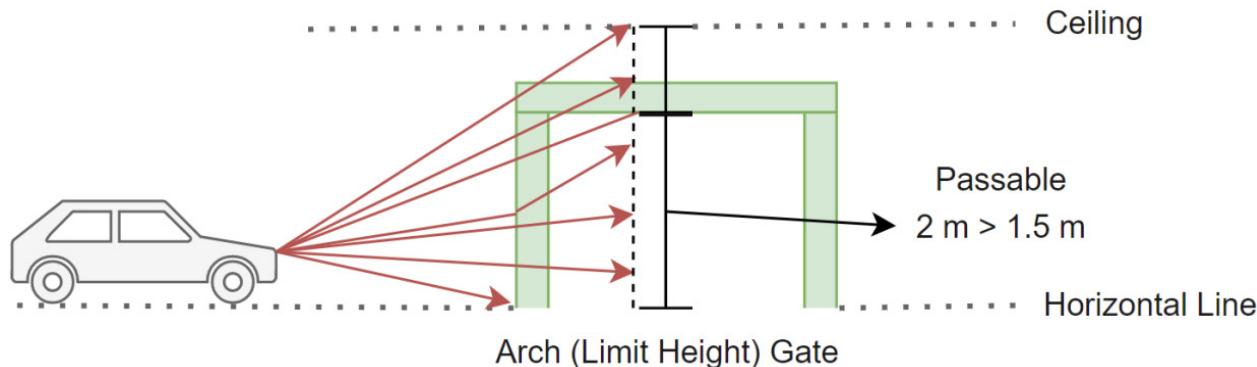
Similarly, by checking the established rules, **Speed Bumps** and **Limited Height Gates** are recognized.

# Speed Bump & Arch Gate Recognition

## Speed Bump Recognition



## Arch Gate Recognition





# 03

## **System Architecture**

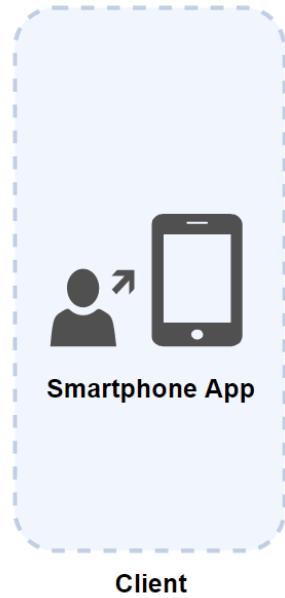
Architecture; Gazebo Setup;  
Simulations and Octomap; Protocols; Server; App.

# Client-Server-Vehicle (ROS) Architecture

## Client

- Smartphone Android App through which data is sent and received.

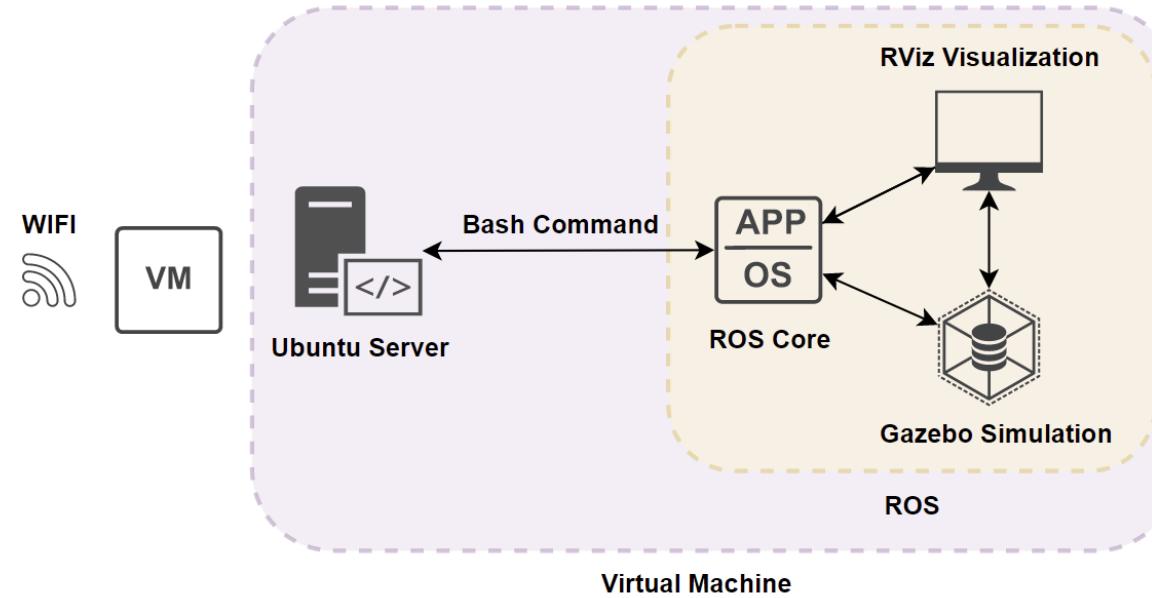
- **Control**
- **Visualization**



## Server

- Ubuntu Server Written in Kotlin to execute commands from cellphone and send data.

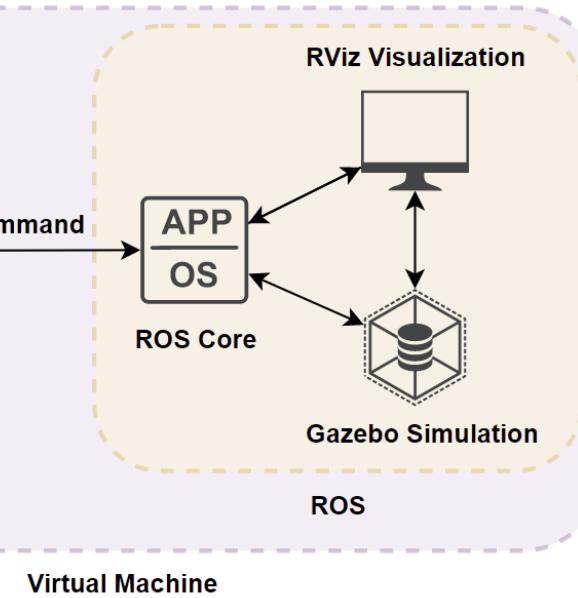
- **Connection**
- **Data Resolving**



## Simulation

- Two scenarios in Gazebo, Low-price 3D LiDAR simulation

- **Vehicle Simulation**
- **LiDAR Verification**



## Visualization

- Octomap Visualization of Point Cloud Data, Path and Obstacles Display

- **Octomap View**
- **Information View**

# ROS – Gazebo Setup



Real-time Obstacle Detection



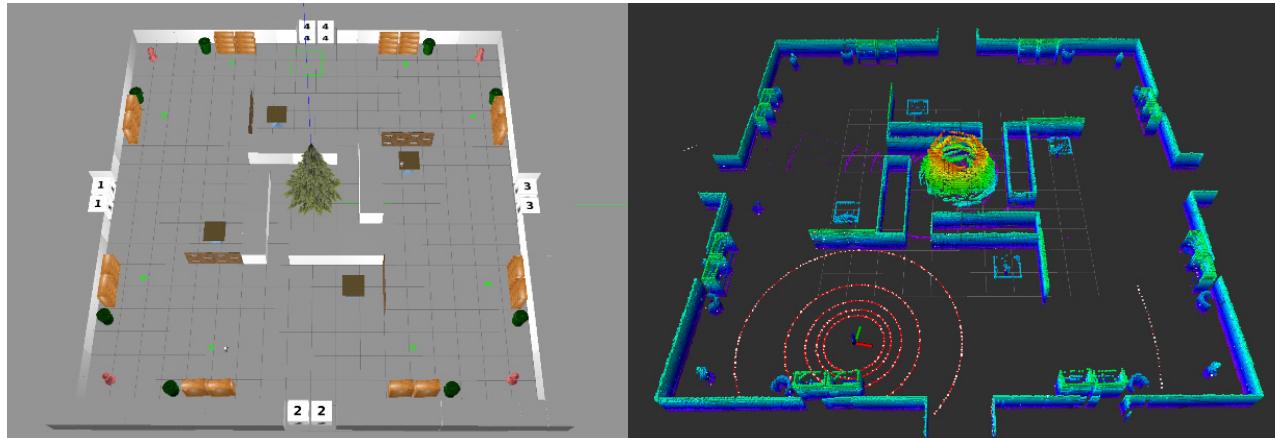
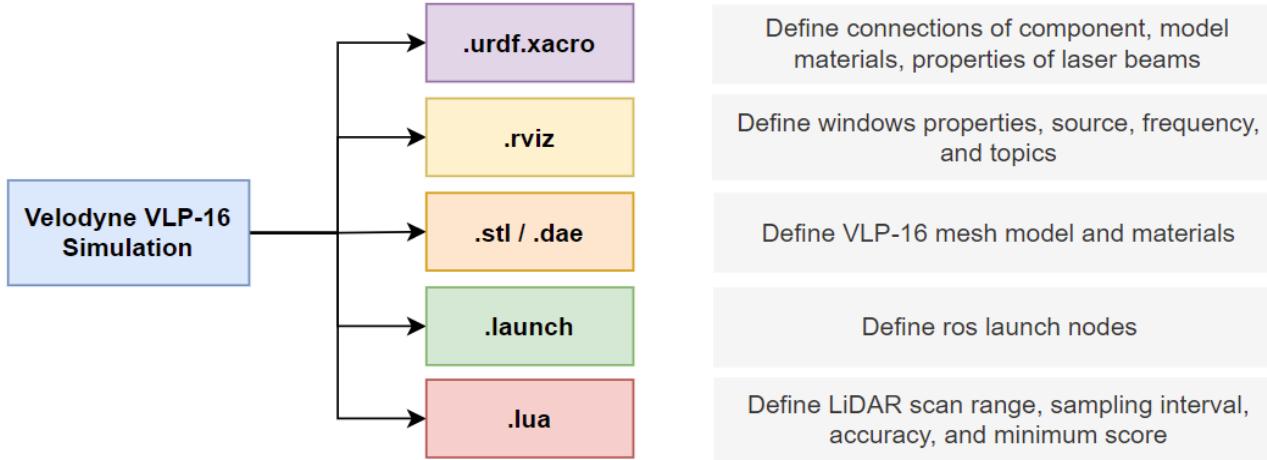
3D Dynamic Path Planning



Speed Bumps & Arch Gates Recognition

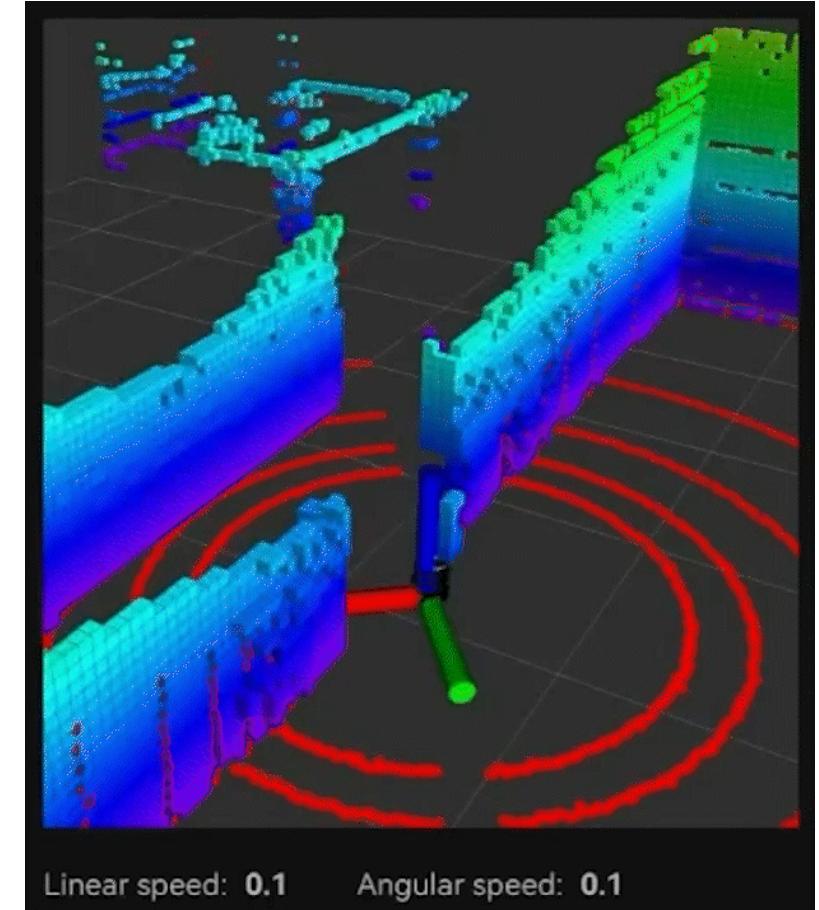
# ROS – LiDAR Simulation & Octomap

## Velodyne LiDAR Simulation



Left: Real-world scene. Right: Map building result.

## Result Visualization (Octomap)



# Middleware – Server & Protocols



c | 0 | 0 | 0

Fix Instruction

t | 2 | 2 | \$ | 7

Data Instruction

n | [ | 5 | m | ...

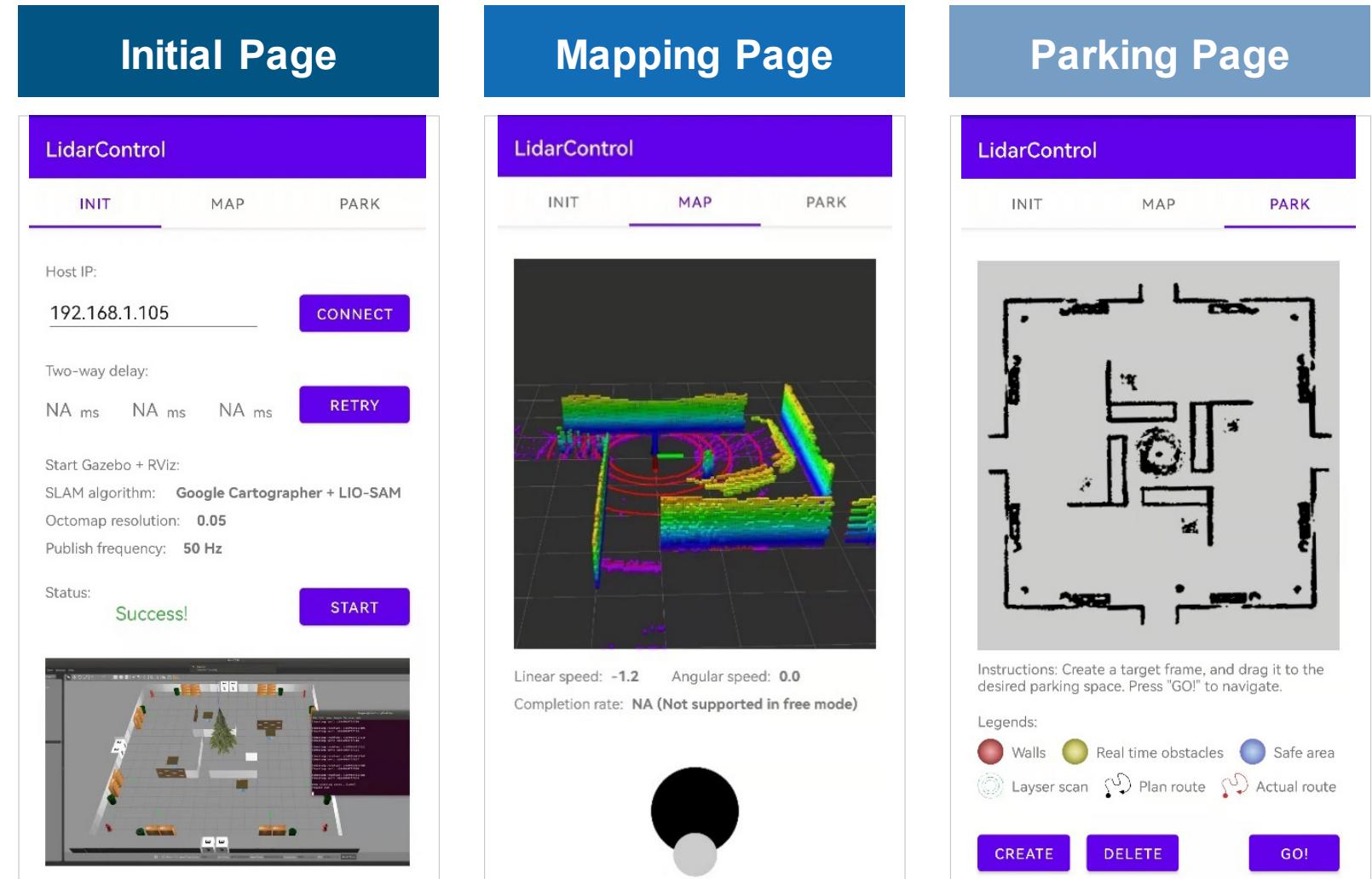
Image & Video Instruction

Inst.	E.g.	Sender	Type	Data	Explanations
X***	x16490 8728...	Client	Test delay	Standard timestamp (longlong)	Test the two-way connection delay
C000	c000	Client	Start core	NAN	Start Gazebo and RViz simulations
P***	pae:[se !45a...	Server	Preview	Image (bitmap)	Send screenshot of image in bit streams
T***	t22\$7	Server	Robot info	Linear \$ Angular velocity (int\$int)	Send current linear and angular velocity
.....	.....	.....	.....	.....	.....

## Instruction & Bash Mapping

Inst.	Bash Command	Explanation
C000	#!/usr/bin/env bash # omitted below Source ~/velodyne_ws/devel/setup.bash roslaunch velodyne_slam_simulation gazebo_turtlebot3.launch	Start Gazebo and RViz
P000	wmctrl -a Gazebo scrot -b -z -o -q 75 'gazebo_rviz_preview.jpg' mogrify -resize 50% gazebo_rviz_preview.jpg	Take screenshots or video
T*	source ~/.profile target_term -run 1 roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch target_term -run 1 \$1	Input control instructions in specific command line
.....	.....	.....

# Client – Android App Design





# 04

## **Result Conclusion**

Requirements;

Scenario 1: Outdoor Parking Lot; Scenario 2; Conclusion

# Prerequisites & Dependencies



## Software Versions

Item	Platform / Software	Version
Virtual Machine	VMware® Workstation 16 Pro	16.2.2 build-19200509
Virtual Computer	Ubuntu	18.04.6 LTS
	ROS	Melodic 1.15.8

## Hardware Properties

Device	Hardware	Specification
Host Computer	CPU: Intel® Core™ i7-11800H	Cores: 8, Processors: 16
	RAM: Ramaxel Technology 32GB	Type: DDR4, 3200MHz
	Graphics Card: RTX 3080 Laptop	Video Memory: 16GB

## Ubuntu Dependencies

### Core (up to date version):

- ros-melodic-desktop-full
- gazebo
- ros-melodic-gazebo-ros-pkgs
- ros-melodic-navigation
- cartographer
- libgtsam-dev
- kotlin

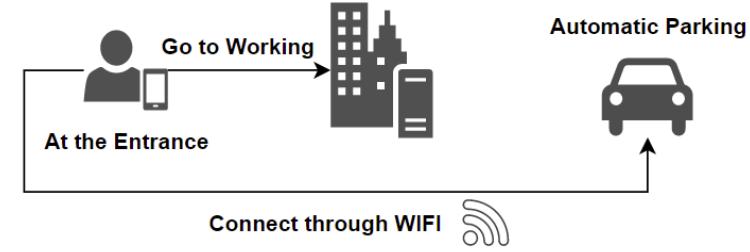
### External Tool:

- wmctrl
- scrot
- mogrify
- xdotool

# Scenario 1: Outdoor Parking Lot



Outdoor scenario



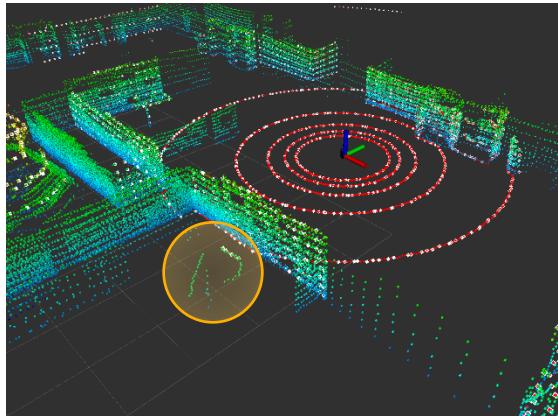
## Difficulties

- A tree with complex details  
**-> Map Building**
- Narrow and dense paths  
**-> Navi Planner Performance**
- Small obstacles, e.g., tables, fenders, fire hydrants, trash cans  
**-> Obstacles Avoidance**

# Scenario 1: Outdoor Parking Lot



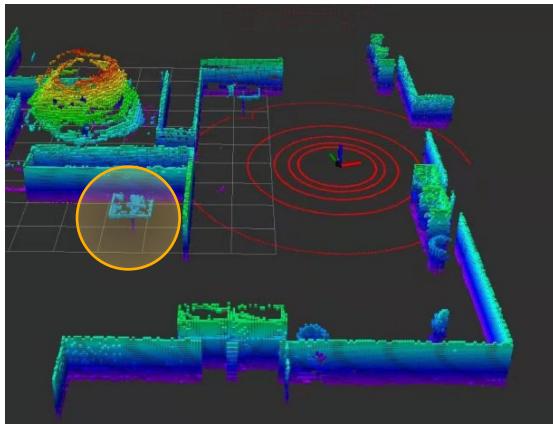
## Visualization of Mapping and Navigation



### Cartographer

#### Poor Table Recognition

The walls are sparse, and misaligned. The OGM map generated by map building result cannot be used to navigation.



### SLAM+

#### Precise Object Detection

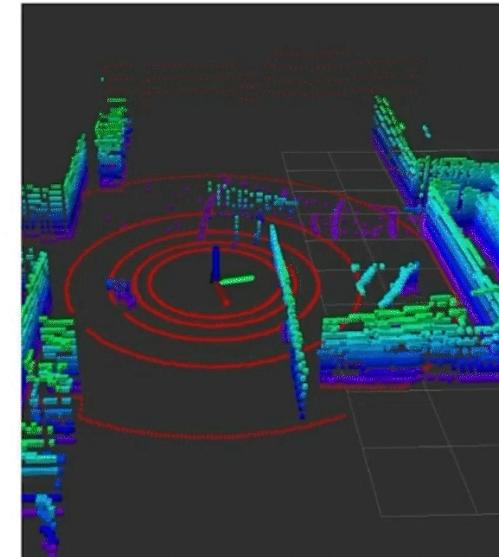
The shape of tables are well restored. All walls and other obstacles have dense point cloud data. The whole result is clean and fine.

### LidarControl

INIT

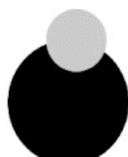
MAP

PARK



Linear speed: 0.3    Angular speed: 0.0

Completion rate: NA (Not supported in free mode)



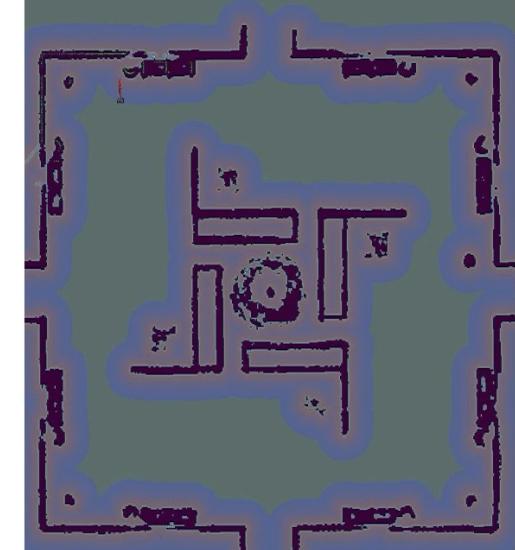
Left: Mapping, Right: Navigation

### LidarControl

INIT

MAP

PARK



Instructions: Create a target frame, and drag it to the desired parking space. Press "GO!" to navigate.

Legends:

- Walls
- Real time obstacles
- Safe area
- Laser scan
- Plan route
- Actual route

CREATE

DELETE

GO!

Top: Cartographer, Bottom: SLAM+

# Scenario 1: Outdoor Parking Lot

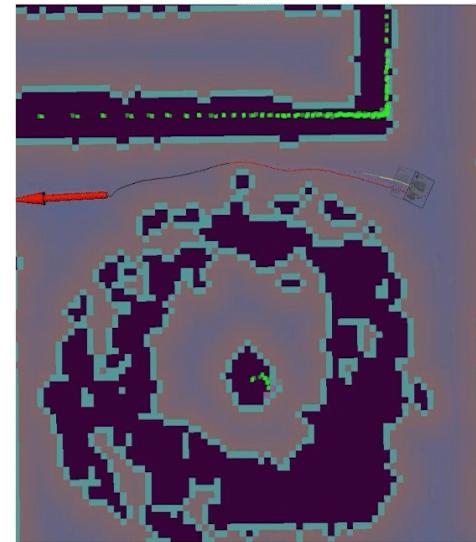
Real-time Dynamic Planning



Top: Real-time object detection, Bottom: Dynamic planning

LidarControl

INIT MAP PARK



Instructions: Create a target frame, and drag it to the desired parking space. Press "GO!" to navigate.

Legends:

- Walls
- Real time obstacles
- Safe area
- Layer scan
- Plan route
- Actual route

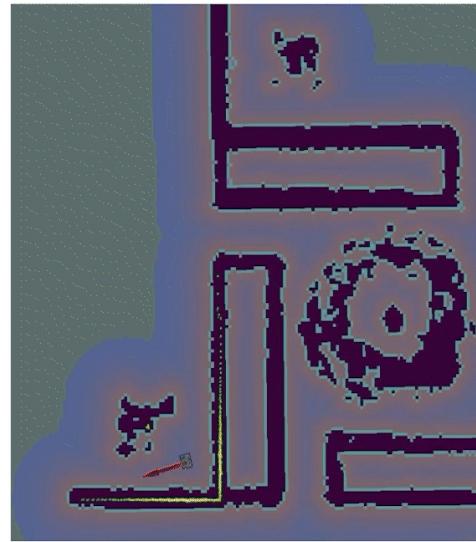
CREATE

DELETE

GO!

LidarControl

INIT MAP PARK



Instructions: Create a target frame, and drag it to the desired parking space. Press "GO!" to navigate.

Legends:

- Walls
- Real time obstacles
- Safe area
- Layer scan
- Plan route
- Actual route

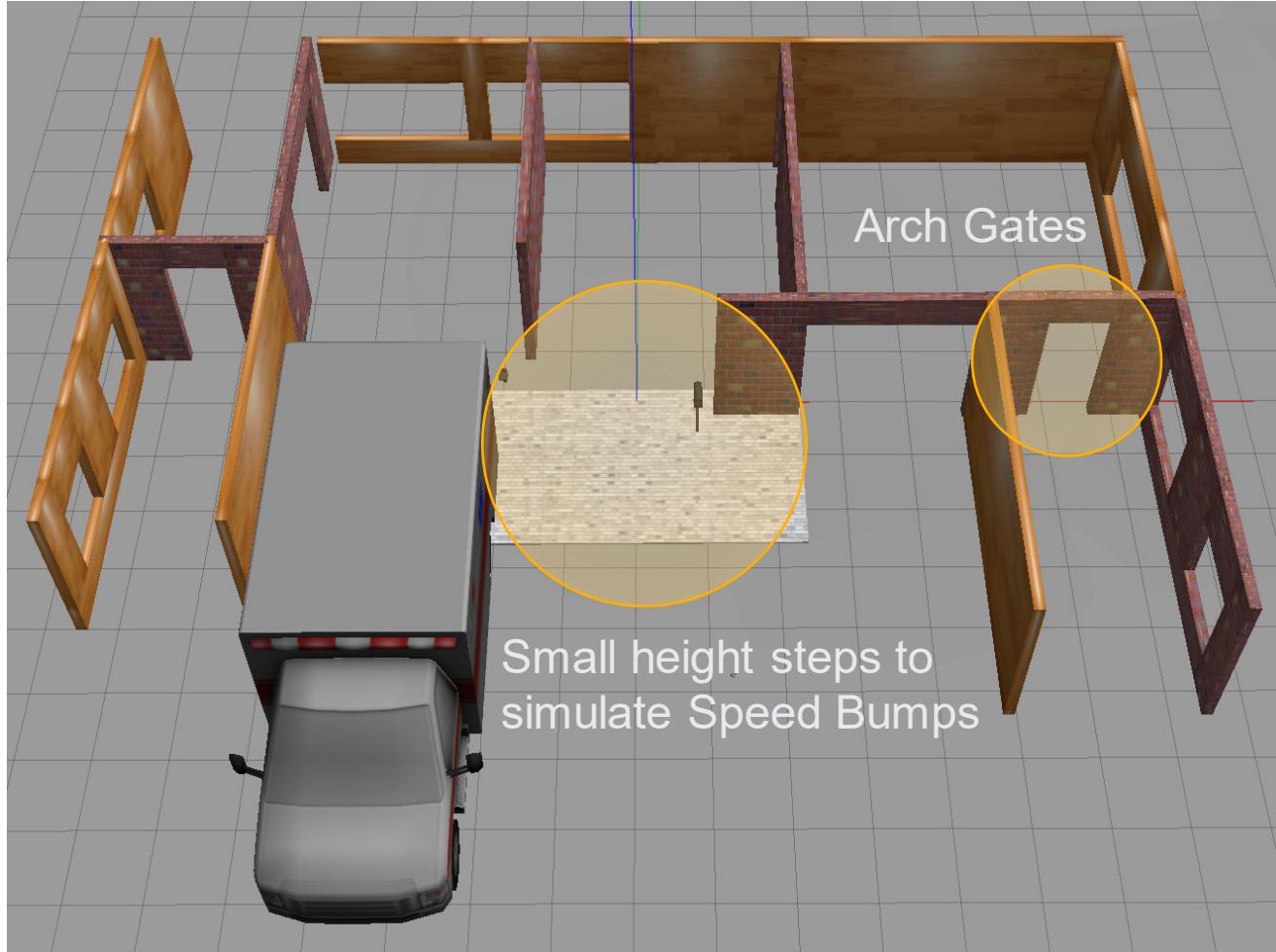
CREATE

DELETE

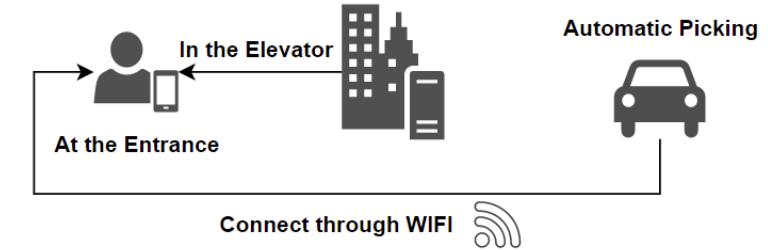
GO!

Left & Right: Dynamic planning preview

# Scenario 2: Indoor Parking Lot



Indoor scenario



## Difficulties

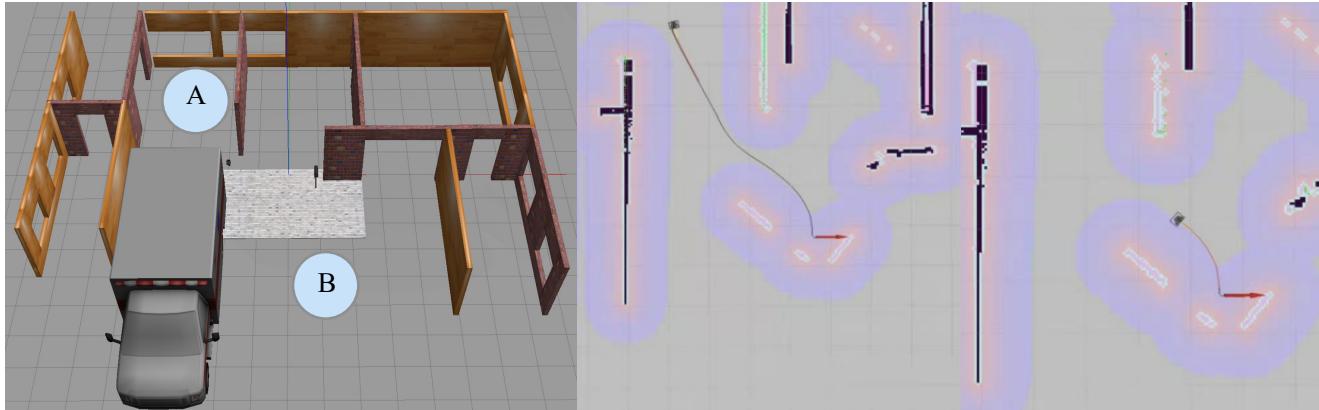
- Multiple paths  
-> **Optimal Path Test**
- Speed Bumps (white steps)  
-> **Speed Bumps Passable**
- Limited-height Gates  
-> **Arch Gates Passable & Hang-over Objects Detection**

# Scenario 2: Indoor Parking Lot

Speed Bumps Passable



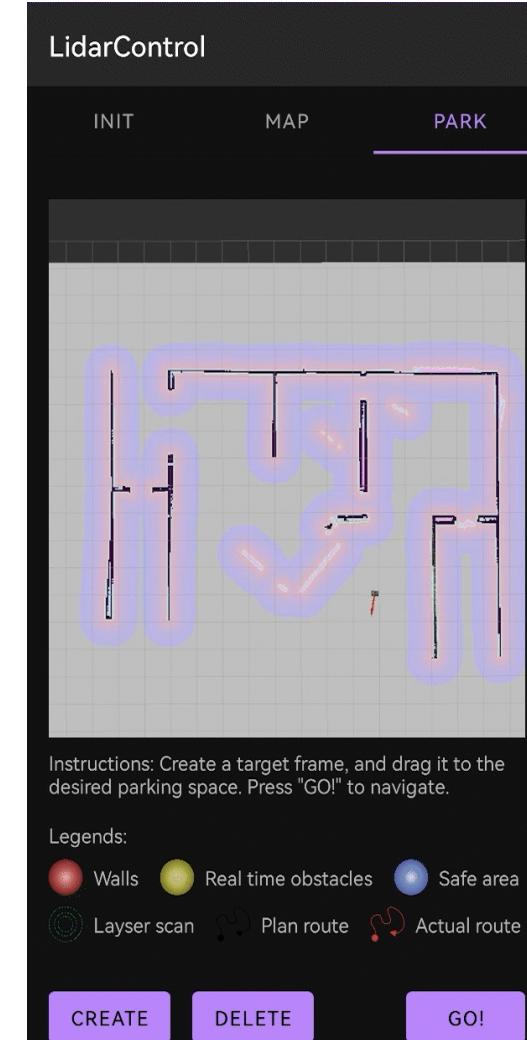
Speed Bumps  
Simulation



Top: Speed bumps simulation. Bottom: Speed bump passable test

LidarControl

INIT MAP PARK



Instructions: Create a target frame, and drag it to the desired parking space. Press "GO!" to navigate.

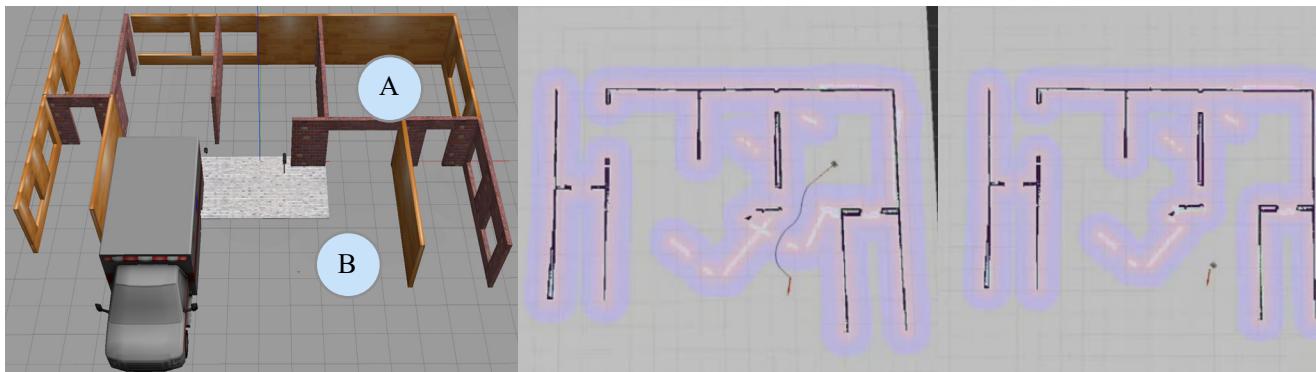
Legends:

- Walls
- Real time obstacles
- Safe area
- Layser scan
- Plan route
- Actual route

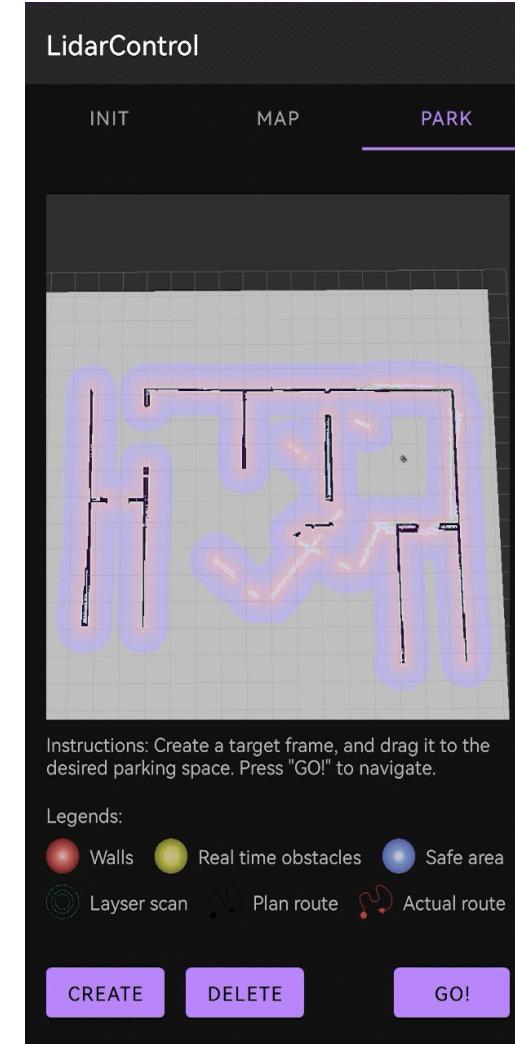
CREATE DELETE GO!

# Scenario 2: Indoor Parking Lot

Limited Height Gates Passable



Top: Limited-height gates simulation. Bottom: Limited-height gates passable test



# Conclusion – Analysis

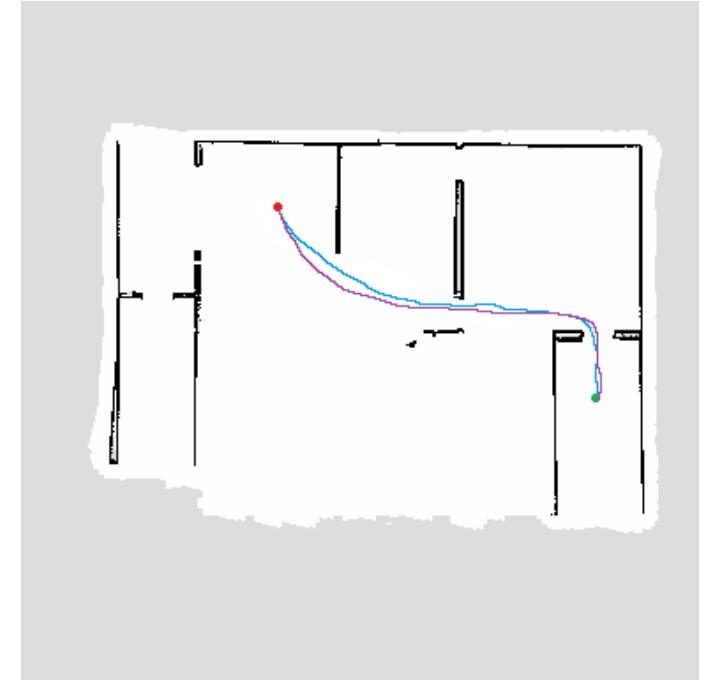
## Mapping Time

Algorithm	Scenario 1: Outdoor (s)	Scenario 2: Indoor (s)
Cartographer	487	239
SLAM+	566	258

## Navigation Performance

Algorithm	Simple Path Navi. Time	Complex Path Navi. Time	Far Dest. Navi. Time	Far Dest. Calc. Time	Dest. in Obstacles
Global (ROS build-in)	11.25	26.34	45.48	0.69	Report failure
Carrot	10.97	Failed	Failed	NAN	Approach point
DStar	26.77	Failed	Failed	NAN	Calculate, and report failure
RAStar	12.06	17.93	48.63	0.05	Report failure
RRT+	12.48	19.65	44.34	0.28	Trapped in loop

## Path Comparison



Red Point: Start; Green Point: End; Blue Path: RRT+; Purple Path: ROS build-in.

**Shorter and smoother path in complex environment**

# Conclusion & Future Work



## My Work

- Scenarios built up, VLP-16 3D LiDAR simulation
- SLAM+: Improved 3D map building and visualization
- RRT+: Optimized 3D real-time dynamic navigation
- Ubuntu server, Android control application



## Pros. & Cons.

- Remote control and visualization via App
- Fine map building with inexpensive 3D LiDAR
- Real-time dynamic 3D navigation
- Not employed due to hardware shortage
- Low video streaming efficiency; limited LAN range.

INIT      MAP      PARK

Host IP:

192.168.1.105

CONNECT

Two-way delay:

NA ms    NA ms    NA ms

RETRY

Start Gazebo + RViz:

SLAM algorithm: Google Cartographer + LIO-SAM

Octomap resolution: 0.05

Publish frequency: 50 Hz

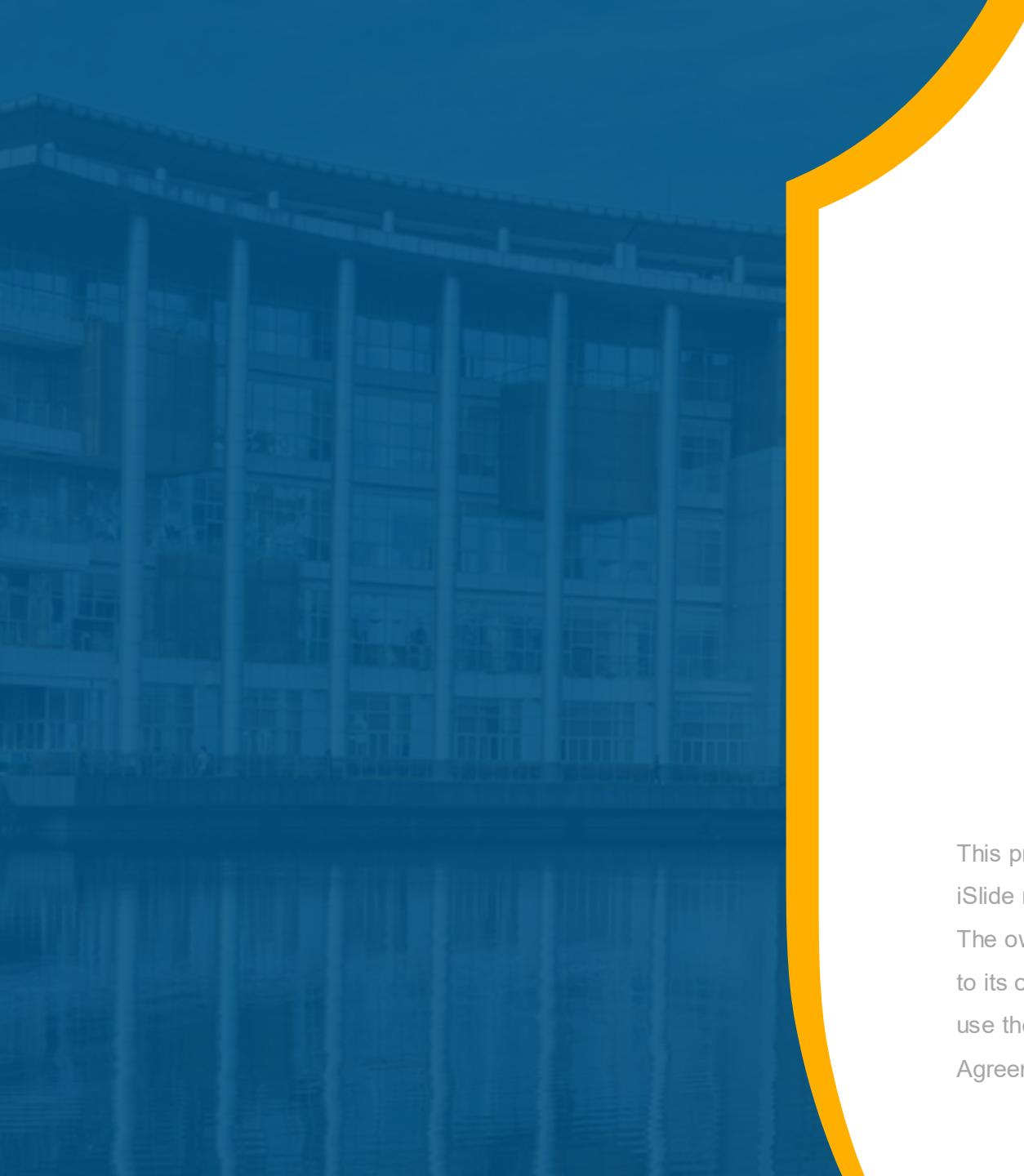
Status:

Not Connect!

START



IMAGE NOT FOUND



# Thanks for Watching!

Hu, Yadong

el18hh@leeds.ac.uk

This presentation file is supplied by iSlide.

iSlide respects all intellectual property rights and protects all the rights its users acquired.

The ownership and intellectual property of any content in this PowerPoint presentation belongs to its owner or the assignee of this ownership. What you purchased or got for free is the right to use the content. The usage should be subject to the terms of "Copyright Notice", "User License Agreement", "Premium Agreement" and more Agreements of iSlide official website.

# ROS – Octomap Visualization



```
<node pkg="octomap_server" type="octomap_server_node" name="octomap_server">
  // Define the resolution of a cube (Velodyne point). Too high results in low speed.
  <param name="resolution" value="0.05" />
  // Frame defined in rviz file. Odom for 3D visualization and map for 2D.
  <param name="frame_id" type="string" value="odom" />
  // Define the maximum detection range of the sensor. See LiDAR manual.
  <param name="sensor_model/max_range" value="8.0" />
  // Does not show the ground.
  <param name="filter_ground" value="true" />
  // Cast point cloud data into Velodyne points
  <remap from="cloud_in" to="/velodyne_points" />
</node>
```

# RRT – Height Judgement



```
// ObstacleFree returns 1:free space, -1:unkown (frontier region), and 0:obstacle
// Add support for 3d checking
Char checking = ObstacleFree(x_nearest, x_new, z_nearest, z_new, mapData);
if (checking == -1) {
    // Obtain points from each frame
    exploration_goal.header.stamp = ros::Time(0);
    exploration_goal.header.frame_id = mapData.header.frame_id;
    exploration_goal.point.x = x_new[0];
    exploration_goal.point.y = x_new[1];
    // Original code: exploration_goal.point.z = 0, add z-level obstacles checking
    exploration_goal.point.z = z_new[0];
    p.x = x_new[0];
    p.y = x_new[1];
    p.z = z_new[0];
    points.points.push_back(p);
    pub.publish(points);
    // Each time recalculate the path
    targetspub.publish(exploration_goal);
    points.points.clear();
    V.clear();
    .....
    x_new[0] = transform.getOrigin().x();
    x_new[1] = transform.getOrigin().y();
    z_new[0] = transform.getOrigin().z();
    V.push_back(x_new);
    line.points.clear();
} else if (checking == 1){
    V.push_back(x_new);
    p.x = x_new[0];
    p.y = x_new[1];
    p.z = y_new[0]
    line.points.push_back(p);
    p.x = x_nearest[0];
    p.y = x_nearest[1];
    p.z = z_nearest[0];
    line.points.push_back(p);
}
pub.publish(line);
```

# ROS – Launch File



```
<launch>

    <!-- Arguments -->

    <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle,
waffle_pi]"/>

    <arg name="map_file" default="$HOME/map.yaml"/>
    <arg name="open_rviz" default="true"/>
    <arg name="move_forward_only" default="false"/>

    <!-- Turtlebot3 -->
    <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
        <arg name="model" value="$(arg model)" />
    </include>

    <!-- Map server -->
    <node pkg="map_server" name="map_server" type="map_server" args="$(arg map_file)"/>
```

# ROS – Launch File Continued



```
<!-- AMCL -->  
  
<include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>  
  
<!-- move_base -->  
  
<include file="$(find turtlebot3_navigation)/launch/move_base.launch">  
  <arg name="model" value="$(arg model)" />  
  <arg name="move_forward_only" value="$(arg move_forward_only)"/>  
</include>  
  
<!-- rviz -->  
  
<group if="$(arg open_rviz)">  
  <node pkg="rviz" type="rviz" name="rviz3" required="true"  
    args="-d $(find turtlebot3_navigation)/rviz/turtlebot3_navigation.rviz"/>  
</group>  
</launch>
```

# Middleware – Web Socket



```
// From main()
Thread { server1() }.start()
.....
fun server1() {
    // Accept the connection on port 7230
    val server = ServerSocket(7230)
    val client = server.accept()
    while (true) {
        val output = PrintWriter(client.getOutputStream(), true)
        val input = BufferedReader(InputStreamReader(client.getInputStream()))
        val inputText = input.readLine()
        if (inputText == null || inputText == "") continue
        // Decode instructions by the first character
        when (inputText[0]) {
            'x' -> { ..... }
            'c' -> { ..... }
            .....
        }
    }
}
```

# Middleware – Web Socket Continued



```
private fun socketConnectThread() {  
    object : Thread() {  
        override fun run() {  
            try {  
                // Obtain the IP address from user input, and specify the port  
                listerSocket = Socket(ip.text.toString(), uPORT1)  
                val output = PrintWriter(listerSocket!!.getOutputStream(), true)  
                val input = BufferedReader(InputStreamReader(listerSocket!!.inputStream))  
                // Keep sending data if the connection is not closed, and the data is not null  
                while (!closeSocket) {  
                    if (textBufferWrite1 != "") {  
                        when (streamType) {  
                            0 -> {  
                                try {  
                                    // Read data from the server  
                                    textBufferRead1 = input.readLine()  
                                    if (textBufferRead1 != "") {  
                                        // If read non-null data, then decode data  
                                        decodeMessage(textBufferRead1)  
                                    }  
                                } catch (e: Exception) { ..... }  
                            }  
                            // start cores, and streaming preview images  
                            1, 2 -> { ..... }  
                            else -> { ..... }  
                        .....  
                    }  
                }  
            } catch (e: Exception) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```



# Client – Android App

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help LidarControl - fragment\_main.xml [LidarControl.app]

Android Projects > app > src > main > res > layout > fragment\_main.xml

PlaceholderFragment.kt fragment\_main.xml colors.xml SocketRequest.kt JoystickView.kt

Project Resource Manager Favorites Build Variants Structure

**Palette**: Common, Ab TextView, Text, Buttons, Widgets, Layouts, Containers, Helpers, Google, Legacy, Project.

**Component Tree**: ConstraintLayout, constraintLayout1, start "Start", connect "Conn...", Ab ip "192.168.1.105", retry "Retry", Ab textView "Host ...", Ab textView2 "Tw...", Ab textView17 "St...", Ab textView9 "Sta...", Ab textView10 "SL...", Ab textView13 "G...", Ab textView15 "O...", Ab textView16 "5...", Ab textView11 "O...", Ab textView12 "...", Ab textView4 "NA".

**Attributes**: id, Declared Attributes, Layout, visibility, transforms, Rotation, rotation, rotationX, rotationY, scaleX, scaleY, translationX, translationY, translationZ.

**Code Split Design**

**Instructions:** Create a target frame, and drag it to the desired parking space. Press "GO!" to navigate.

**Legends:** Walls (Red circle), Real time obstacles (Yellow circle), Safe area (Blue circle), Layer scan (Green circle), Plan route (Blue arrow), Actual route (Red arrow).

**Buttons:** CREATE, DELETE, GO!, 1:1, Device File Explorer, Emulator.

androidx.constraintlayout.widget.ConstraintLayout > androidx.constraintlayout.widget.ConstraintLayout > ImageView

Run TODO Problems Terminal Build Profiler App Inspection Event Log Layout Inspector 3 497:59 LF UTF-8 4 spaces Launch succeeded (46 minutes ago)

# **Client – Android App**



The screenshot shows the Android Studio interface with the following details:

- Top Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help, LidarControl - PlaceholderFragment.kt [LidarControl.app]
- Project Tree (Left):** Shows the project structure under "Android Projects > app > src > main > java > com > hyd > lidarcontrol > ui > main > PlaceholderFragment". The "PlaceholderFragment" file is selected.
- Code Editor (Right):** Displays the PlaceholderFragment.kt code. The code handles socket communication to receive byte arrays and update UI threads. It includes logic for different stream types (2, 3, 6, 7) and threads (commThread1, commThread2, commThread3).
- Bottom Bar:** Run, TODO, Problems, Terminal, Logcat, Build, Profiler, App Inspection, Event Log, Layout Inspector.

```
938
939
940    init {
941        try {
942            val `in` = clientSocket.getInputStream()
943            input = DataInputStream(`in`)
944        } catch (e: IOException) {
945            e.printStackTrace()
946        }
947    }
948
949
950    inner class UpdateUIThread(private val byteArray: ByteArray) : Runnable {
951        override fun run() {
952            val bitmap = BitmapFactory.decodeByteArray(byteArray, 0, byteArray.size)
953            if (streamType == 2) {
954                imgPreview.setImageBitmap(bitmap)
955                Thread(commThread1).interrupt()
956            } else if (streamType == 3) {
957                if(isStreaming) {
958                    mapStream.setImageBitmap(bitmap)
959                } else if(isMapping) {
960                    mappingStream.setImageBitmap(bitmap)
961                }
962                Thread(commThread2).interrupt()
963            } else if (streamType3 == 6 || streamType3 == 7) {
964                mappingStream.setImageBitmap(bitmap)
965                Thread(commThread3).interrupt()
966            }
967        }
968    }
969
970
971 }
```