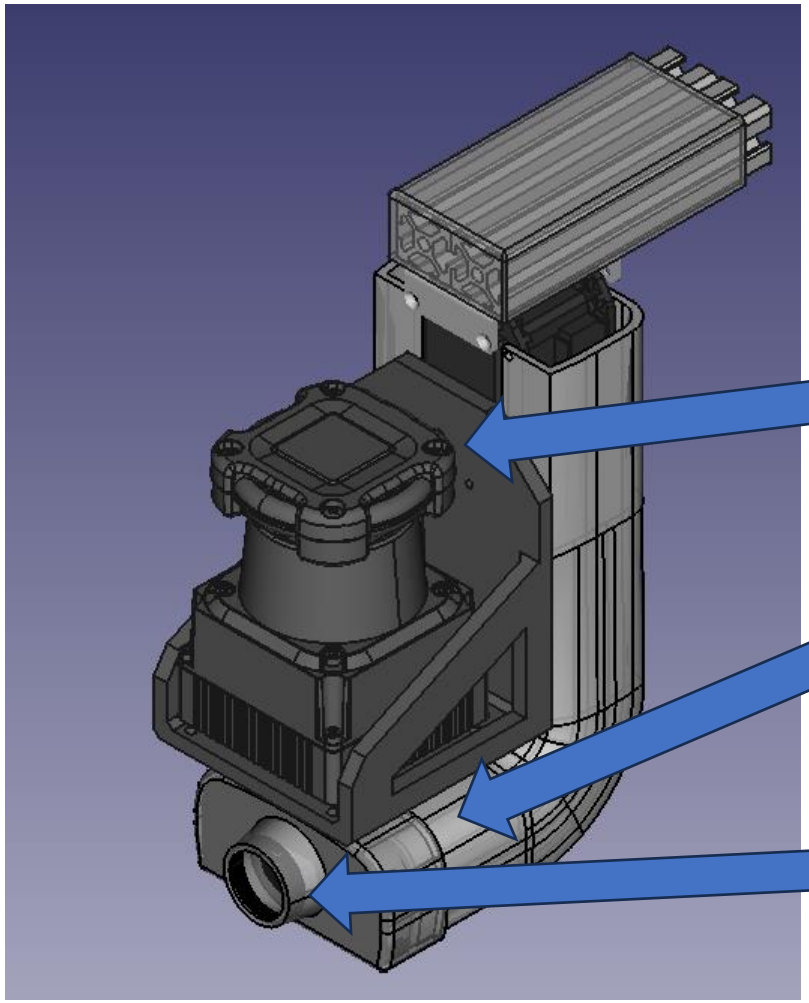
GET Lab

Prof. Dr.-Ing. Bärbel Mertsching
Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik – GET Lab

Disaster Response Robots Project Group

Hardware and Sensors

- Robot uses many different sensors for mapping

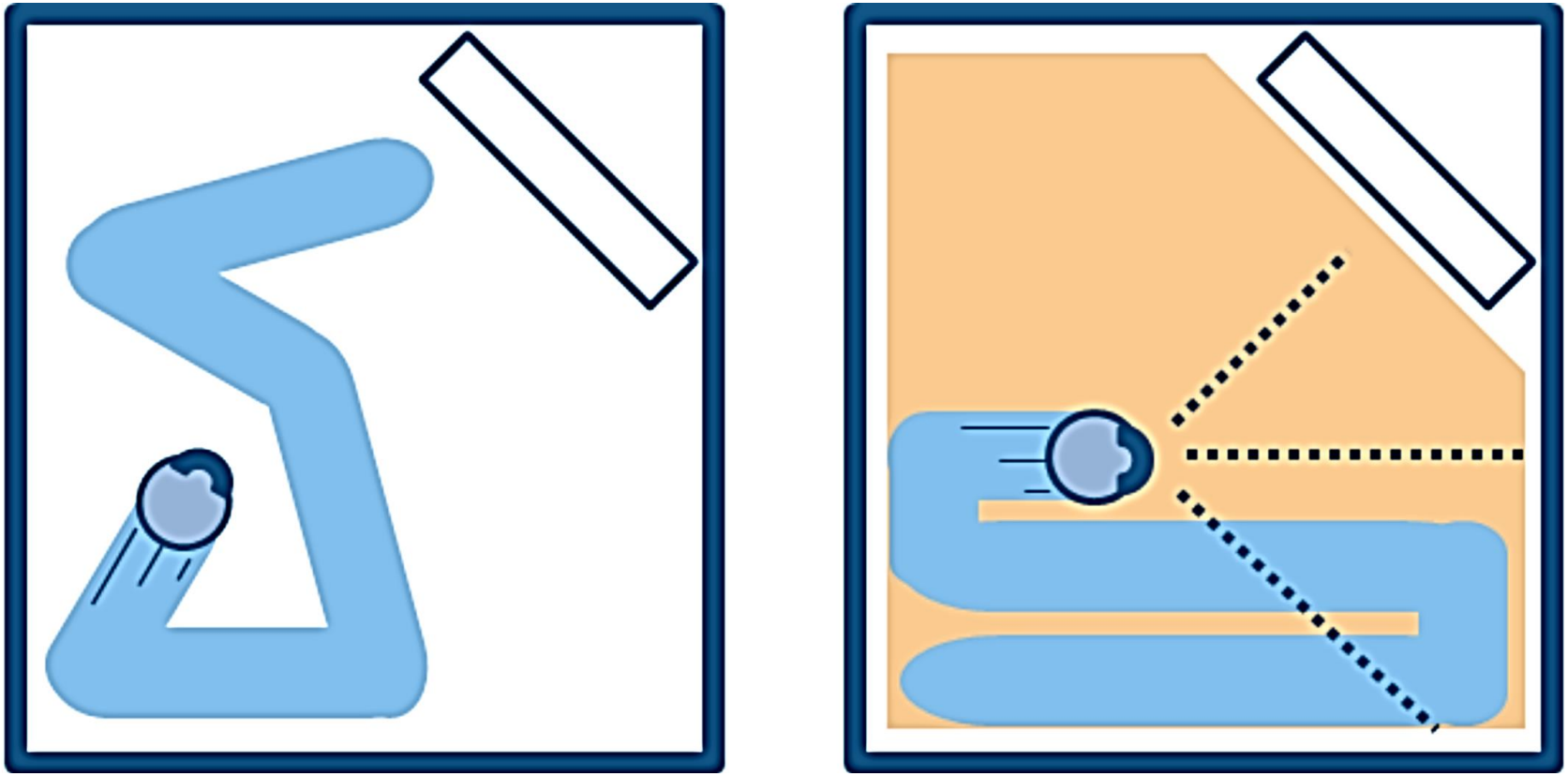


https://getwww.uni-paderborn.de/wiki/msr/RRS_(WS_2022-23)/Hardware/Concept_and_Implementation#/media/File:New_mapping_unit.png

Hardware and Sensors

- Improving the structure of the old calibration
- New IMU pose estimation with Georgia Tech Smoothing and Mapping (GTSAM)
- Restructure old code to allow using of the new pose estimation

Mapping



[https://www.mathworks.com/discovery/slam.html]

GET Lab

Prof. Dr.-Ing. Bärbel Mertsching
Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik – GET Lab

Mapping Subgroup

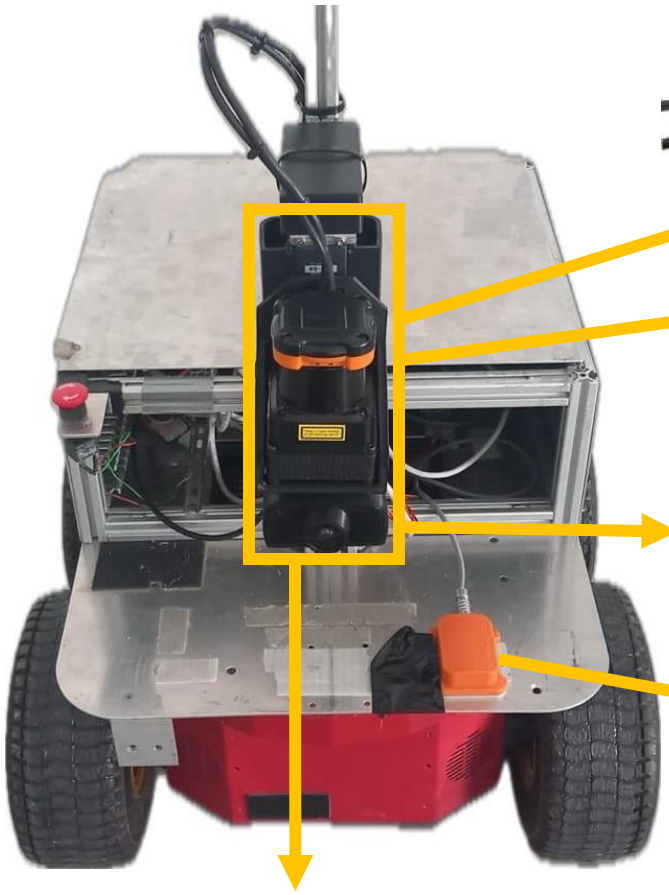
Ekansh Bajpai, Max Pape

Contents

- Introduction
- Fundamentals
- Code Analysis
- Next Best View
- Summary
- Future Work

Contents

Robot Setup



Scanning Unit

Camera

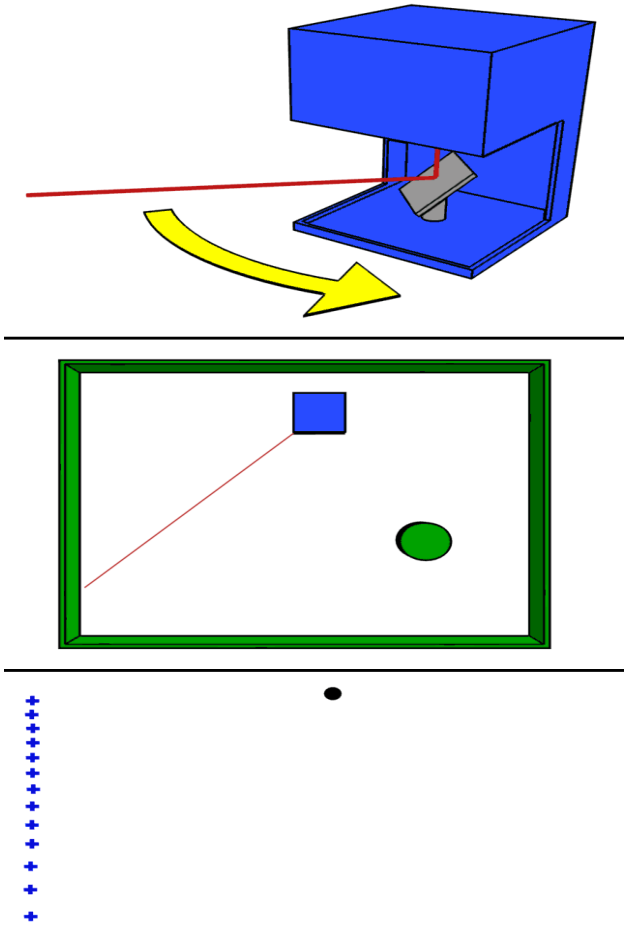
Hokuyu-UTM-30LX
Laser Scanner

Dynamixel MX-64
Servo Motor

XSens Mti IMU

Data Acquisition: LIDAR

Working of LIDAR



[https://en.wikipedia.org/wiki/Lidar]

Data Acquisition

Data Acquisition

Lidar Odometry and Mapping (LOAM)

LOAM: Lidar Odometry and Mapping in Real-time

[Zhang et. al., 2014]

Contents

Why Code Analysis

- Loam algorithm already implemented
- To better understand code implementation of algorithm
- Several enhancements are being made in algorithm
  - Next Best View
  - Loop Closure, etc
- Same Point cloud registration class used in algorithm & calibration
- Better management of such classes for use in enhancements
- Other sub-groups like Hardware group can use same classes

Current Code Analysis

Issues Identified

- Scan data records robot body resulting in false features
- Incomplete motor sweep in simulation environment
- Redundant and non-user-friendly code
- Systematic Documentation missing

Current Code Analysis

Proposed Solutions

- Filtering out the robot body from scan data
- Bridging of gap created by incomplete rotation of motor.
- Code optimisation and restructuring.
- Complete documentation

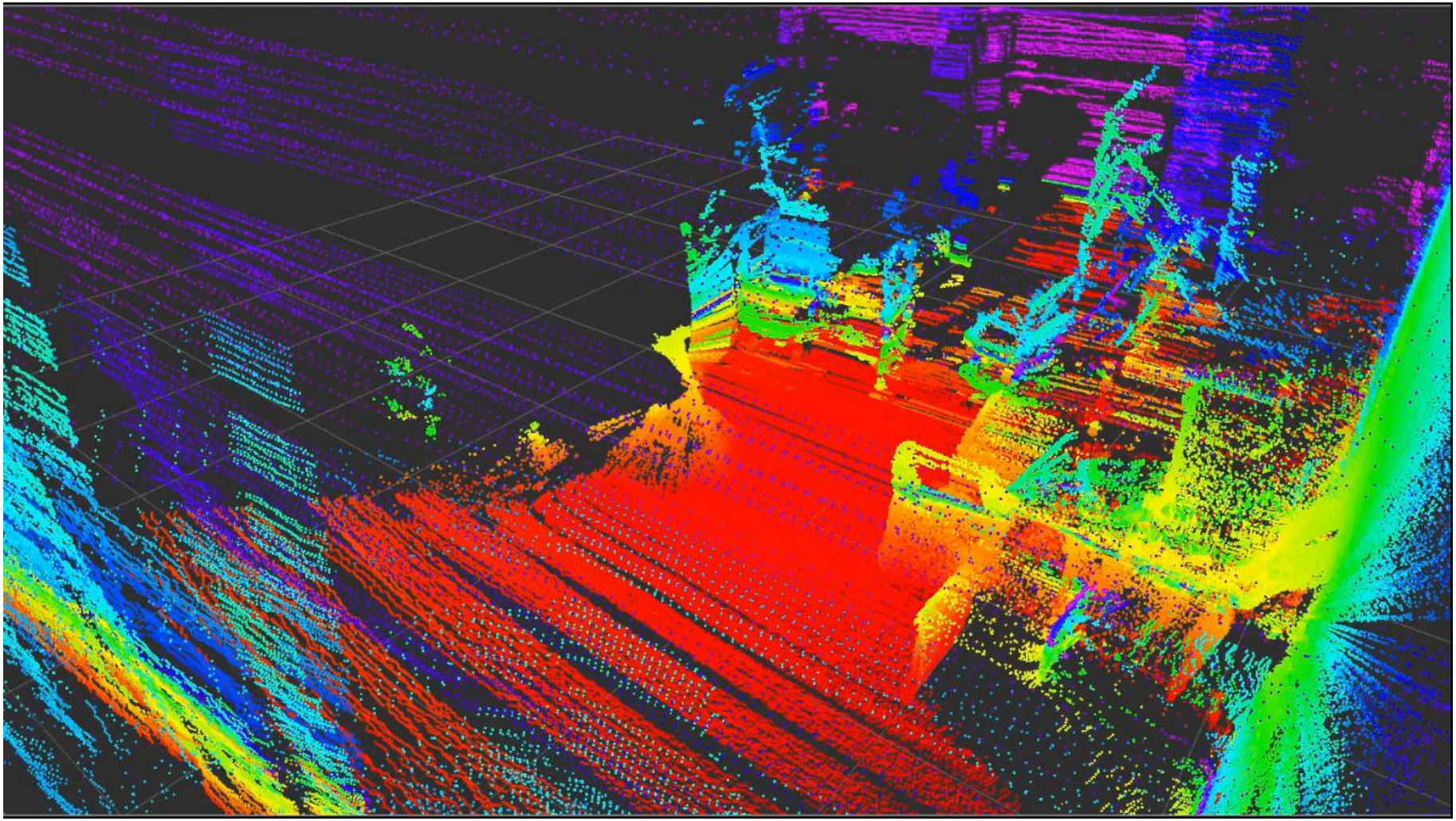
Contents

Bug 1: Robot Body In Laser Scan Data

Analysis

- Planar range of LIDAR is 270^{o}
- Motor moving in +π/2 to -π/2 range
- Part of robot also scanned due to combined movement
- Scan data have points corresponding to robot
- Results in false features recorded
- Data from one robot setup not reliable for another setup

Bug 1: Robot Body In Laser Scan Data



Bug 1: Robot Body In Laser Scan Data

Solution

- Ignore scan points very close to LIDAR.
- Set ranges less than a threshold to infinity.
- Variable Threshold
- Threshold for current robot: 0.6m
- Threshold should be almost equal to minimum clearance area

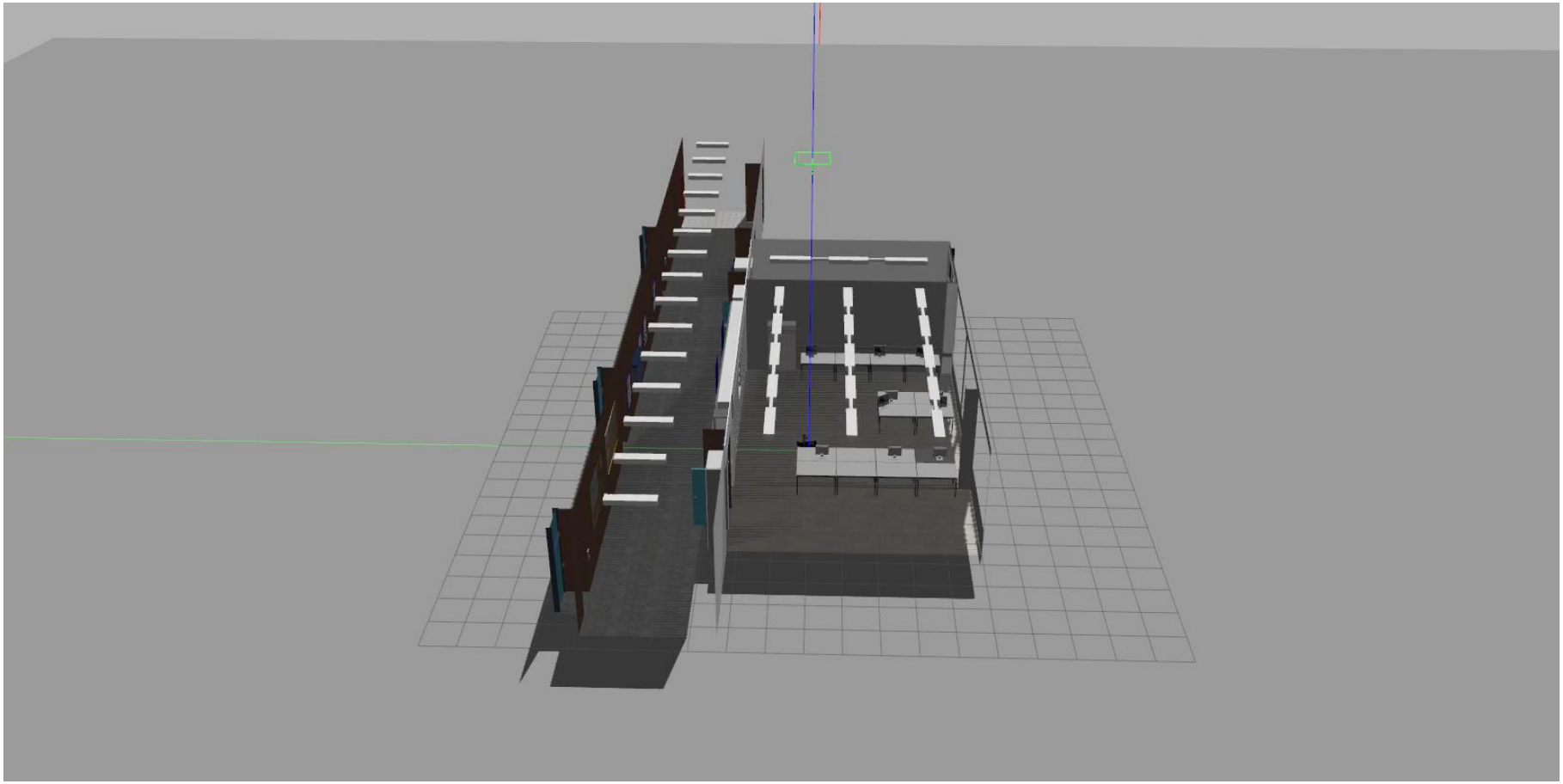
Bug 1: Robot Body In Laser Scan Data

Bug 2: Incomplete motor sweep in Gazebo Env.

Analysis

- ros::Rate() used currently for loop
- Motor goal switched on every loop
- No validation of reaching the goal
- Goal switched before being achieved

Bug 2: Incomplete motor sweep in Gazebo Env.



Scanning Environment

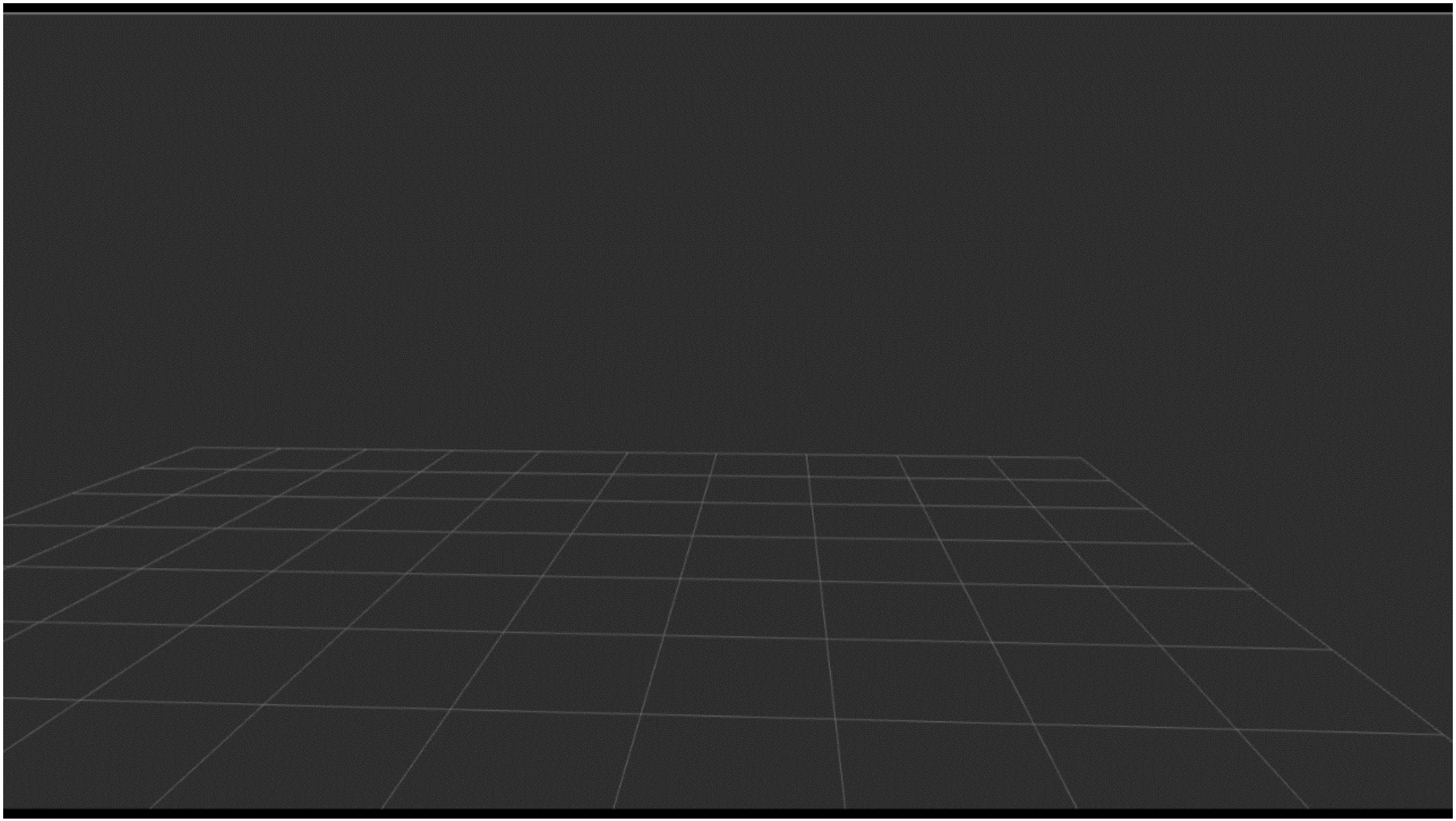
Bug 2: Incomplete motor sweep in Gazebo Env.

Bug 2: Incomplete motor sweep in Gazebo Env.

Solution

- Remove rate dependency on goal change
- Change goal only if previous goal is almost reached
- Difference of current position and goal is taken
- Difference is compared with a threshold value
- Current value for threshold: 0.005

Bug 2: Incomplete motor sweep in Gazebo Env.



Contents

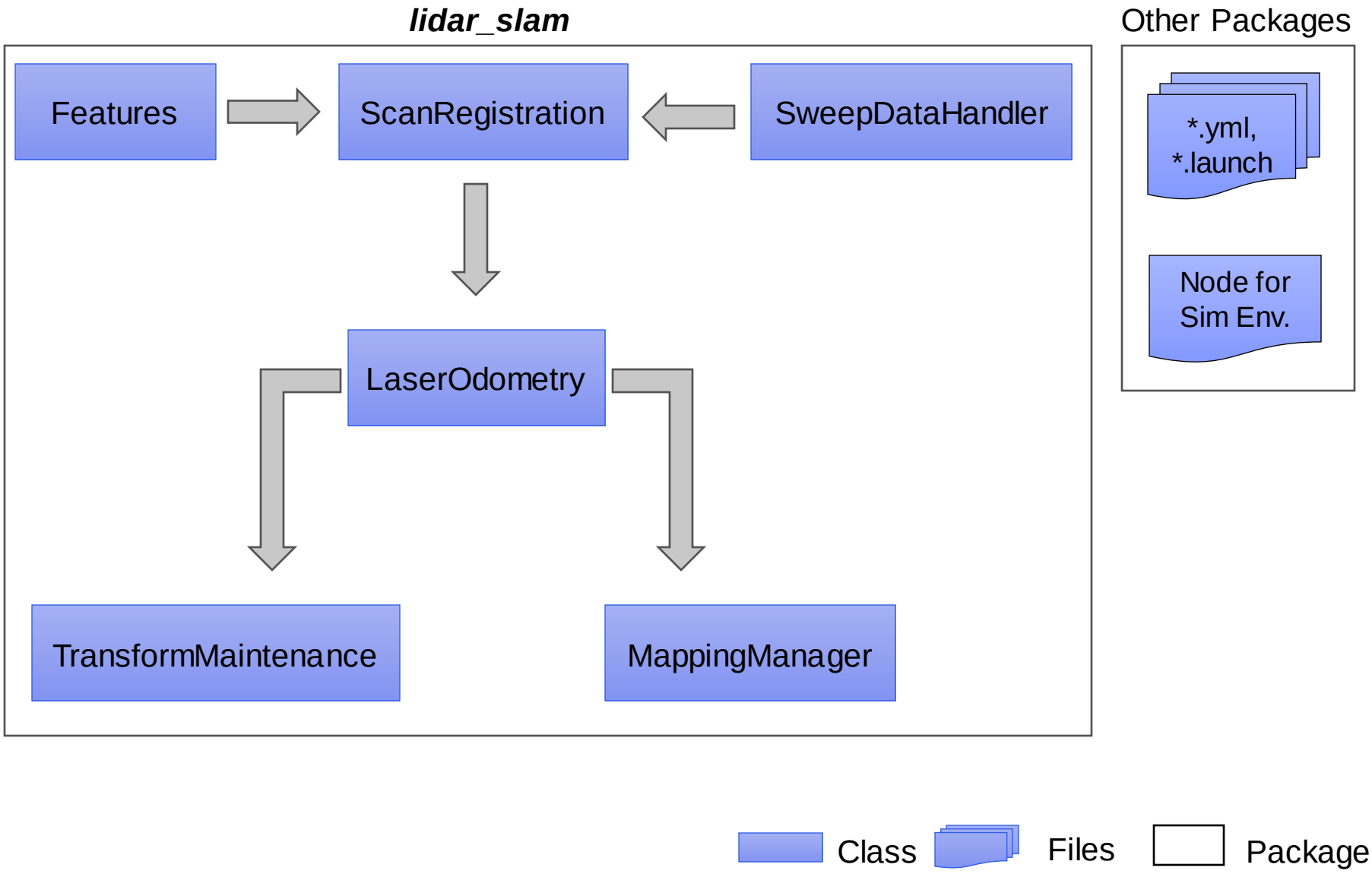
Readability & Usability: Issues

- Same code blocks in multiple files
- Several Nodes/Classes present in package not used
- Following items present in separate packages:
  - Several essential nodes/classes
  - Real/Simulation robot related files e.g *.launch, *.world
  - Configuration and calibration data
- Calibration & configuration data saved in single *.yml file
  - Less flexibility in using different models of sensors/actuators.
  - Little description of data in the file

Readability & Usability: Issues

- Several required class members related to different transducers in single class
  - Less flexible if extra transducers are added.
  - Buffers for *JointState* & *LaserScan* can be used for one device only
- No centralized access to ROS parameters.
  - Multiple changes required if any parameter's name changes.
  - Difficult to keep track of parameter access.
- No Single-responsibility principle present.
  - A module should be responsible to one, and only one, actor

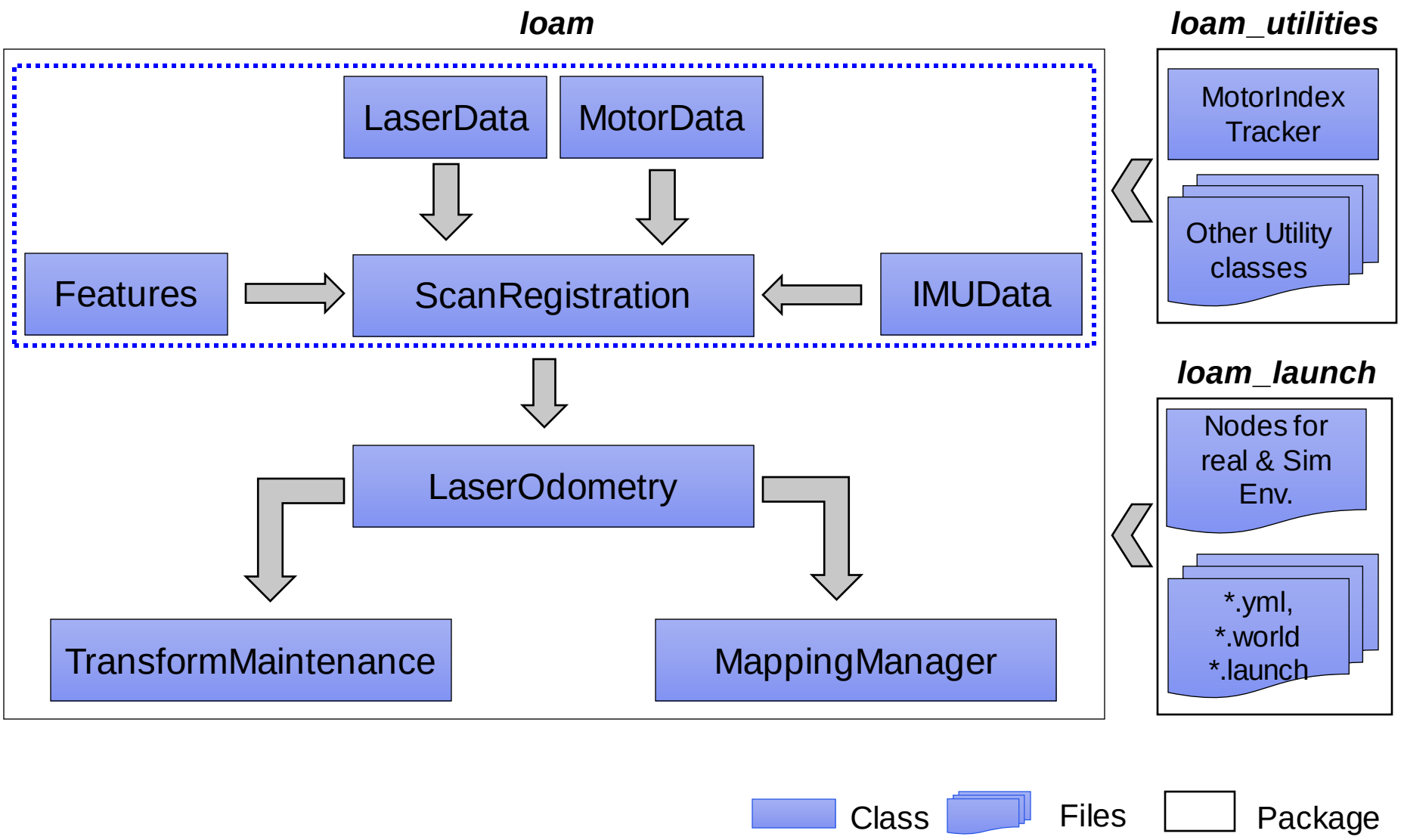
Readability & Usability: Current Code Structure



Readability & Usability: Solutions Implemented

- New code structure follows Single-Responsibility Principle.
- Separate packages for actual algorithm, utility classes & algorithm running nodes.
- Packages are as follows:
  - ***loam***: Contains actual algorithm implementation
  - ***loam_utilities:*** Utility classes not part of algorithm. Can be used indivdually.
  - ***loam_launch***: Classes & nodes for running algorithm & related files such as *.yml, *.launch, *.world.

Readability & Usability: New Code Structure



Package: Loam

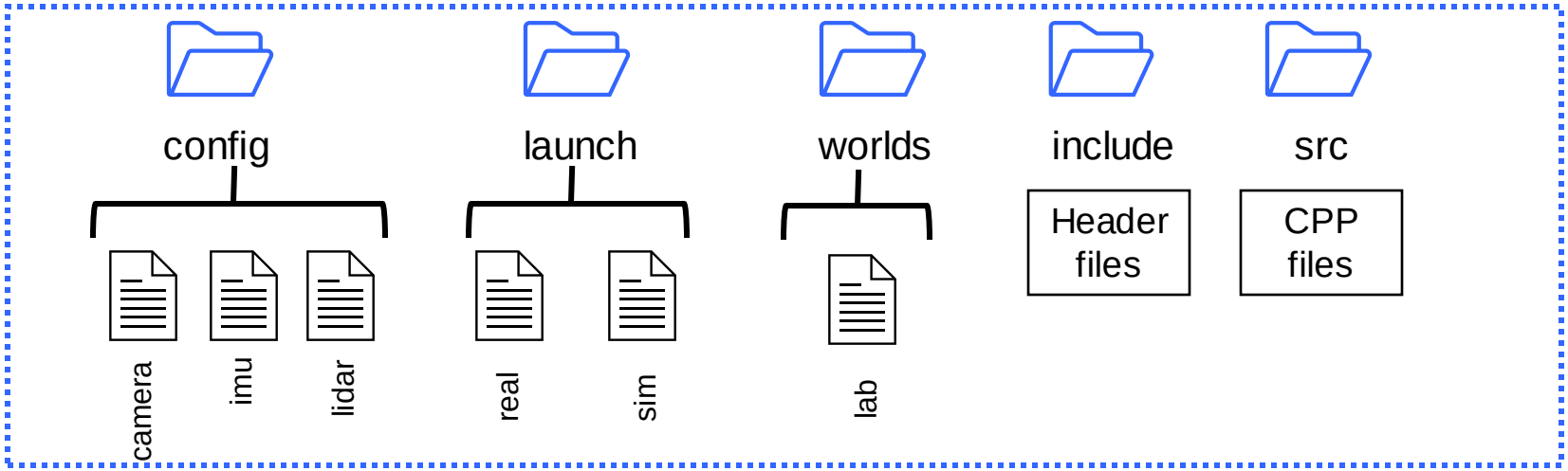
- Package now contains code for core algorithm
- Separate classes for transducers
- Multiple objects of certain transducers type can be added
- Each objects subscribes to its topic separately
- Enables having separate data buffers for each transducer
- Each object can process its data buffer more flexibly

Package: Loam_Utilities

- Exports utility classes as a library
- Utilities not part of algorithm
- Act as support classes for the algorithm
- Can be used in other nodes for different purposes
- Ex. MotorIndexTracker class
  - Tracks index of Motor name in JointState message
  - Provides new index if index of Motor name is changed

Package: Loam_Launch

- Contains files required for running algorithm
- Following type of files present:
  - Launch files to run algorithm real world and simulations
  - Config & calibration files for transducers, etc.
  - Files for Gazebo simulation i.e *.world
  - Nodes and classes for Motor rotation



Parameter Management

- Common class for parameters access required in algorithm
- Values accessible using getters
- Parameters requiring changes need change at single place
- Better way to keep track of parameters

```
//.....................PARAMETERSERVER.CPP.....................//


ParameterServer::ParameterServer(ros::NodeHandle &nh)
{
    nh.getParam ("scanner_unit/l2m_time_offset", l2mTimeOffset);
    nh.getParam ("scanner_unit/imu/intrinsics/bias", imuOffset);
    nh.getParam ("scanner_unit/laser/rotation_matrix", laserMountingOffsets);
    nh.getParam ("scanner_unit/camera/intrinsics/matrix/data", cameraMatrix);
    nh.getParam ("scanner_unit/camera/extrinsics/rotation/data", rotation);
}



//.....................PARAMETERSERVER.CPP.....................//
```

Config Data Management

- No single file for all config & calibration data
- Config data for different transducers in separate files
- All configuration files in separate *config* folder
- Flexibility in using different combinations of transducer data
- Name of config files has defined format i.e:

  <Transducer>_<Model_Name>_<Serial_Name>

  Eg. *Lidar_Hokuyo_AB123PQ*
- Namespace hierarchy created for different motor sections

  Eg. */GETjag/scanner_unit/camera1/lidar_Hokuyo_AB123PQ*

Config Data Management: File System

```
//.....CAMERA_XTION_XX1011.YML.....//

camera:
   # Camera intrinsic configuration i.e internal camera matrix
   intrinsics:
      matrix:
         rows: 3
         cols: 3
         dt: f
         data: [4.48411134e+02, 0., 4.063411580e+02, 0., 4.47164994e+02, 2.86936200e+02, 0., 0., 1.]

//.....CAMERA_XTION_XX1011.YML.....//




//.....IMU_XSENS_ZZ2101.YML.....//

imu:
   # IMU intrisic values
   intrinsics:
      bias: # Bias offset for the IMU (x, y, z)
        - 0.124680014534554
        - -0.01681371945652202
        - 9.778030257622401

//.....IMU_XSENS_ZZ2101.YML.....//
```

Contents

Documentation

- Identified key code sections missing/requiring comments
- Added comments, annotations, and explanations
- Added a *Readme.md* file for introductory idea of code
- Utilized *Doxygen* for automated documentation generation
- Current documentation available in HTML and PDF format

Contents

- Introduction
- Fundamentals
- Code Analysis
- Next Best View
  - Concept
  - Implementation
  - Results
- Summary
- Future Work

Why ground truth data?

- LOAM algorithm creates maps of environment in real-time
- Many ways to alter parameters for better output
- BUT: how to evaluate „better output“ ?

- Ground Truth Maps: point clouds with high detail / precision
  - stationary scans, more data to process
  - time consuming

- Make data collection more efficient: Next Best View

Next Best View

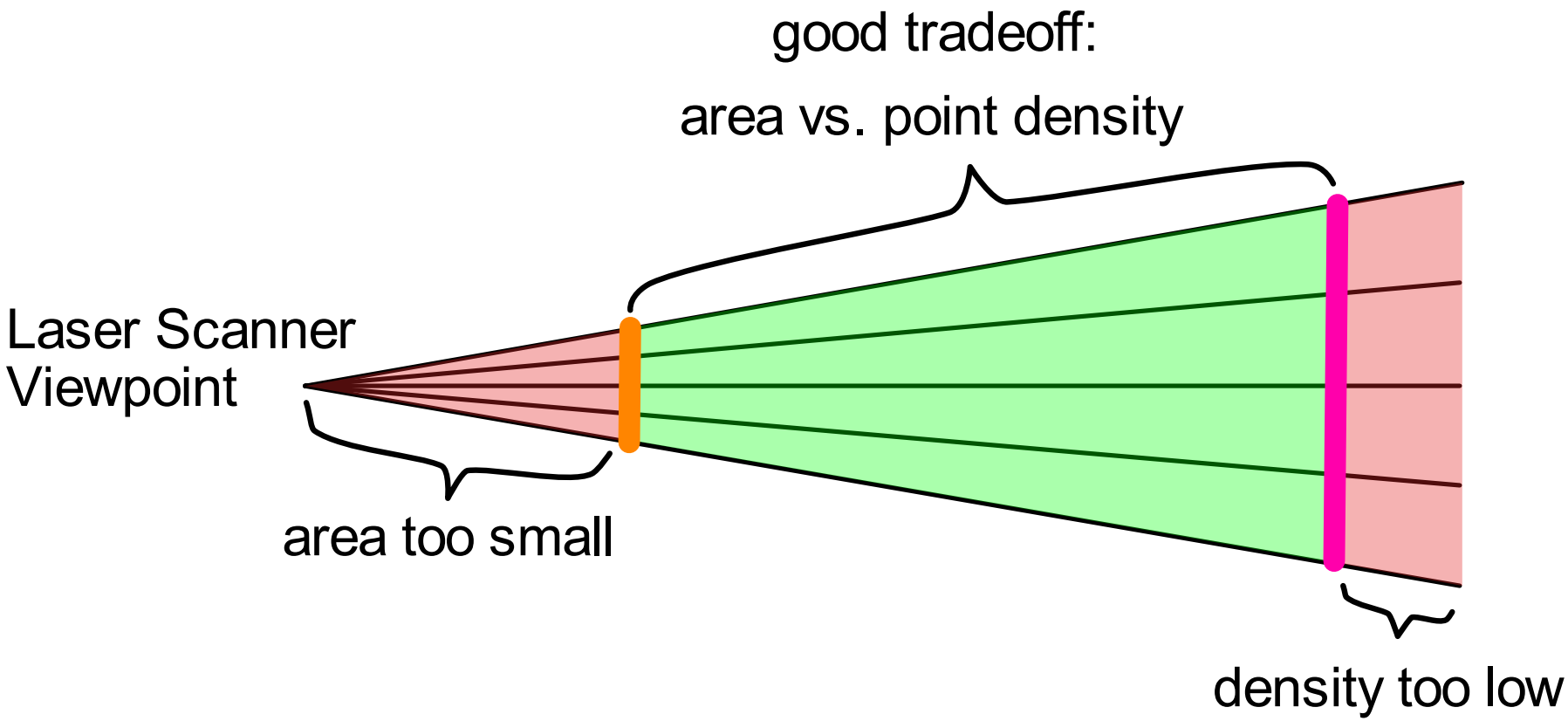
- Selecting most informative viewpoint for a sensor
- Planning viewpoints to optimize understanding of a scene
- Optimizing use of resources
  - time
  - energy
- Reduce redundancy and enhance efficiency of data collection

NBV: Concept

1. **Initial View:** capture partial view of scene
2. **Candidate Viewpoints:** propose new viewpoints
3. **Scoring or Ranking:** evaluate viewpoints for information gain
4. **Sensor Movement:** move sensor and capture new data

Considerations / Limitations

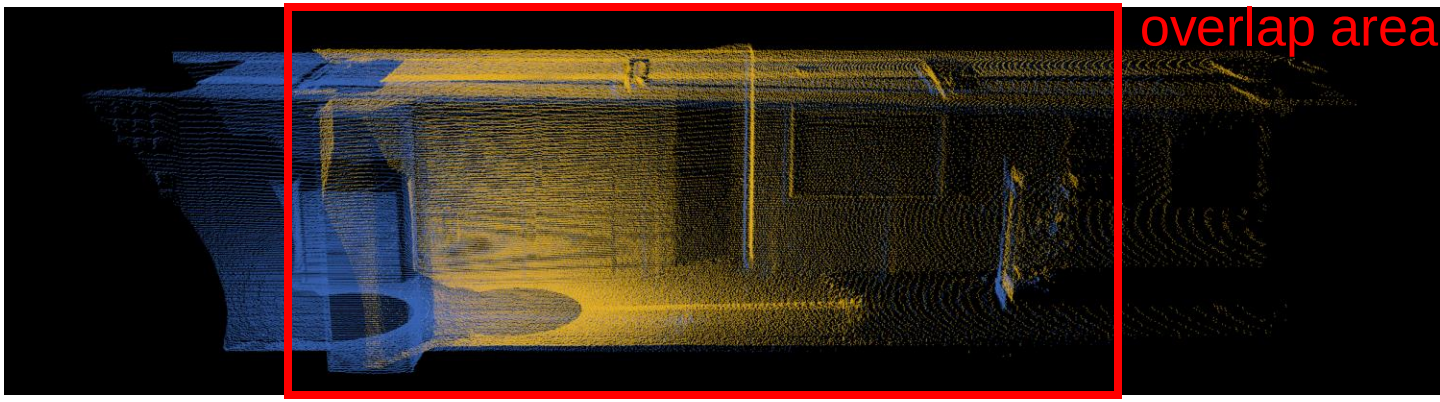
- **Covered area and Point density:**



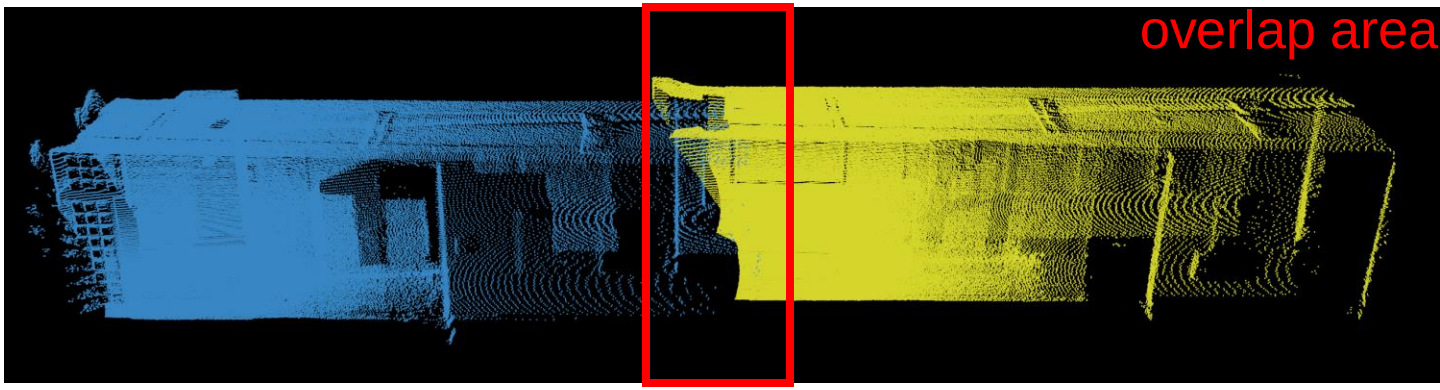
Considerations / Limitations

- **Registration constraints:**
  - for larger maps, multiple scans are combined
  - sufficient overlap of individual scans needed

- Sufficient:

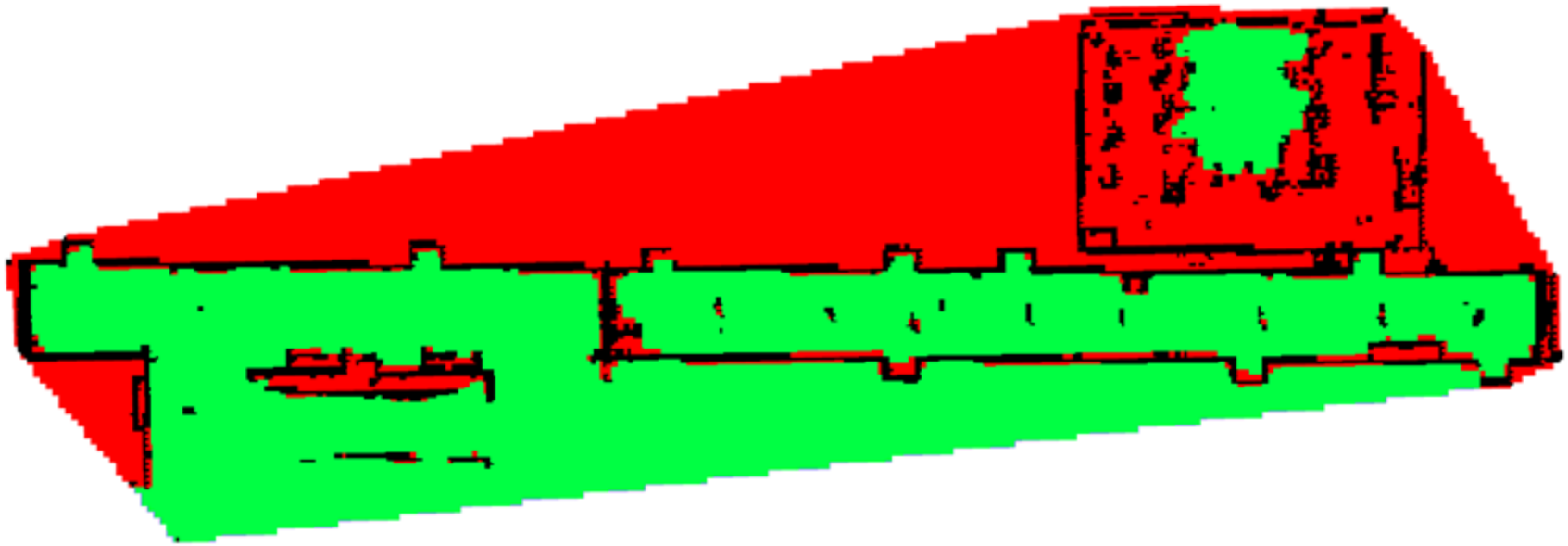


- Insufficient:



Considerations / Limitations

- **Position constraints:**
  - Robot is bound to 2D movement on floor

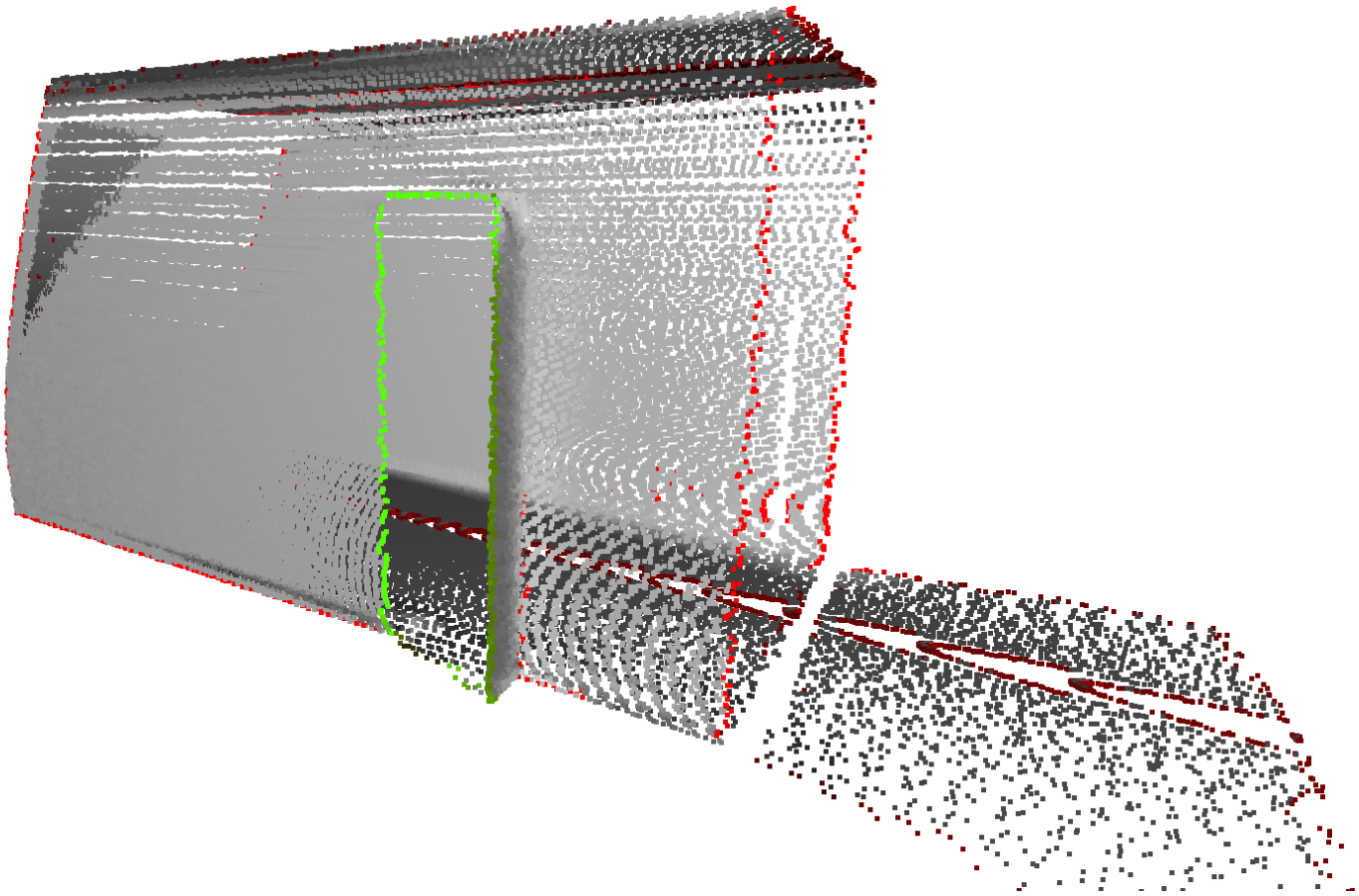


Contents

- Introduction
- Fundamentals
- Code Analysis
- Next Best View
  - Concept
  - Implementation
  - Results
- Summary
- Future Work

Detect Boundary Points

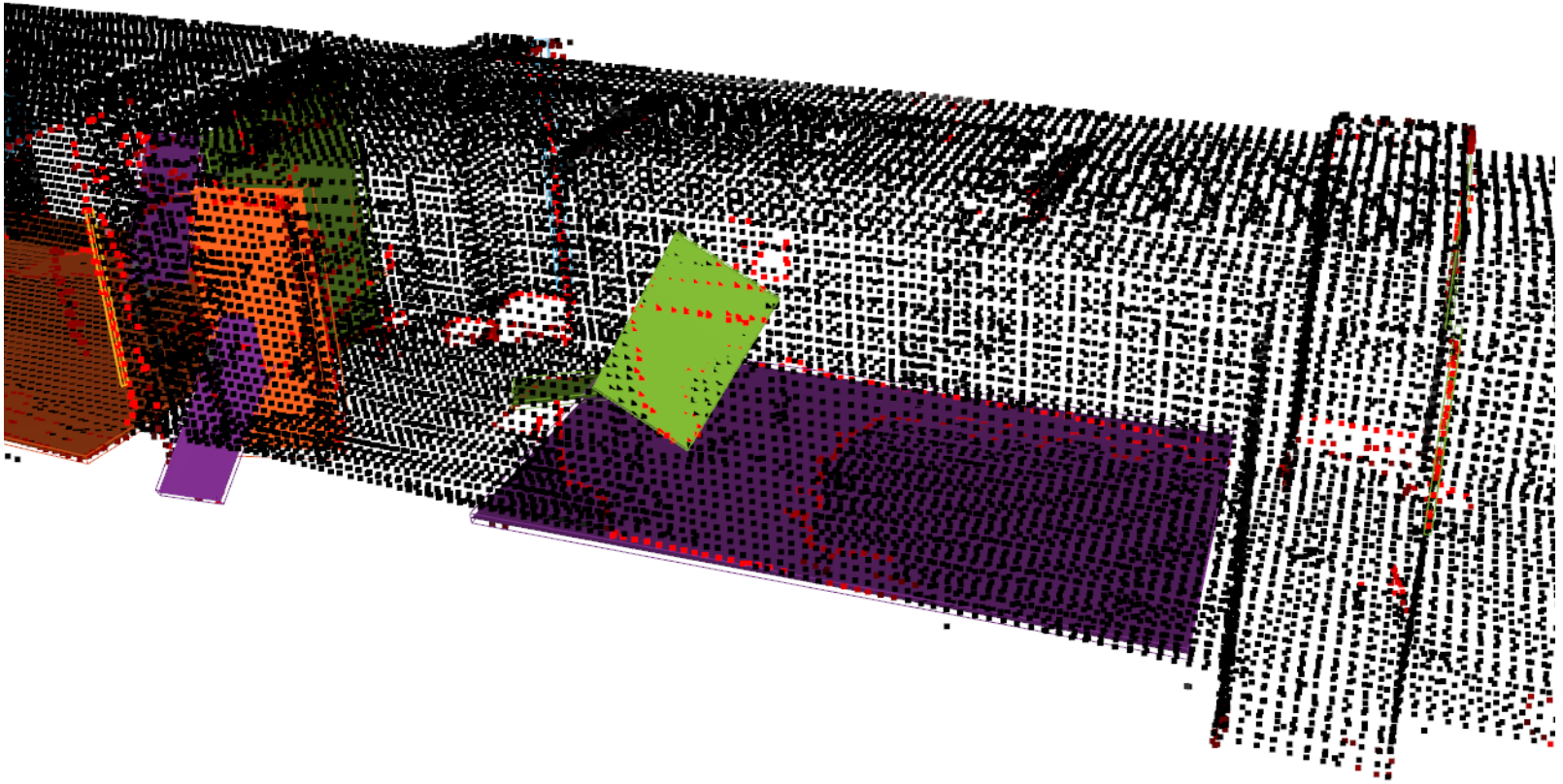
- Idea: each boundary point in point cloud corresponds to a hole in that point cloud



Hole patches from boundary points

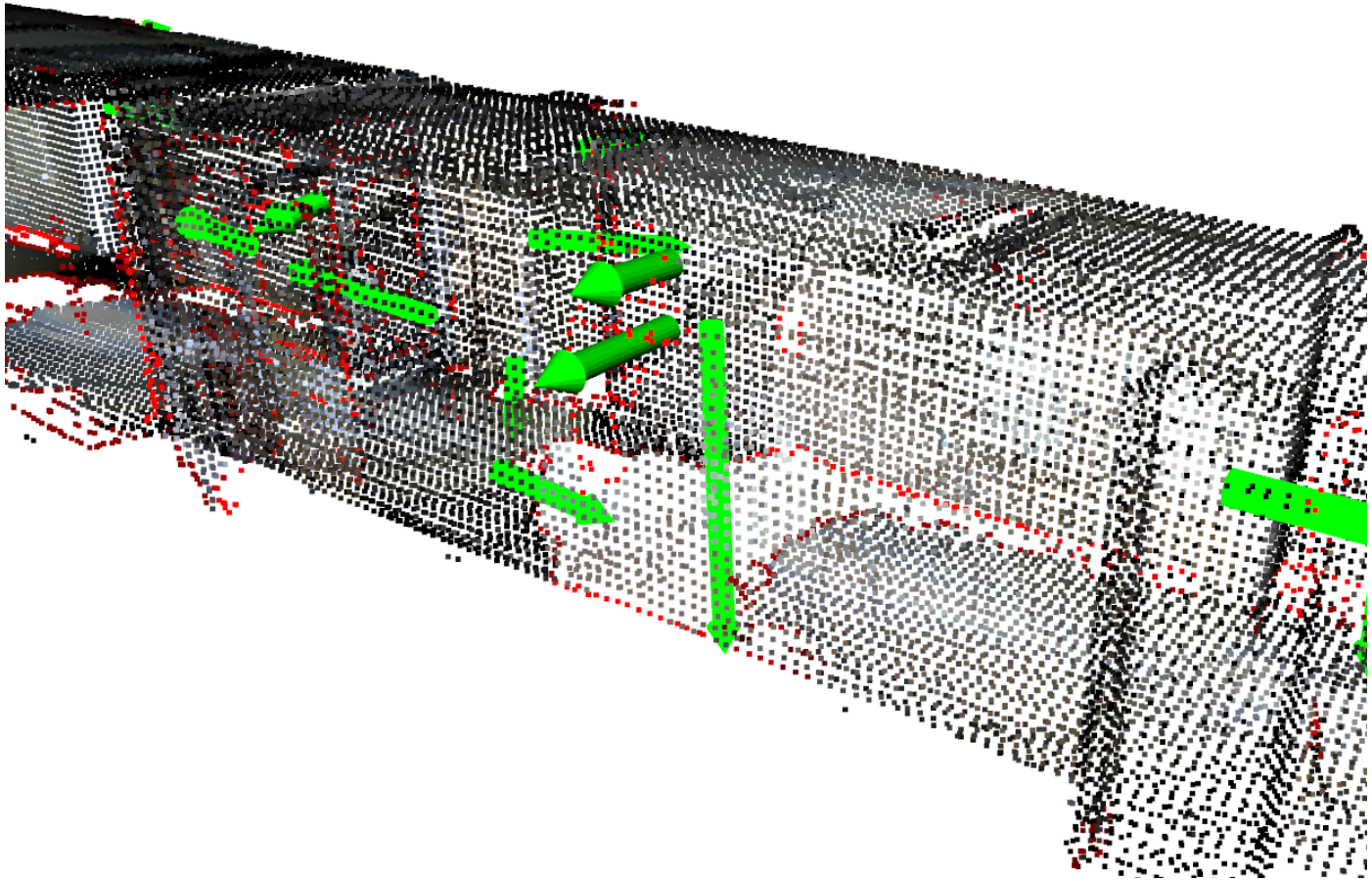
- Idea: identify boundary points that belong together, forming a hole
- Extract boundary points, detect planar patches

[open3d]



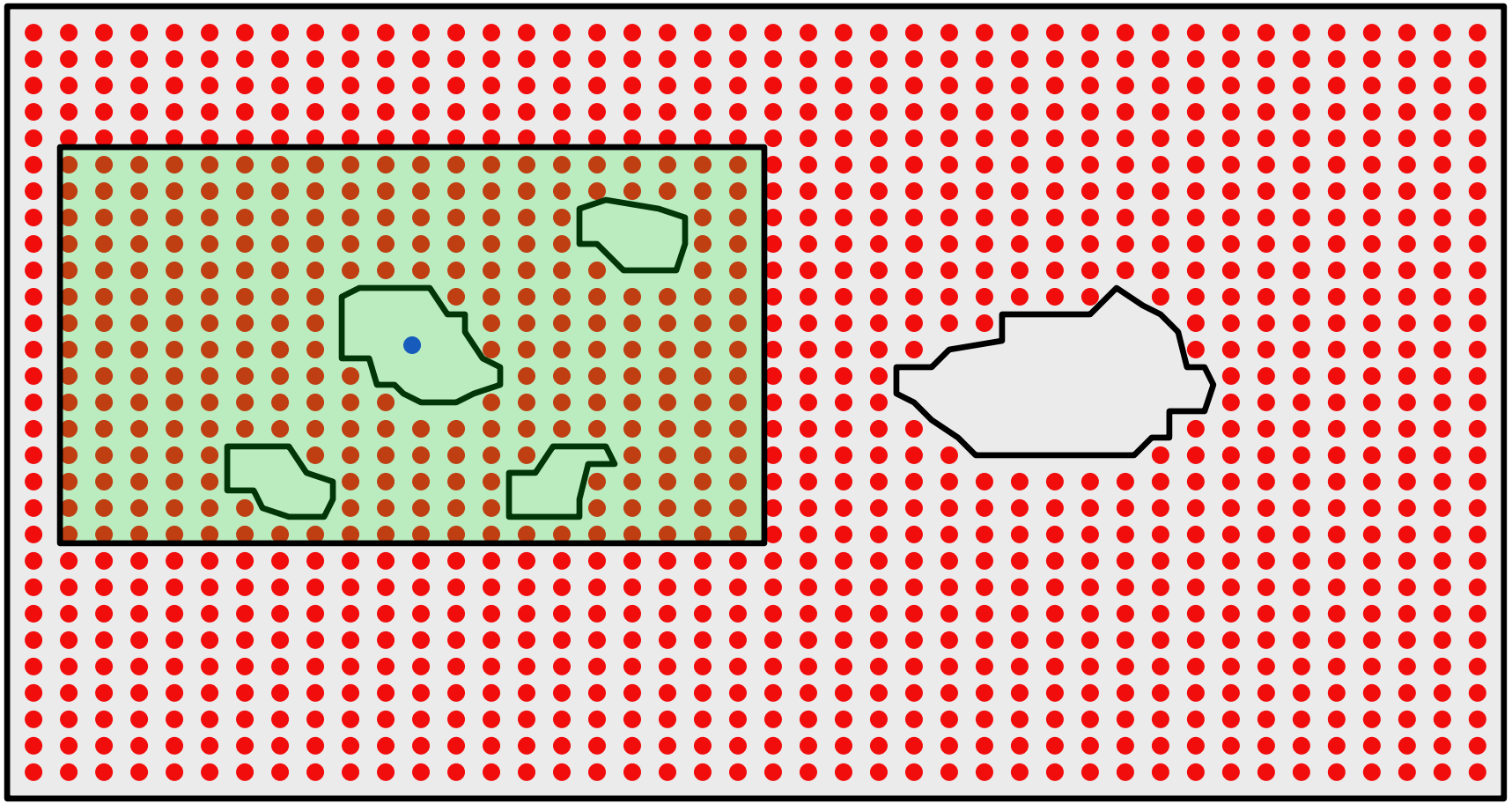
Initial Next Best Views

- Propose views which are evaluated for information gain
- Ideal view: perpendicular to surface being covered



Initial Next Best Views

- Many individual views, try to cover multiple at once
- Evaluate area information gain for each view

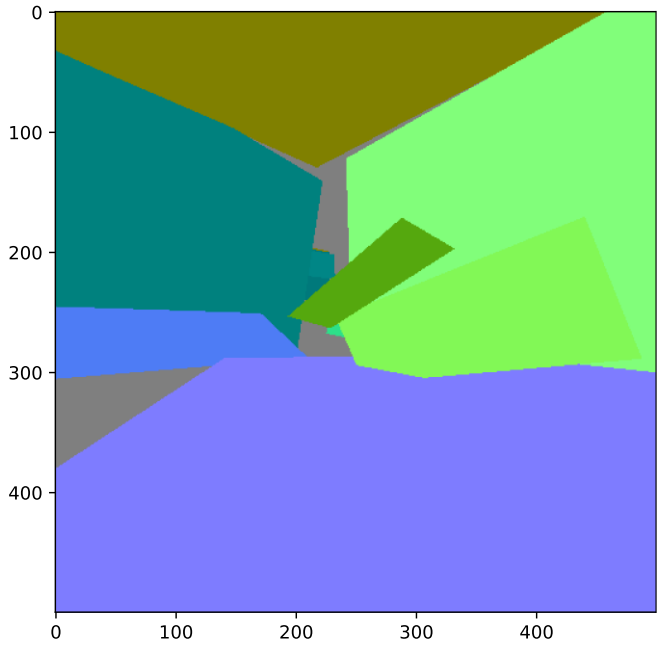


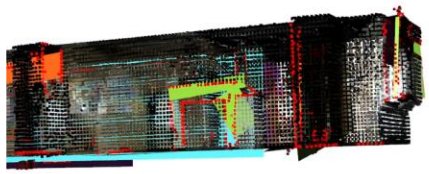
Evaluate Views: Ray Casting

- Utilize Open3D ray casting scene
- Ray: 6D-vector, origin and orientation
- Casted ray can return:
  - Distance to surface
  - Normal of surface
- Simulate virtual camera with array or rays
  - FOV
  - Resolution

Evaluate Views: Ray Casting

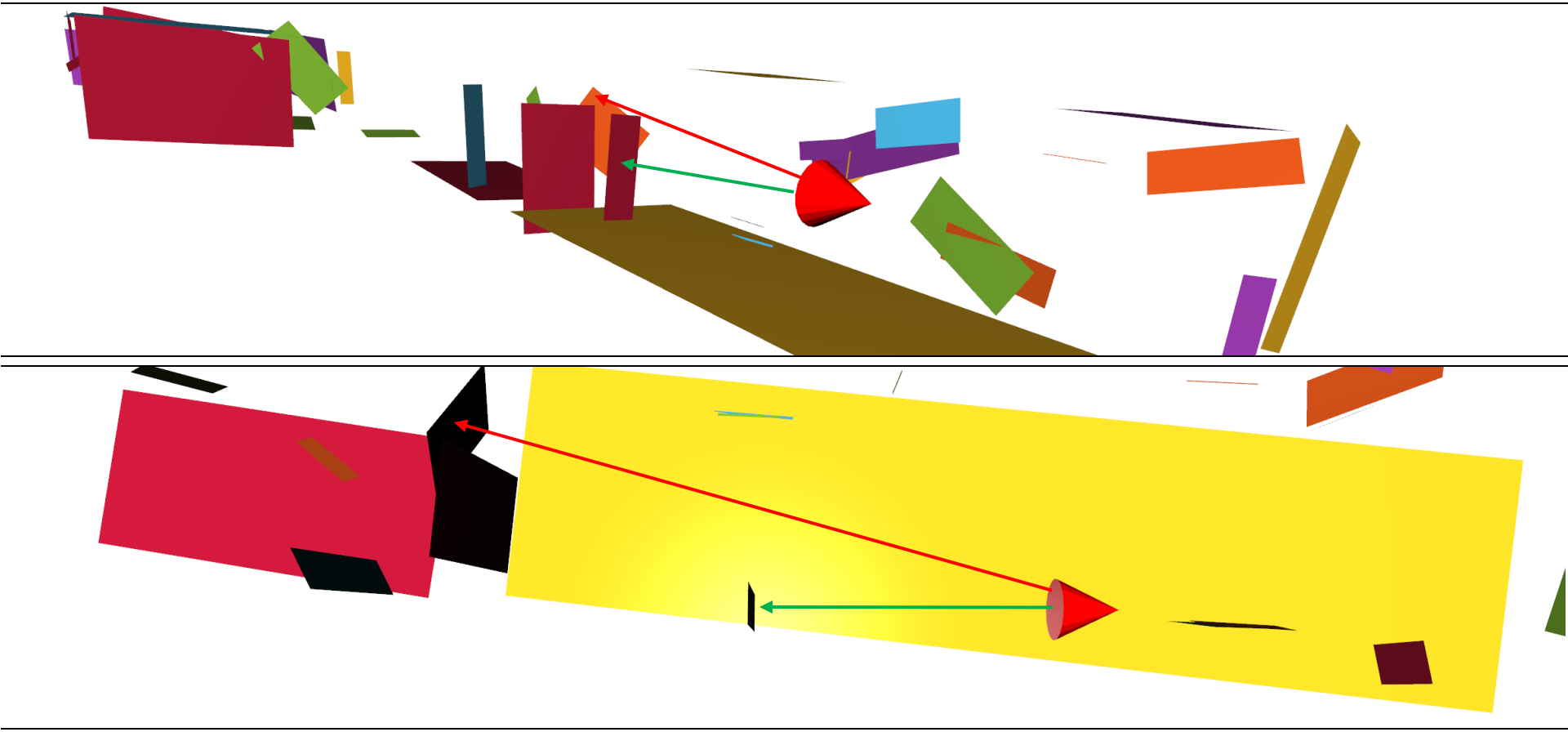
- Take virtual pictures of environment
- Analyze picture for information gain





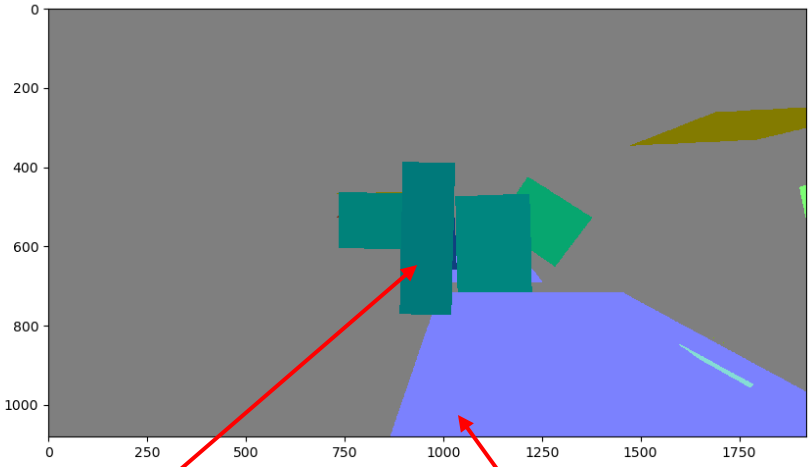
Ranking Criteria

- Distance to scanned object:
- Green: valid range Red: out of range

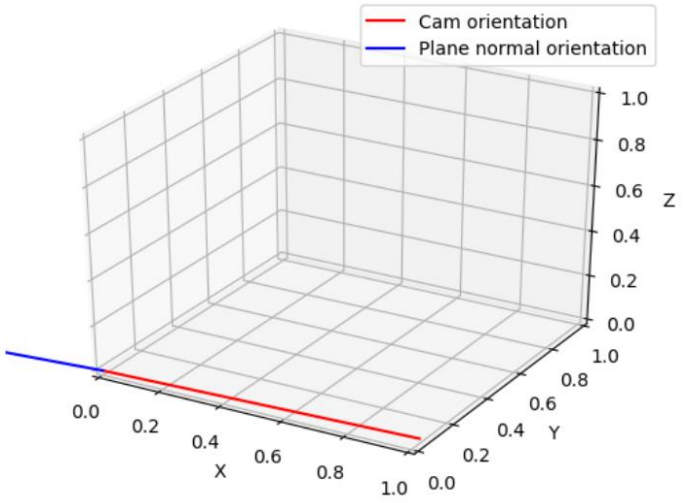


Ranking criteria

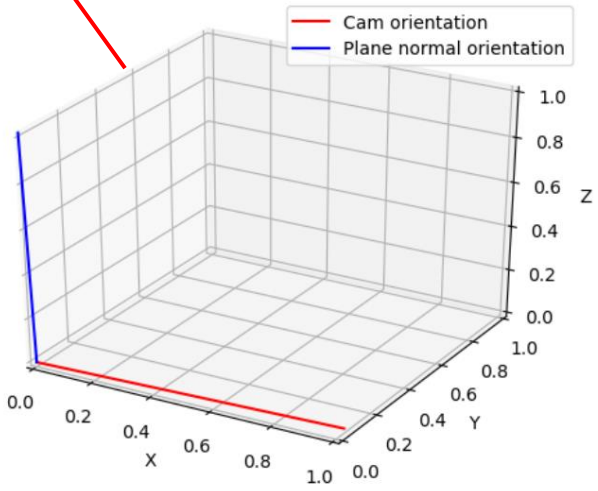
- Angle between surface normal and laser beam



Good orientation:

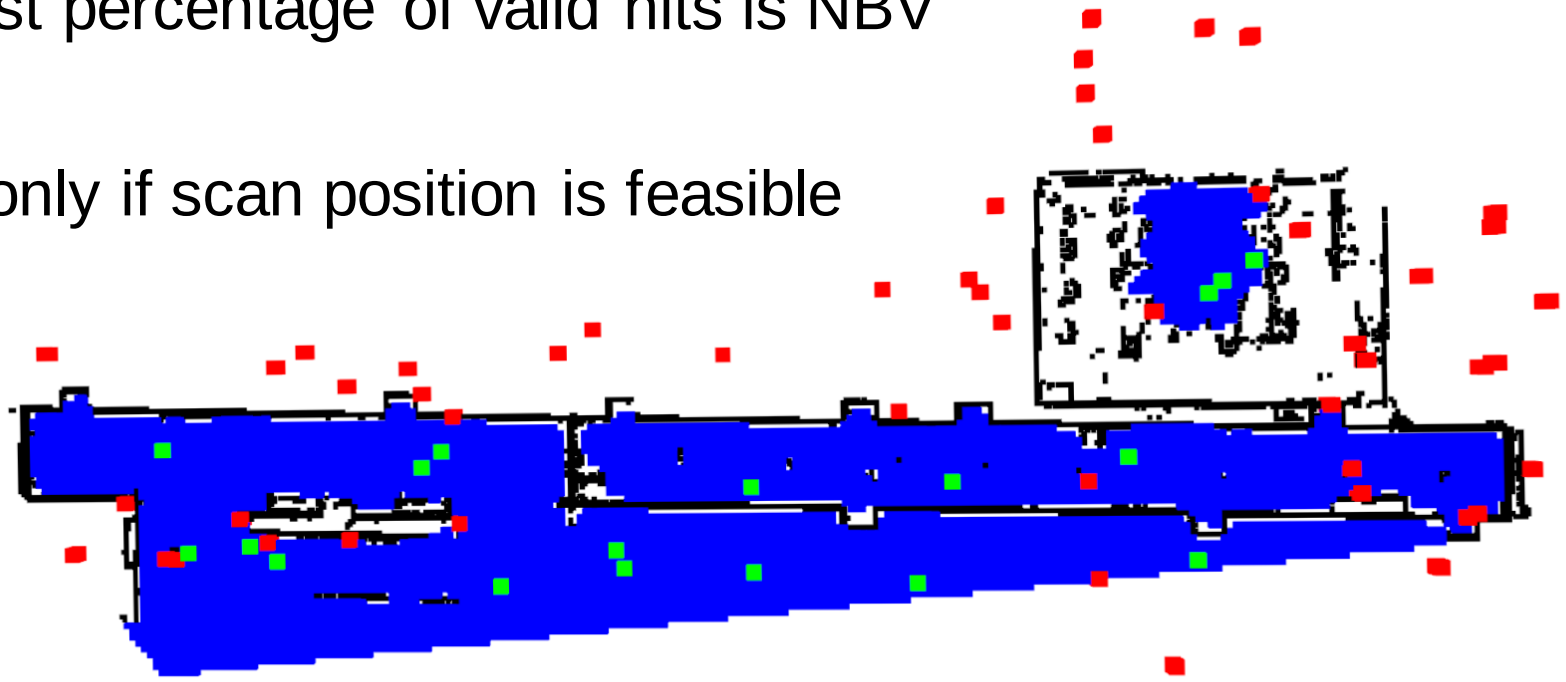


Bad orientation:



Ranking

- Count valid hits fulfilling distance and angle threshold
- Calculate percentage of valid hits to rays casted
- Highest percentage of valid hits is NBV
- BUT: only if scan position is feasible



Contents

Floor Space Detection

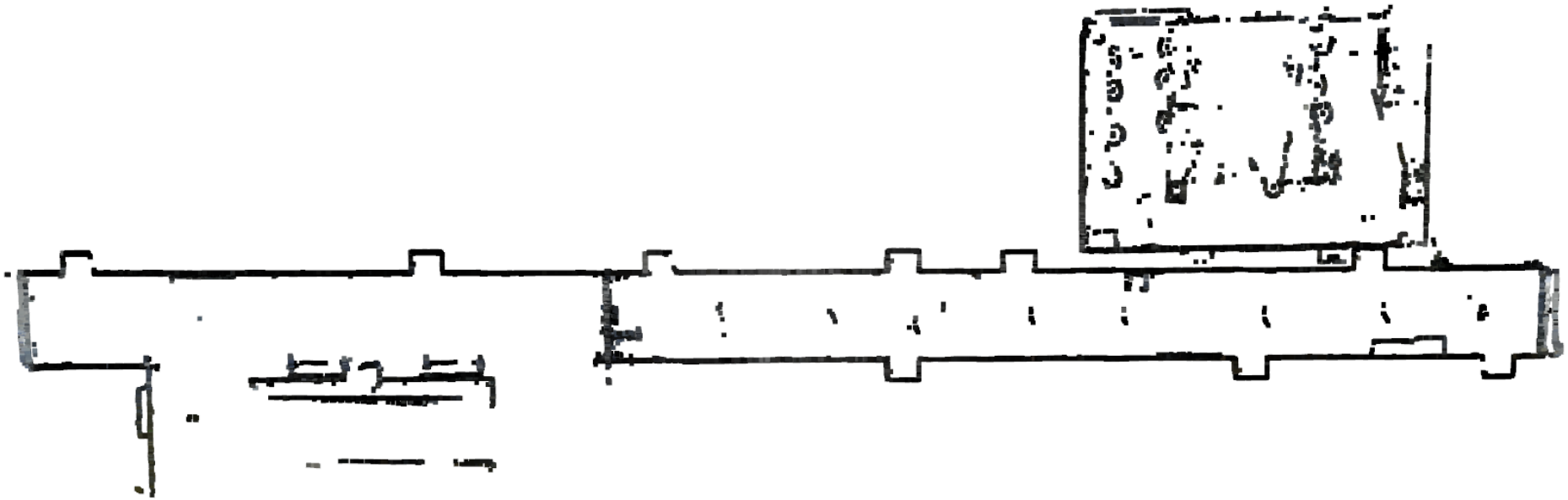
- NBV may not be feasible for robot
- Verify the robot's ability to reach a position in the given environment
- Detect free floor space inside the point cloud
- Exclude inner wall space: appears as a void area, but is not accessible

Void Growing Approach

- Idea based on: “VOID-GROWING: A NOVEL SCAN-TO-BIM METHOD FOR MANHATTAN WORLD BUILDINGS FROM POINT CLOUD”
- Void detection in 3-dimensional point clouds of Manhattan World buildings
- Adapted to 2-dimensional use case
- Concept: find seeds of void space, grow in all directions until obstacle is hit

2D Projection

- Remove points belonging to floor or ceiling
  - Point normals taken as criteria
- Project 3D point cloud on x-y-plane

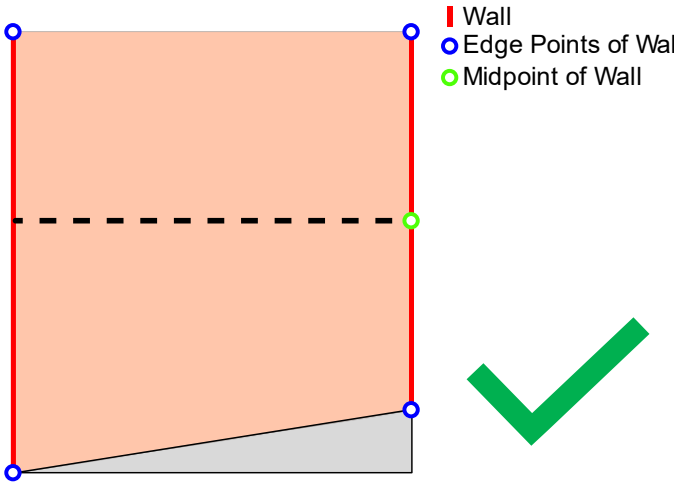


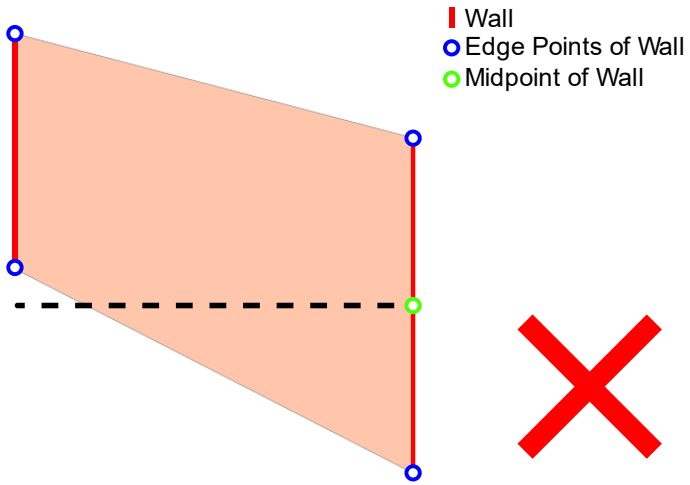
Wall detection

- Utilize plane detection to identify walls in point cloud
- Assumption: 4 walls enclosing a room
- If corresponding walls can be detected:
  - Inner space must be void area
- Goal: find pairs of walls belonging to one room
  - Midpoint between walls is seed for void growing

Wall detection - Criterias

1. **Wall distance:** pairs must have minimum distance
2. **Wall alignment:** midpoints in between endpoints

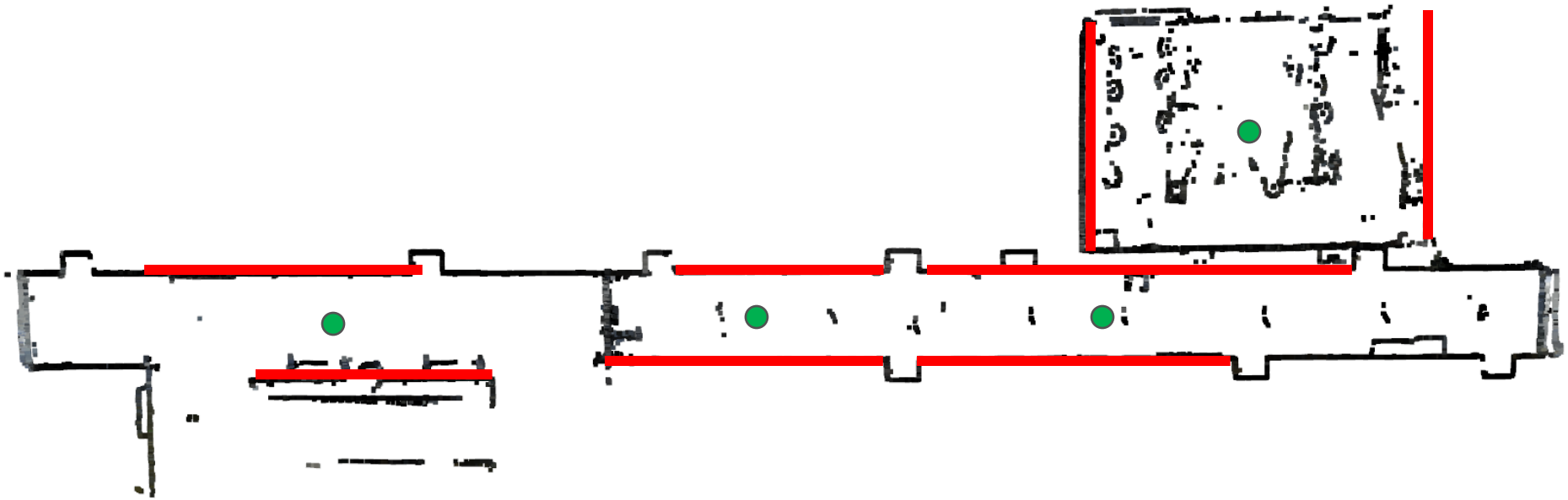




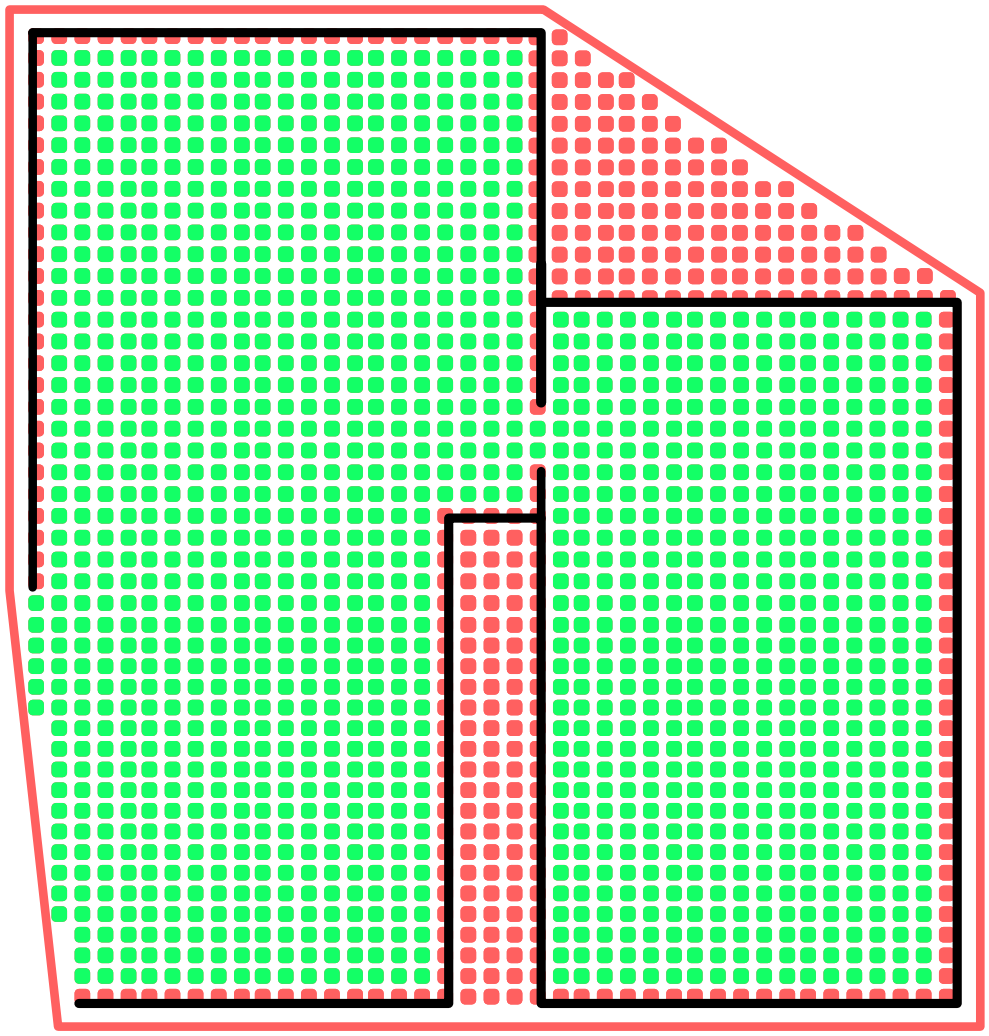
3. **Nearest wall:** only the nearest wall fulfilling both criteria can be corresponding pair

Initial Seeds

- Calculate midpoints between corresponding walls
- Midpoints are assumed to be inside room => void space
- Initial seed for void growing



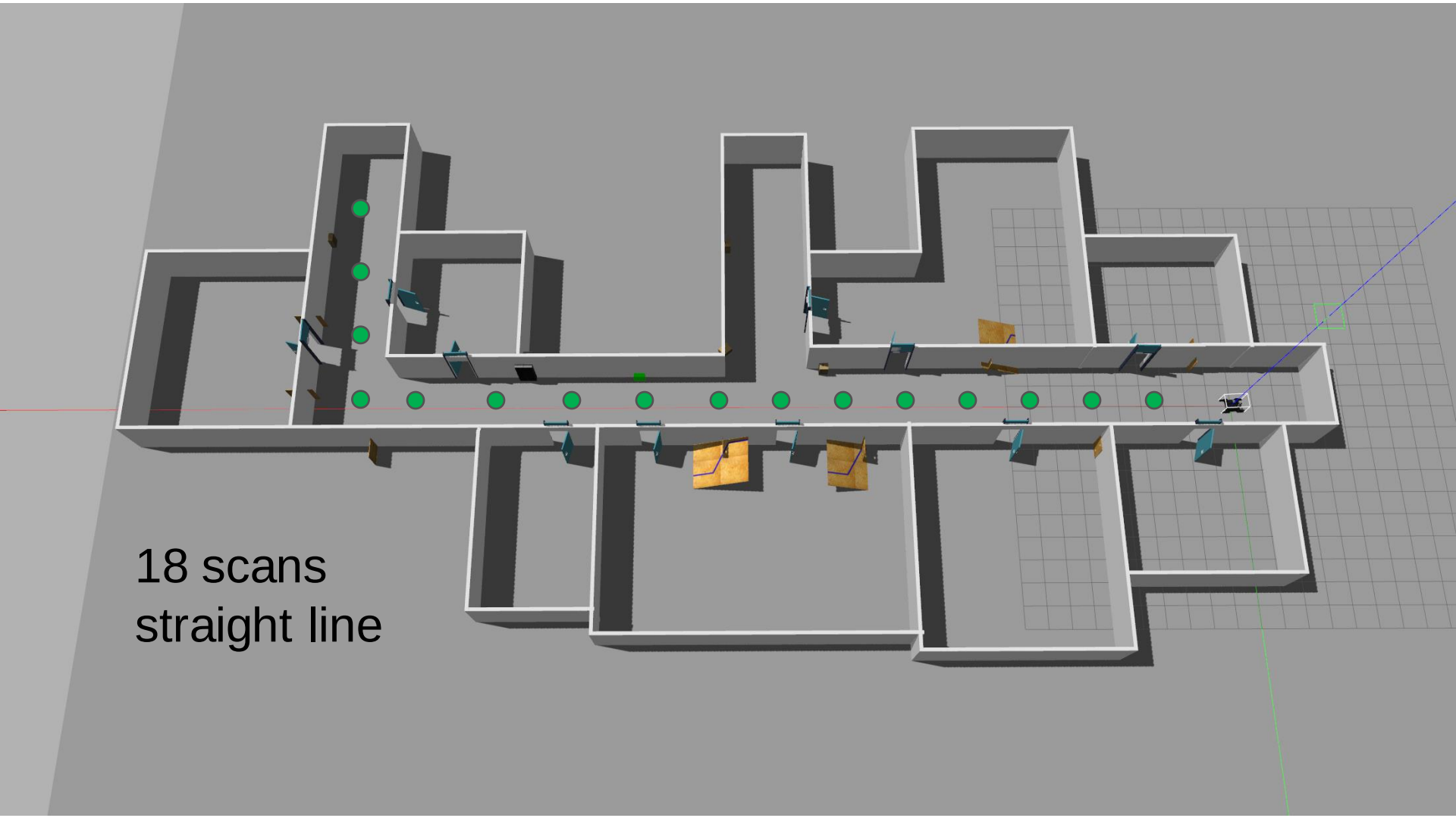
Void Growing



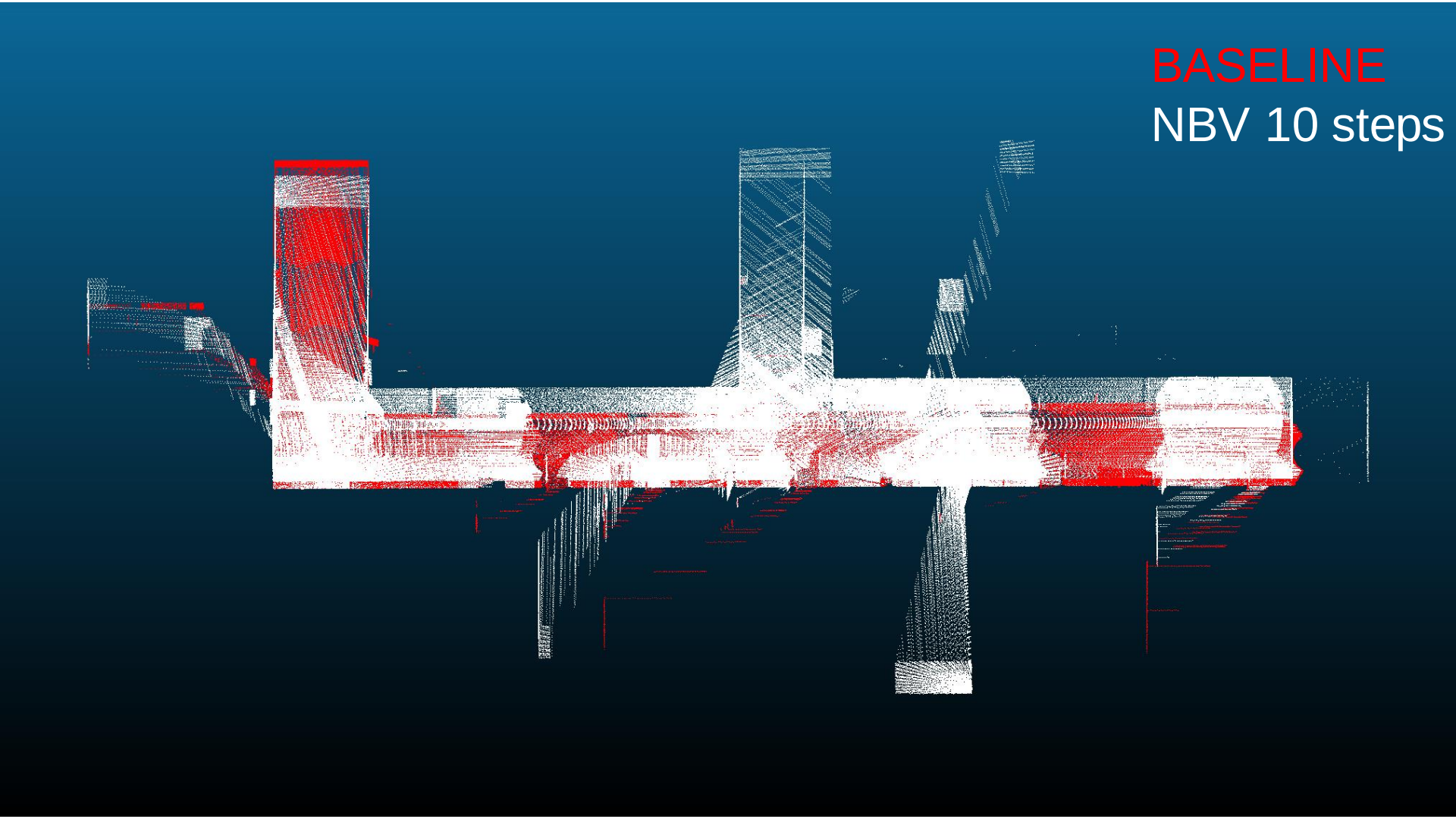
Contents

- Introduction
- Fundamentals
- Current Code Analysis
- Next Best View
  - Concept
  - Implementation
  - Results
- Summary
- Future Work

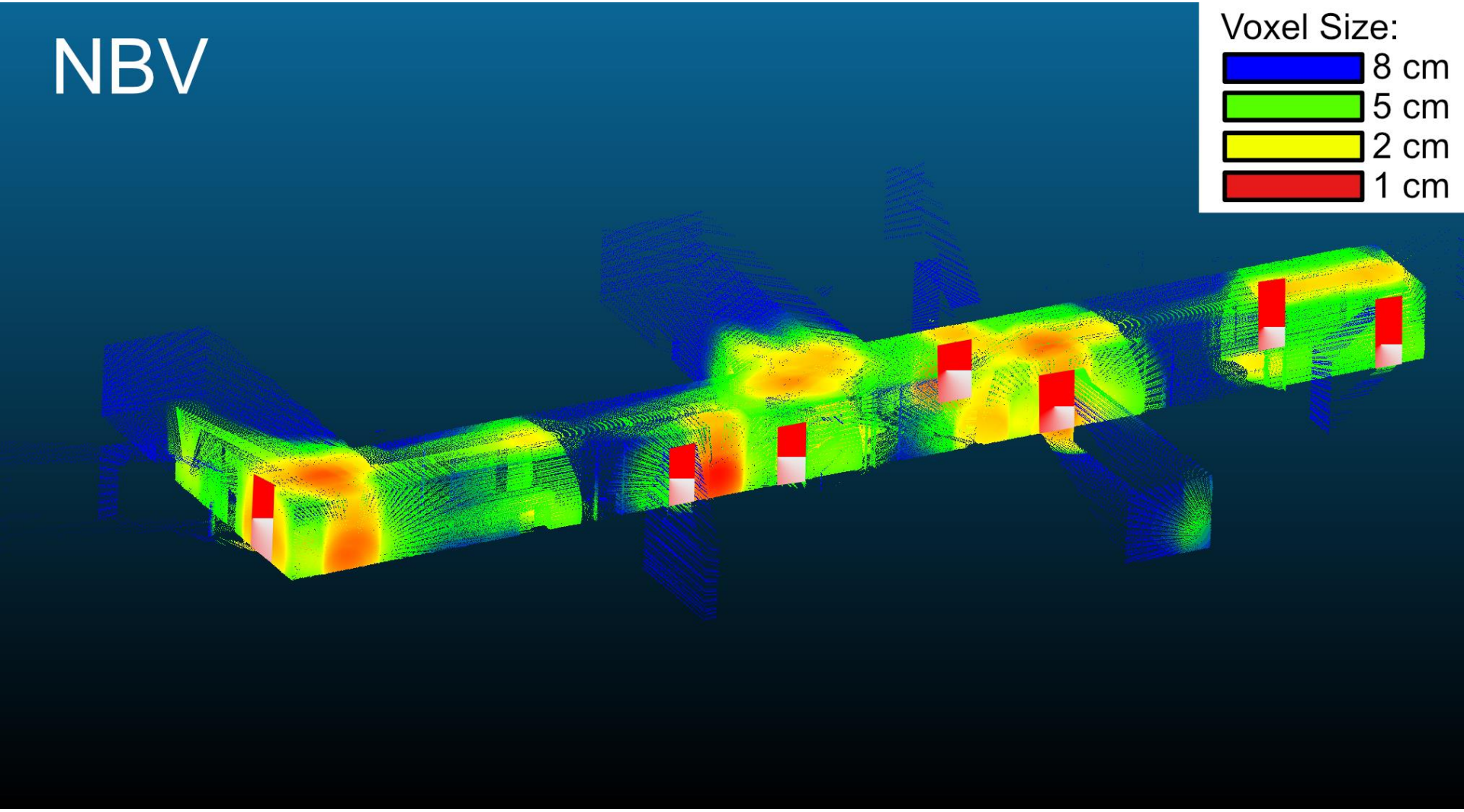
Testing Environment



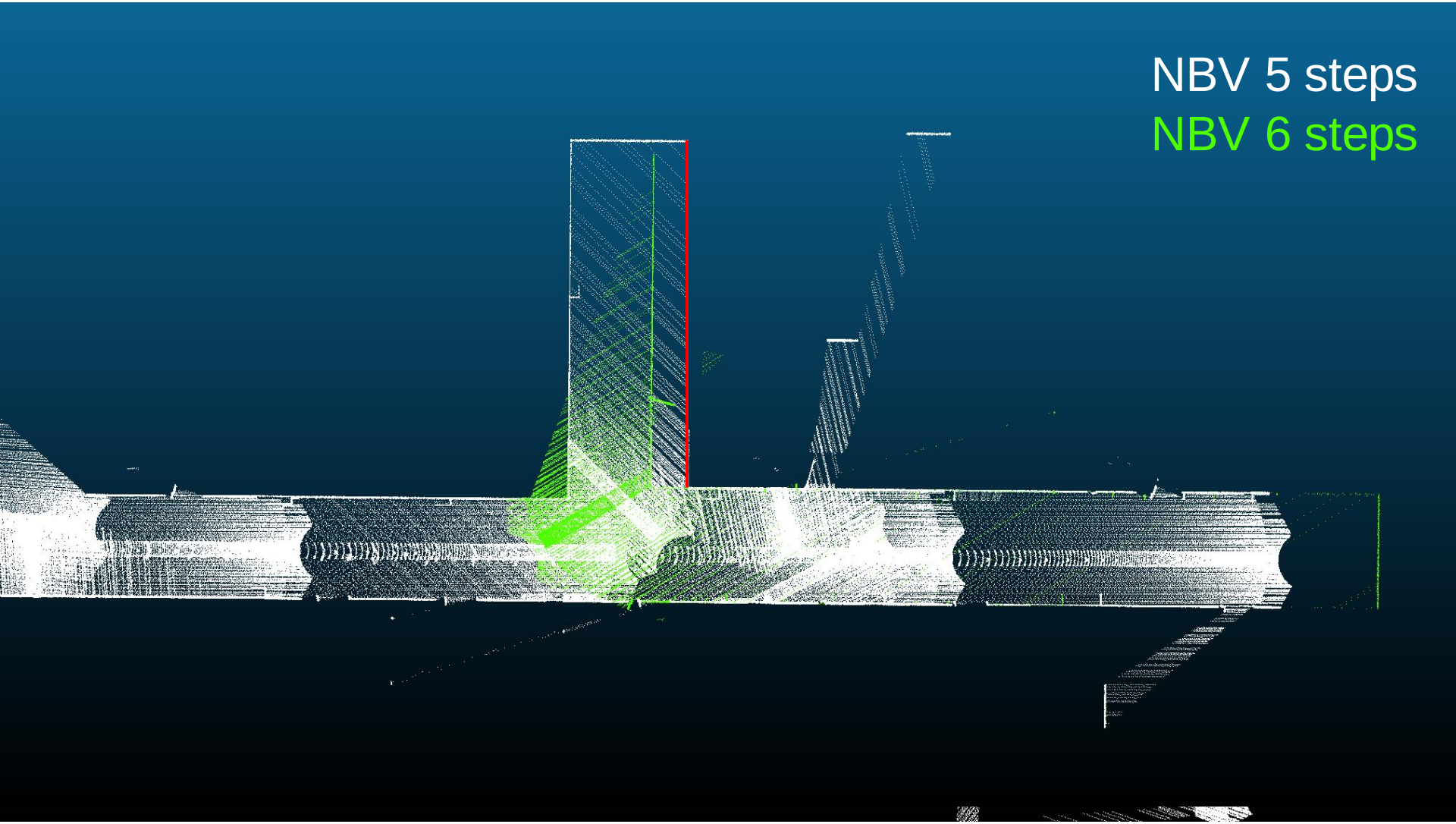
Straight Line vs NBV



Point Density



Problem with registration



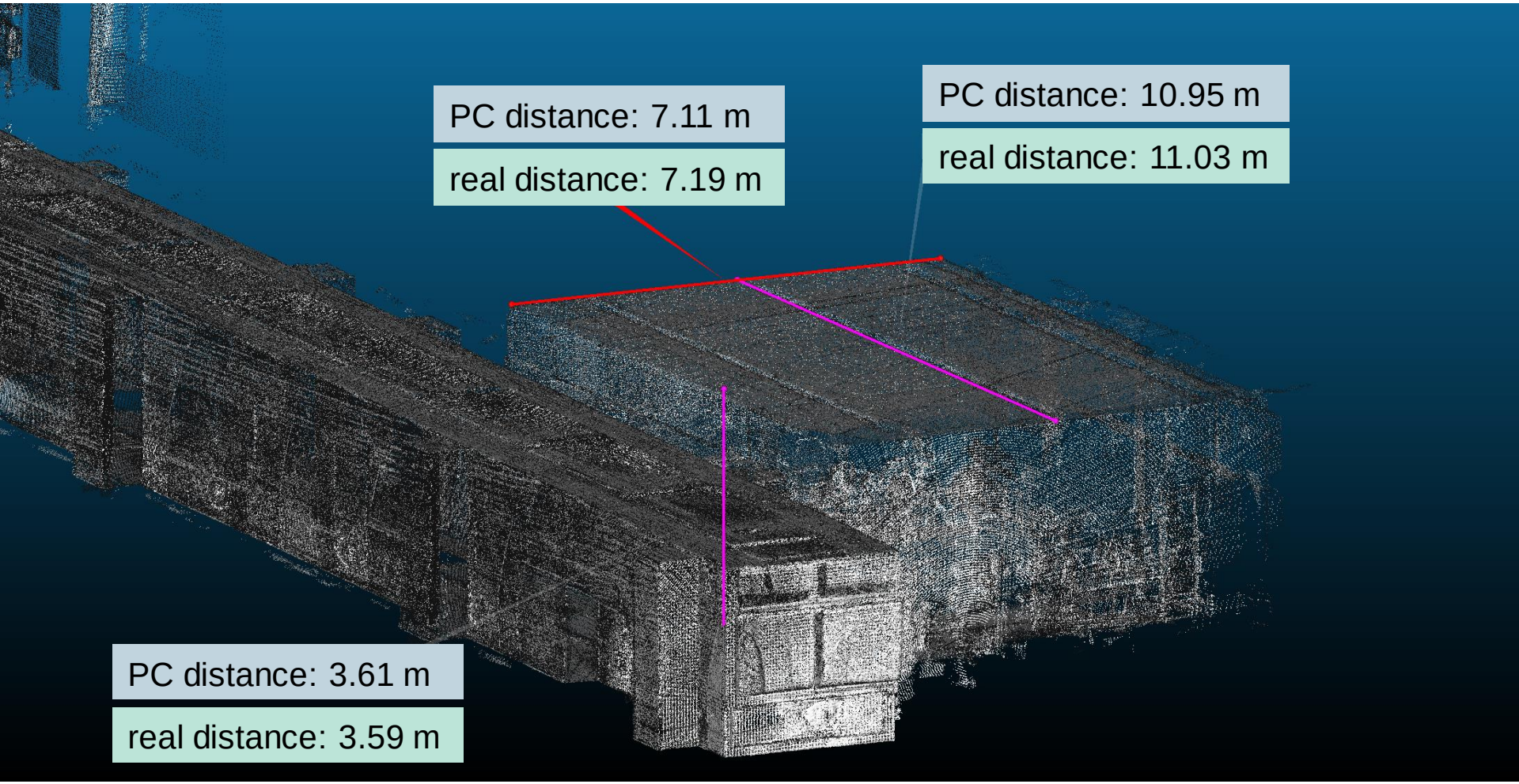
Takeaways

- NBV has problems without initial coverage
- NBV approach more suitable for fixing missing parts

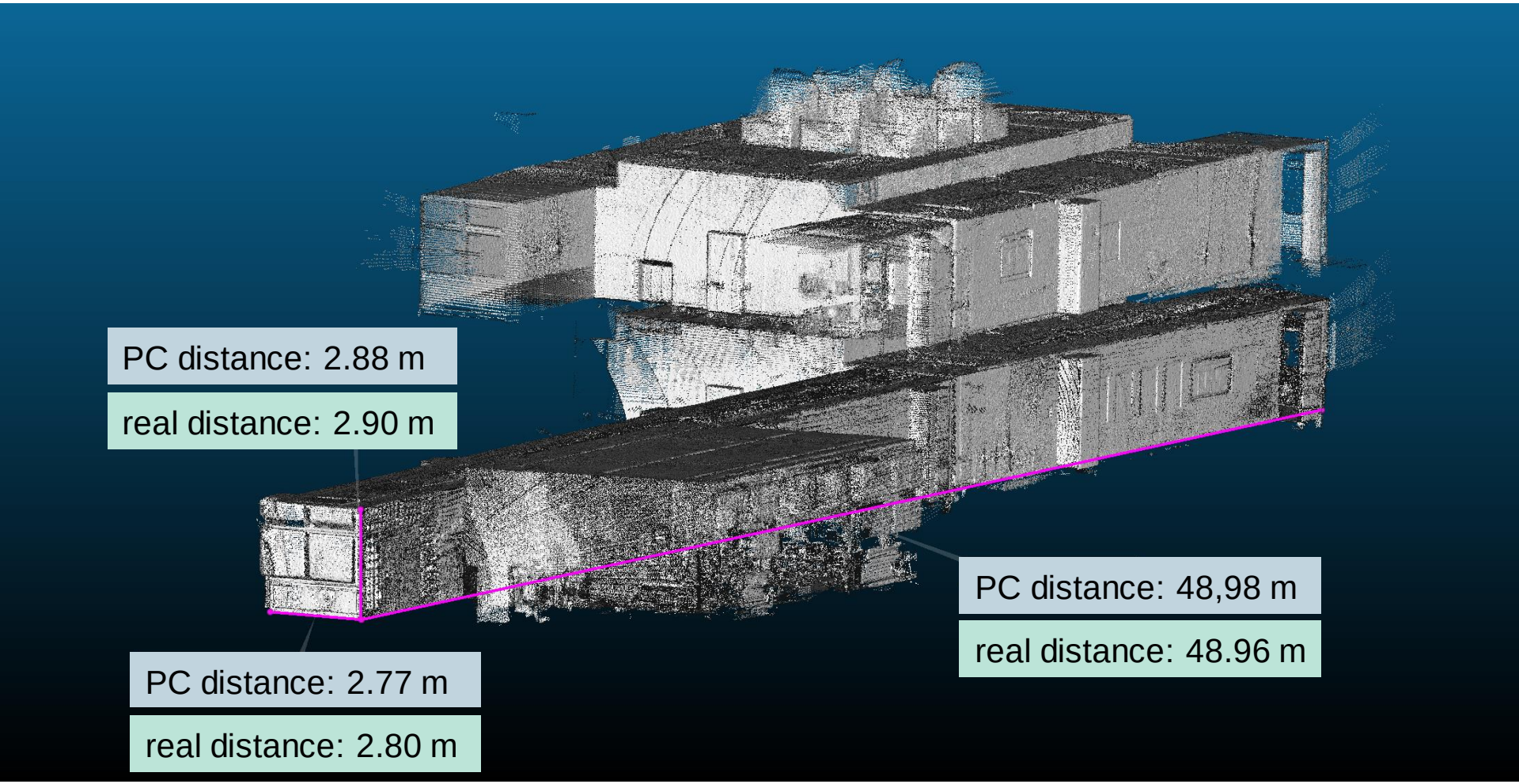
- NBV has problems with wide open areas
- High information gain can lead to registration problems

- Combination of initial scans and NBV works best
- Resulting in high quality maps

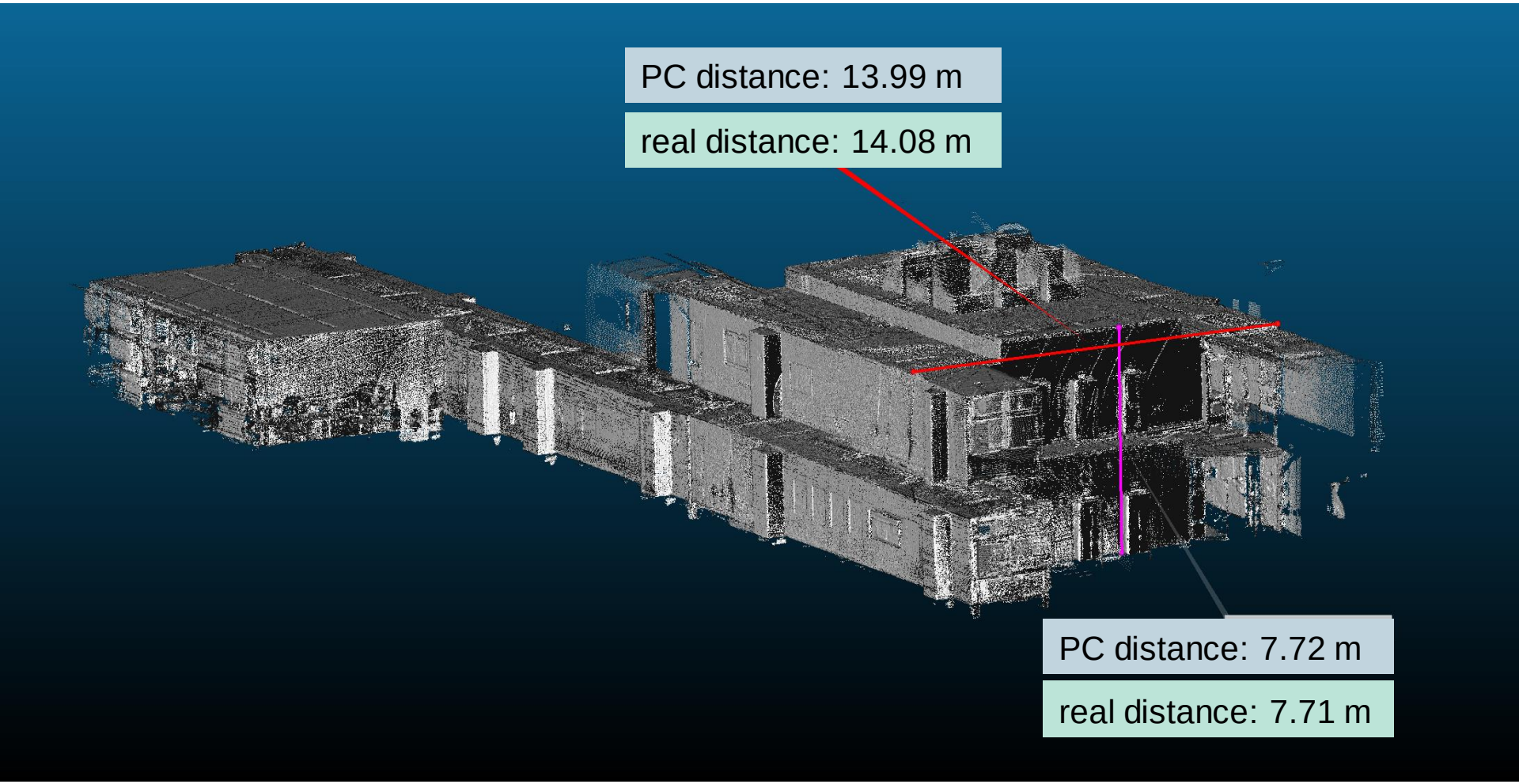
Point Cloud vs Real Distance



Point Cloud vs Real Distance



Point Cloud vs Real Distance



Point Cloud vs Floorplan

Floorplan
P1.6

Point Cloud vs Real Distance

Distance in Point Cloud [m]	Real Distance [m]	Absolute Deviation [m]	Deviation from real distance [%]
7.11	7.19	0.08	1.11
10.95	11.03	0.08	0.73
3.61	3.59	0.02	0.56
2.88	2.9	0.02	0.69
2.77	2.8	0.03	1.07
48.98	48.96	0.02	0.04
13.99	14.08	0.09	0.64
7.72	7.71	0.01	0.13
	Average:	0.04375	0.62

Limitations

- Limited to movement on single floor
  - pose estimation from wheel odometry
  - no z-height
- Limited to indoor spaces
  - hole detection based on planar patches
  - floor space detection based on finding opposing walls
- Registration constraints not considered
  - NBV can propose view with insufficient overlap

Contents

Summary

Code Analysis:

- Fixing of bugs present in current code
- Restructuring and optimization of code base
- Completion of Systematic Documentation

NBV:

- Strategic viewpoint selection with NBV
- Detection of possible scan positions
- Creation of high detail / precise ground truth maps

Contents

Future Work

- Optimization of Mapping Manager & Transform Maintenance classes
- Improving resolution of generated map
- Implement 3D descriptors for loop closure
- Merge keyframes having same data

Future Work

- Use different sensors for height information
- Extend / Optimize for different environments
  - density based clustering vs. planar patch detection
- Integrate registration constraints
- Take robot movement effort into account

Thank you!

Questions?

References

- Zhang, Ji; Singh, Sanjiv (2014-07-01). *LOAM: Lidar Odometry and Mapping in Real-time*
- Zhang, Ji; Singh, Sanjiv (2018). "Laser–visual–inertial odometry and mapping with high robustness and low drift".
- https://getwww.uni-paderborn.de/wiki/msr/RRS_(SS_2021)/Mapping/Print_Version
- www.researchgate.net/publication/318671034_A_Loosely-Coupled_Approach_for_Metric_Scale_Estimation_in_Monocular_Vision-Inertial_Systems
- http://www.open3d.org/
- https://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html
- https://mathworld.wolfram.com/Point-PlaneDistance.html
- VOID-GROWING: A NOVEL SCAN-TO-BIM METHOD FOR MANHATTAN WORLD BUILDINGS FROM POINT CLOUD, Yuandong Pan, Alex Braun, André Borrmann, Ioannis Brilakis, DOI: 10.35490/EC3.2021.162

References

- Östermann, Marc (January 2012). Autonomous 3D Mapping and Exploration in Interior Environments based on a Next-Best-View Approach (Master's Thesis). Paderborn University.
- Baker, Haydar M. (January 2013). Hierarchical Next Best View Planning For Efficient 3D Exploration Of Indoor Environments (Master's Thesis). Paderborn University.
- https://getwww.uni-paderborn.de/wiki/msr/RRS_(SS_2023)/Mapping

Detect Boundary Points

- Idea: each boundary point in point cloud corresponds to a hole in that point cloud

- Angle Calculation:
  - local neighborhood with radius around a point.
  - calculate the angles between the vectors of the central point to its neighboring points. The calculated angles are used for the evaluation.
- Angle Criterion:
  - threshold determines whether a point is considered part of a boundary or not.
  - on a boundary, the angles between neighboring vectors tend to change abruptly

[open3d]

Open3D Planar Patch detection

- Divide point cloud using octree
- Try to fit plane to each chunk: media point position
  - a*x + b*y + c*z + d = 0
- Planarity check:
  - 1. check variation angles between plane and point normals
  - 2. check variation of distance between plane and points
- Merge small plane fragments into larger planes

Void Growing

1. **KNN-Radius search:** search for neighboring points in grid and input point cloud

2. **Split directions:** divide found points into 4 directions: up, down, left, right

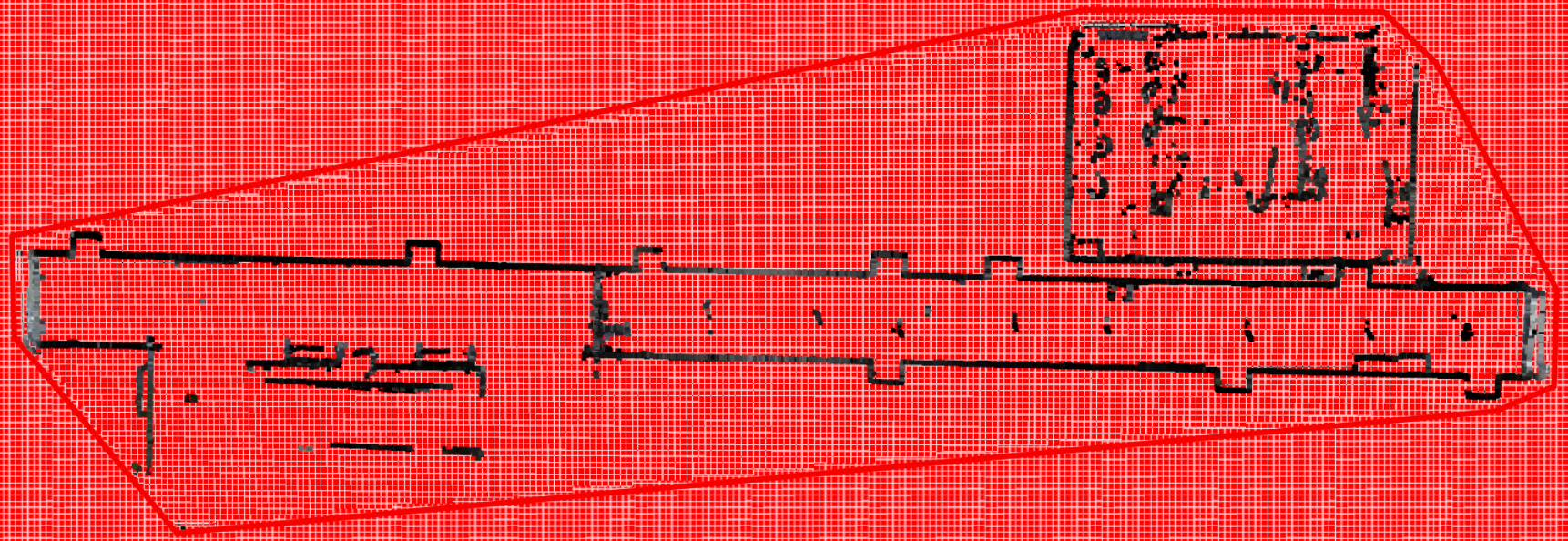
3. **Threshold check:** if number of found points in certain direction is below threshold => void area

4. **Mark points:** void points are colorized in grid and added to set of know void-points

5. **Iterate:** repeat steps for all newly found void points

Void Grid

- Grid-like point cloud to mark void space
- Cover area of input point cloud
- BUT: be as small as possible => convex hull



Documentation Screenshots

LIDAR SLAM 1.0

Main Page | Classes | Files | Search

LIDAR SLAM Documentation

Packages for 3D mapping based on LOAM: Lidar Odometry and Mapping in Real-time #How To Start: GETjag:

```
rosrun loam_misc scannerReal

roslaunch loam_mapping loam.launch
```

Simulator:

```
roslaunch loam_mapping gazebo.launch

roslaunch loam_mapping loamSimulator.launch
```

GETbot

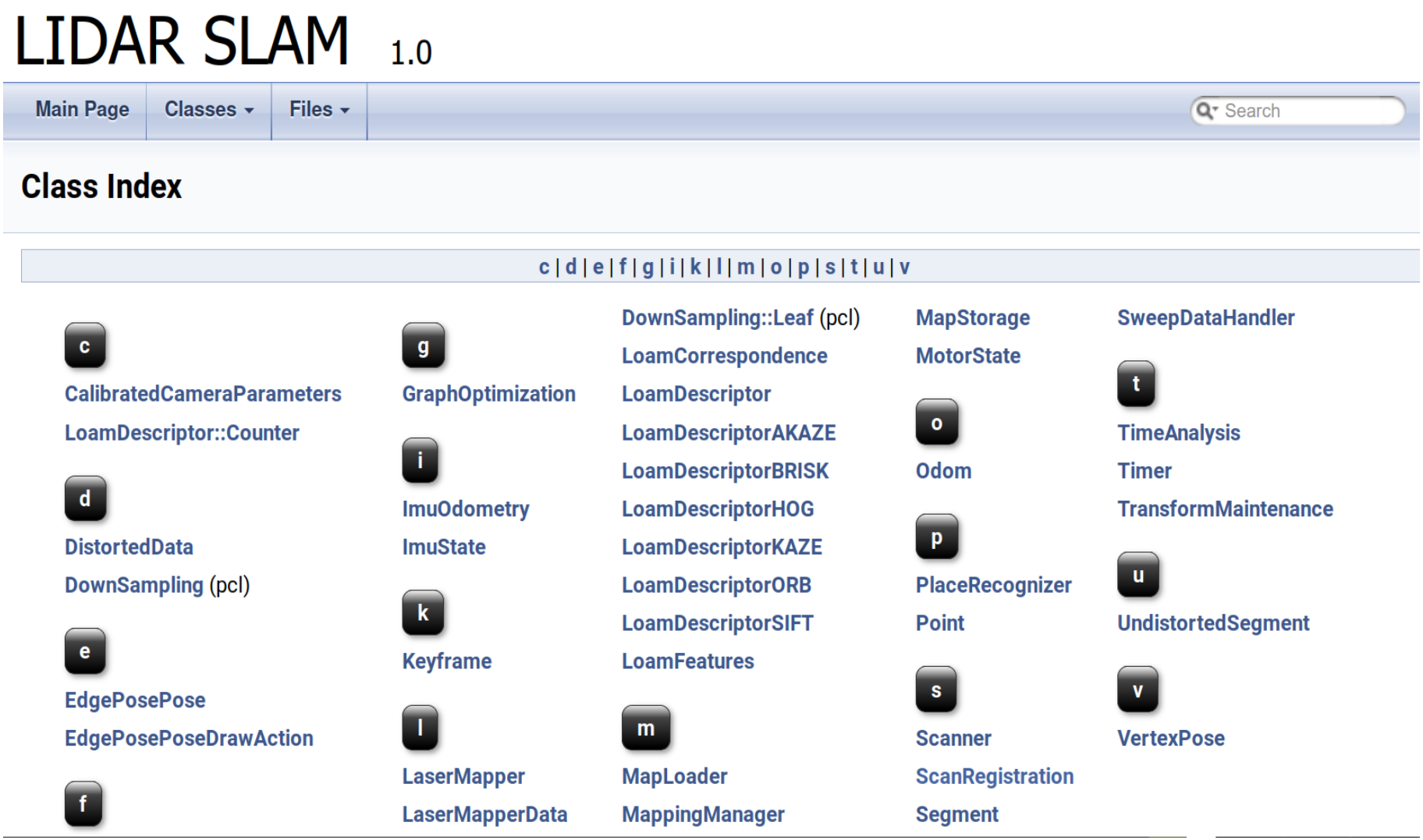
```
rosrun loam_misc scannerReal

roslaunch loam_mapping loamGETbot.launch
```

#Packet Structure: loam_mapping contains the SLAM system. **ScanRegistration** is used to create a 3D scan. **LaserOdometry** estimates the robot motion and undistorts the 3D scan. **MappingManager** inserts the 3D scan into the map. Transform maintenance calculates the robot pose.

The nodes similarityChecker and similarityChecker2 can be used to test the place recognition system. The node simulation can be used to test loop closure. You need to give a name of a saved map in order to run these nodes, which has to be placed inside the loam_misc/results/ folder.

Documentation Screenshots



LIDAR SLAM 1.0

Main Page | Classes | Files | Search

Class Index

c | d | e | f | g | i | k | l | m | o | p | s | t | u | v

c

CalibratedCameraParameters

LoamDescriptor::Counter

d

DistortedData

DownSampling (pcl)

e

EdgePosePose

EdgePosePoseDrawAction

f

g

GraphOptimization

i

ImuOdometry

ImuState

k

Keyframe

l

LaserMapper

LaserMapperData

DownSampling::Leaf (pcl)

LoamCorrespondence

LoamDescriptor

LoamDescriptorAKAZE

LoamDescriptorBRISK

LoamDescriptorHOG

LoamDescriptorKAZE

LoamDescriptorORB

LoamDescriptorSIFT

LoamFeatures

m

MapLoader

MappingManager

MapStorage

MotorState

o

Odom

p

PlaceRecognizer

Point

s

Scanner

ScanRegistration

Segment

SweepDataHandler

t

TimeAnalysis

Timer

TransformMaintenance

u

UndistortedSegment

v

VertexPose

Documentation Screenshots

LIDAR SLAM 1.0

Main Page | Classes | Files | Search

Public Types | Public Member Functions | Public Attributes | List of all members

ScanRegistration Class Reference

Public Types

using **processSweepFunPtr** = std::function< void(const **DistortedData** &)>

Public Member Functions

ScanRegistration (ros::NodeHandle &nh, Features::FeaturesPtr &features, bool &activateIMU, processSweepFunPtr processS=[](const **DistortedData** &){})

void **setProcessSweepFunction** (processSweepFunPtr &processSFun)

void **setLaserMountingOffsets** (const double &roll, const double &pitch, const double &yaw)

void **setCameraCalibratedSettings** (const std::vector< double > &cameraMatrix, const std::vector< double > &distCoeffs, const std::vector< double > &translation, const std::vector< double > &rotation)

Public Attributes

SweepDataHandler **dataHandler**

Detailed Description

Definition at line 41 of file ScanRegistration.h.

Documentation Screenshots

Chapter 1

Main Page

Packages for 3D mapping based on LOAM: Lidar Odometry and Mapping in Real-time #How To Start: GETjag:

```
rosrun loam_misc scannerReal
```

```
roslaunch loam_mapping loam.launch
```

Simulator:

```
roslaunch loam_mapping gazebo.launch
```

```
roslaunch loam_mapping loamSimulator.launch
```

GETbot

```
rosrun loam_misc scannerReal
```

```
roslaunch loam_mapping loamGETbot.launch
```

Documentation Screenshots

Chapter 4

Class Documentation

4.1 CalibratedCameraParameters Struct Reference

```
#include <ScanRegistration.h>
```

Public Attributes

- cv::Mat **cameraMatrix**
- cv::Mat **distCoeffs**
- cv::Mat **translation**
- cv::Mat **rotation**

4.1.1 Detailed Description

A struct to hold all camera parameters such as extrinsic and intrinsic parameters.

Definition at line 24 of file ScanRegistration.h.

The documentation for this struct was generated from the following file:

- include/lidar_slam/ScanRegistration.h