EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mechanical Engineering

A Parametric Dataset Generator For Updating Digital Twins Through 2D LiDAR Perceptual Data

by

P.J. van den Akker

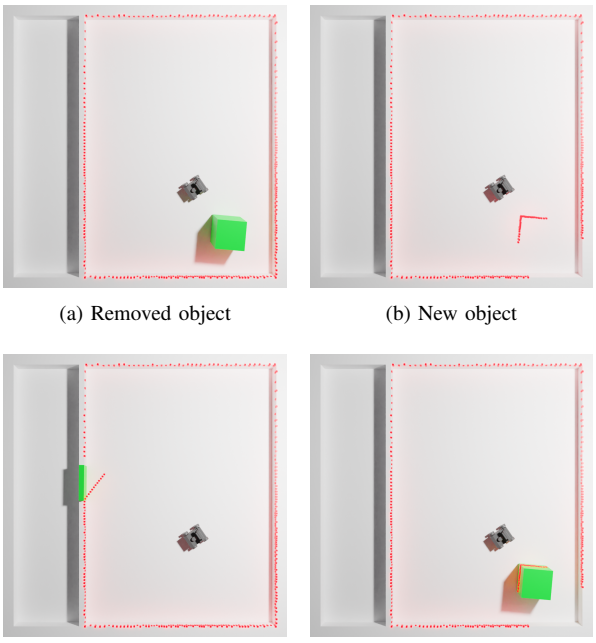
MSc Thesis

Thesis committee		**Graduation**	
Chair:	Elena Torta, dr.	Program:	Artificial Intelligence and Engineering Systems
Member 1:	Pieter Pauwels, dr. ir.	Research group:	Control Systems Technology
Member 2:	Jos Elfring, dr. ir.	Thesis supervisor:	Elena Torta
Advisory member:	Gijs van Rhijn	Date of defense:	26/09/2024
		Student ID:	1758160
		Study load (ECTS):	45
		Track:	High-tech systems and robotics

***Abstract*—This thesis introduces a novel parametric dataset generator designed for training instance segmentation models to update digital twins using 2D LiDAR perceptual data. The generator creates synthetic datasets by creating prior maps derived from simulated digital twins. These digital twins are extracted from sources such as Building Information Models (BIM). The generated 2D maps are overlaid with 2D LiDAR readings and used to train an instance segmentation model (i.e. mask R-CNN) exclusively on synthetic data. Post-training, we evaluate the trained model on a newly made real-world dataset. The results show that the synthetic dataset effectively trains the model to detect changes in real-world environments; however, there is a discrepancy, or domain gap, between the synthetic and real-world data. Additionally, classification remains challenging for objects without prior information due to the limited data provided by 2D LiDAR. To identify regions with insufficient data in the model's predictions, an uncertainty area detection method was implemented. This method identifies regions where the model had multiple conflicting predictions, helping to highlight areas requiring further analysis. These findings underscore the potential of synthetic data generation in training models that can detect mismatches between prior maps and 2D LiDAR detections, for improving the robustness of digital twin updating and robotic navigation systems.**

I. Introduction

For navigation tasks, robots use global path planning to plan routes and avoid obstacles beyond their direct line of sight. This path planning is often performed using 2D gridmaps. These maps can be generated using SLAM (Simultaneous Localization and Mapping), which uses sensor data to create maps. However, this information is limited by what the robot has already directly observed. We can use other sources of information such as Building Information Models (BIM) to supplement direct robot observations. BIM models are 3D models that, in addition to their geometric data, contain information about object types and materials. These models are created for the construction of buildings, making them readily available and a convenient source of building data useful for a robot's operation. Using this existing information, BIM models can serve as a foundation for digital twins, which are virtual representations of real-world environments that can be continuously updated with real-time data. By comparing the information from the robot's sensor and information sources such as BIM models, maps with both geometry and object descriptions (semantics) can be updated. As discussed in [1] these maps are useful because the additional semantic information can help a robot make more informed decisions. For example, if a robot has information it is approaching a door it can plan to open it or navigate more carefully in case it opens, improving safety and efficiency.



(a) Removed object (b) New object

(c) Dynamic object (d) Correct

Figure 1: Examples of robot's observation and prior map. Green objects are present on the prior map, the red dots are the 2D LiDAR detections, representing the real world. (a) The green object on the prior map does not have a robot's detection, indicating it is no longer present in the real world. In (b) The robot detects an object not on the prior map, indicating a new object that should be added to the map. In (c) the prior map shows a door (green), but the robot detects it open, indicating a mismatch. In (d) the prior map and robot's detection overlap, indicating the prior map is correct.

While these maps are useful, using them as a digital twin requires continuous updates. We therefore need to find mismatches between the maps derived from the digital twins and the robot's detections. A mismatch between a map and the sensory data of the robot can lead to navigation errors and inefficient task execution. For example, if a robot without the ability to open doors navigates into a space with a door the map indicates is open while it is closed, it would waste time travelling to the door. An up-to-date map would have prompted the robot to choose an alternative path, avoiding unnecessary travel.

It is therefore crucial to detect these mismatches between the map and the robot's perception. In this thesis, we aim to identify these mismatches using instance segmentation. Instance segmentation is a computer vision task that involves detecting objects within

an image and assigning a unique class (e.g. chair, pillar), bounding box, and precise shape to each *instance* of an object (e.g., chair 1, chair 2, pillar 1). Because instance segmentation is a supervised machine learning algorithm, a large amount of annotated data is necessary [2]. However, manual data collection is time-consuming and expensive, especially if the data should describe a variety of environments. To enable digital twin updates through instance segmentation, we create synthetic datasets generated by computer simulation. More specifically, we create a fully parameterised dataset generator that creates tailored synthetic datasets for specific environments.

The dataset generator creates datasets with examples of different mismatches between the prior map (the map provided before the robot's detection) and the current robot's perceptual information. The generated data set recreates (a combination of) the following possible mismatches:

- **Removed object** where an object is present on the prior map but is not detected by the robot. As depicted in Figure 1a, The green object is modelled in the digital twin but it is not perceived by the LiDAR sensor;
- **New object** where an object is detected by the robot's detection but is not present on the prior map. In Figure 1b the red robot's detection has no overlapping green prior map object, indicating a new object;
- **Dynamic object** where the object moves. A door, for example, is not expected to match the prior map precisely depicted in Figure 1c.

A. Related Work

Our proposed method of updating prior maps using synthetic data and instance segmentation is to the best of our knowledge novel and thus lacks direct comparison to existing research. Existing research however can assist with the tasks of synthetic data generation and mismatch detection. We therefore briefly review some related works.

1) SLAM: This thesis aims to update maps with robot's detections, this has significant overlap with Simultaneous Localisation and Mapping (SLAM) [3], even though we only focus on the Mapping part. OBVI-slam [4] creates a local real-time map similar to traditional SLAM, but tracks changes over a longer time using an uncertainty-aware map. A major limitation of this method is the training data. For object tracking, a model needs to be trained with a sufficient amount of usually hand-labelled data that accurately represents the environment. Additionally, this method uses (multiple) RGB(d) cameras, necessitating compute power or connectivity that may not be available to all robots.

Other methods [5] [6] detect map changes using object tracking models like You Only Look Once (YOLO) [7]. However, none of these methods enriches their world understanding by obtaining environmental information from external sources such as BIM models, and need to explore an entire environment before being able to perform global mapping tasks.

2) Computer vision based mismatch models: Detecting mismatches in maps can be achieved through the use of instance segmentation models, An instance segmentation model is an Artificial Neural Network (ANN) that can detect not only a bounding box and a label but also the shape of individual objects. This shape is referred to as the (segmentation) mask. A key feature of the instance segmentation model is the identification of object instances. This is useful because if we have an image with two chairs, the model outputs an individual detection for each chair. We do not explain the underlying functioning of instance segmentation models, but only the use case, for a more in-depth see [8].

Another advantage of instance segmentation models is the possibility of shape completion [9]. An example is a 2D LiDAR image where the LiDAR can see one side of a pillar. Shape completion could subsequently predict the shape of the entire object, based on incomplete data.

One of the best-known instance segmentation models is the Mask Region-Based Convolutional Neural Network (Mask R-CNN) model [10]. This model is well-documented and easy to implement, making it a popular choice for practical applications. While other instance segmentation models exist like the various iterations of YOLO [7]. We prefer Mask R-CNN for its ubiquity and ease of use.

3) Synthetic data: Training instance segmentation models requires a lot of (human) annotated data. This can be time-consuming and expensive. In contrast, synthetic data uses computer simulations to generate data. This means that large amounts of data can be generated relatively cheaply. Moreover, as mentioned in [11] synthetic data has a lot of other advantages: perfect annotation, as opposed to imperfect human annotation, easy variation, privacy, and the use of compute time instead of human time.

An important term in synthetic data generation is the "domain gap" also called the reality gap. This gap de-

scribes the difference between reality and the simulated worlds. One of the ways to compensate for the domain gap issue is so-called "domain randomisation" [12]. Domain randomisation attempts to bridge the domain gap by introducing variation in lighting, camera angle, or textures. This creates so many variations that reality seems like another variation. Synthetic data can perform so well that it can outperform human annotated labelled data [13] in object recognition applications.

While our application is focused on 2D maps, synthetic data research generally focuses on 3D worlds, some work has however been done on synthetic data for 2D worlds. For example, a 2D grid map dataset has been created for robot navigation [14]. Which is a large dataset that consists of many maps and their most efficient paths, but the data does not contain information to detect mismatches between prior maps and current 2D LiDAR detections. Another dataset was created labelling room types for the FloorplanNet model [15]. Other approaches for data generation focus on using previously labelled 3D datasets and converting them to 2D grid maps [16], this has the advantage of producing accurate grid maps but does contain a lot of variation. None of these methods however are suitable for our applications because of the lack of customizability and LiDAR detections.

B. Research question

As mentioned in subsection I-A, randomisation for synthetic data generation plays a key role in the performance of computer vision models. Therefore, we create a fully parameterised synthetic dataset generator to find mismatches between a prior map and a 2D-LiDAR robot's detection. While 2D map datasets exist, they are typically created by taking a 3D model and cutting it into 2D slices, not truly creating new data. None add overlay this data with 2D LiDAR to identify mismatches. Our research makes a unique contribution in two ways:

- Creating a randomised and parameterised 2D dataset generator with extensive randomisation, including the geometry and layout of the rooms and the objects within them.
- Training an instance segmentation model with this data to detect, classify, and highlight mismatches between the prior map and the robot's detection in real time.

Training on a randomised and parameterised dataset should result in a more robust machine-learning model that can adapt to a greater variety of real-world data. The abundance and variability of synthetic data help to overcome the limitations of smaller real-life datasets, theoretically enhancing the model's robustness.

Moreover, the customizability of the dataset allows for tailored dataset generation for specific environments. For example, if it is known that all pillars in an environment are of a certain size or shape, the dataset can be configured to generate only those types of pillars. Additionally, the LiDAR height can be set to match a specific robot. This enables the creation of more customised datasets.

These premises lead us to the formulation of the research questions:

1) How can we create a parametric dataset generator that emulates the data a robot can retrieve from a building's digital twin and its environment?
2) How can a model trained on a synthetic dataset detect mismatches between environmental data queried from the digital twin and runtime robot's perception?
3) How is the performance of the mismatch detection model influenced by the generation of a dataset tailored to a specific environment?

II. Methods

A. Overview

Our goal is to identify discrepancies between two data sources: a prior map (Figure 2a), and real-world LiDAR detection (Figure 2b). This is achieved by feeding a composite image that overlays the prior map and LiDAR detections (Figure 2c) to an instance segmentation model. Using this composite image, an instance segmentation model can classify mismatches.

By combining the LiDAR detections and the prior map into a single image, the prior information can be compared with the robot's detection. For example, in (Figure 2c) the blue table is present in the prior map, however, it has no overlapping real-world LiDAR detection (red dots), indicating that it is not present in the real world. Conversely, in the top left of Figure 2c, the LiDAR detects the contour of an object not present on the prior map, indicating that this is a new object. To detect these mismatches we train an instance segmentation model.

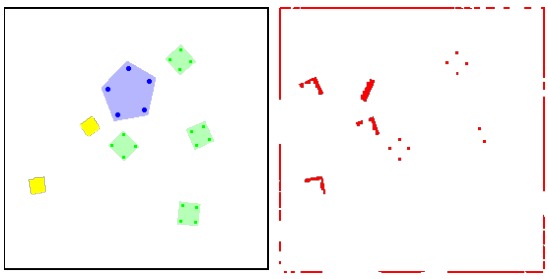
For the generation of the data and training and verification of the model we make the following assumptions:

- We only consider: pillars, walls, doors and chairs as possible objects.
- All environments are inside, in an enclosed room.
- The robot obtained near-perfect localisation.

B. Synthetic data generation

Our data generation starts by creating a parameterised 3D world in Blender. This generated 3D world can be considered the "real world". From this 3D world, we extract 2D LiDAR information and prior maps. Creating a 3D world is necessary because, even though we only create prior maps and detections for 2D LiDAR applications, the height of the LiDAR influences what the output looks like. To illustrate this consider Figure 3, we can observe the same room with two LiDAR detections, located at the same x and y position, one LiDAR is located lower on the z-axis (height) (Figure 3a) and one higher up (Figure 3c), while the LiDAR is located in the same position in the x and y position the LiDAR outputs (Figure 3b and Figure 3d) are completely different. Because we derive our dataset from a 3D world we have the LiDAR height as a parameter, allowing the user to create a dataset for robots of specific heights.

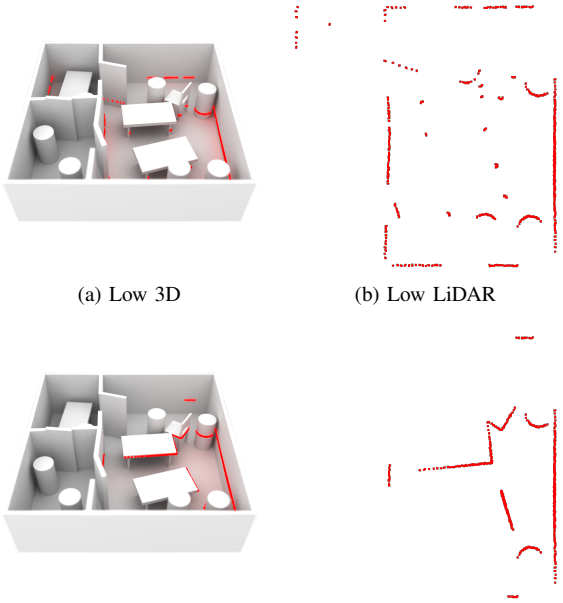
Besides LiDAR height, our data generation method is also designed to create datasets tailored to various environments. For instance, if a robot navigates an environment with a specific chair type, we can generate a synthetic dataset featuring variations of that chair type in different positions, orientations, and quantities. We can customise our method using our fully parameterised, randomised dataset generator, which allows



(a) Prior map (b) LiDAR detection

(c) Combined

Figure 2: Examples showing: (a) the prior map with knowledge about the environment before detection. (b) The LiDAR detection representing the real world. (c) Combining real-world and prior information.



(a) Low 3D (b) Low LiDAR

(c) High 3D (d) High LiDAR

Figure 3: The 3D world with the LiDAR at different heights. In (a) and (c) the LiDAR sensors are located at different heights. In (d) and (b) the difference between LiDAR placement heights is shown.

users to control object types, shapes, and room layouts. We create these rooms and objects using the geometry nodes tool in Blender. This tool allows for the parametric generation of 3D models with algorithms and parameters using flow graphs, instead of manually designing or programming them.

1) Synthetic Data Generation Parametrization: Parametrization involves a set of adjustable parameters that control aspects of a model's geometry. Instead of creating a new object individually, a base model is designed with parameters that can be modified to generate different variations. This results in a practically infinite number of unique 3D models, as showcased in Figure 4, where multiple room variations are displayed and generated from the same parametric model. Each of these variations is an output of the same parameter-based model with user-defined object dimensions, object types and room layouts. The key here is that while the layouts are randomised, the user still has fine control of the parameters for the distribution and object types. By defining parameters such as chair size, wall density or a random seed we can generate a practically infinitely large dataset of mismatch examples.

Another reason we created a parameterised dataset is the previously discussed (subsection I-A) domain gap:

the differences between the real world and synthetic data. We try to compensate for this gap by adding randomisation with varying object sizes, room layouts, LiDAR height and other parameters. Thus we can create a more varied and robust dataset.

Before generating data, the parameters defining the limits of various objects must be set. For example, consider the creation of a chair: some parameters that are used to control a chair model are width, height, length, leg radius and backrest height. Every time a new room is generated new values are chosen uniformly from a user-defined range. For example, chair leg width in the range: $0.1\text{ m} \leq \text{leg width} \leq 0.2\text{ m}$. This way, every time a new room is generated a new unique chair is generated, creating a new unique layout with unique objects every time. While more objects could be added, we only add walls, doors, chairs and pillars to a room. See appendix A for tables containing all the parameters.

A limitation of 2D LiDAR is the sparsity of the information it provides, which we address by embedding prior information from external sources into our data. This is done by creating an image where colours represent different objects as shown in Figure 2c, where LiDAR detections are red dots, tables are blue, pillars yellow, chairs green, walls black, and the background white. These objects are extracted from semantically rich sources like BIM models, enhancing the model's object classification capability.

2) Input Image Generation: For the training of instance segmentation models, we require bounding boxes, labels, and masks for each object in the ground truth. This ground truth is generated during the data creation process. However, this is not possible directly in Blender. As a result, we create an intermediate image where each pixel corresponds to an object ID.

After selecting the parametric values the data generation can start, the data generation follows the steps shown in Figure 5:
1. A new empty room is generated, consisting of inner and outer walls. These walls are randomly placed, giving a different room layout in every image. More detail on the wall generation is provided in appendix A. **2.** Each wall section has a random chance of generating a door. The door position in the wall, width and orientation of the doors are randomly generated. For more on door generation see appendix A. **3.** Points are distributed in the room. **4.** Each point is assigned an object, and the maximum dimensions of these objects are determined, for every point, the distance to the nearest object is calculated, when the closest object is closer than the width or depth of an object, this point is deleted. For more information on object distribution see: appendix A. **5.** Chairs and pillars are placed in the scene. All these objects are randomly generated based on parameters defined by the user. Every new room generation will therefore contain unique objects. **6.** The simulated LiDAR in a random location in the scene. The LiDAR is discussed in more detail in subsubsection II-B4 and subsection E. **7.** Objects have a set probability of being ignored by the LiDAR, this simulates that this object is present on the prior map, but not detected by the LiDAR. **8.** The LiDAR scan is saved as an image. **9.** The area that the LiDAR can detect is saved as a separate image. This is the filled-in shape of the LiDAR points. **10.** The objects that are outside of the LiDAR detection range are removed and some prior shapes (coloured shapes) are removed from the image, simulating they are new objects and not present on the prior map.

Figure 4: Examples of randomly generated rooms, with variations in object sizes, types and layout. LiDAR detections in red.

The images are generated using Blender's real-time Extra Easy Virtual Environment Engine (EEVEE) renderer [17]. Allowing for fast image generation on our lab computer of approximately 3 images/second (Intel I7 14700k, RTX 4080, 32 GB).

3) Ground Truth Generation: To train the instance segmentation model we require a ground truth. In Figure 6 we can observe an input image (Figure 6a) and its corresponding ground truth (Figure 6b), which

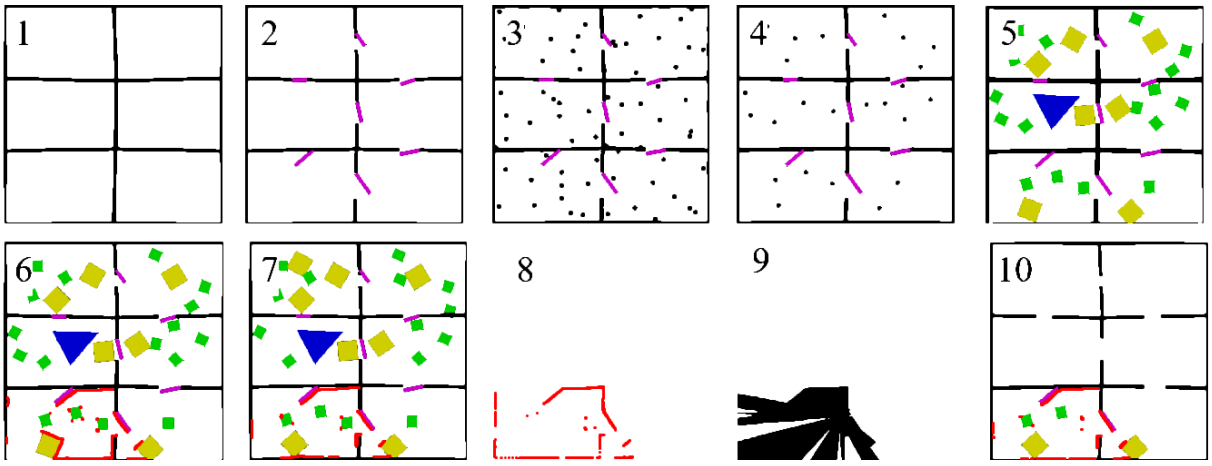


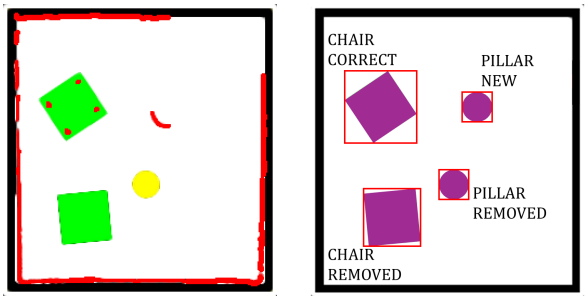
Figure 5: Simplified steps on how the synthetic data is generated. 1) A room is generated. 2) Doors are placed randomly. 3) points are randomly placed. 4) Points that are within a specified range of others are removed. 5) Objects are placed with colours corresponding to their type, green: chairs, yellow: pillars, blue: tables. 6) A LiDAR is placed in the scene. 7) Some objects are set to be ignored by the LiDAR, simulating they are removed from the scene. 8) An image showing just the LiDAR is captured. 9) An image of the area covered by the LiDAR is captured, this image is used to only show objects that are in the LiDAR detection range. 10) The objects that are outside the range of the LiDAR are removed.

consists of segmentation masks, bounding boxes and mismatch labels.

We define three types of mismatch labels: `correct`, `removed` or `new`. A `correct` label indicates that the prior map and detection match. `removed` signifies that an object is on the prior map but not detected by LiDAR, while `new` means the object is detected by LiDAR but not on the prior map.

To identify unique objects we assign them unique IDs during data generation. We distinguish between a class label and an object ID. For example, if multiple `chairs correct` are in a scene, each chair receives the same class label but a unique ID. We create this unique identifier using a two-part format ($xxyyy$), where xx is the class ID (e.g., 02 for `chair correct` and 03 for `chairs removed`). Then yyy is the instance number per instance, this could be 02001 for `chair correct` one, 02002 for `chair correct` two, 03001 for `chair removed` one, etc.

During data generation, the ground truth is labelled using the *bpycv* [18] python library, a set of utilities for segmentation in Blender. We use this library by tagging objects during data generation with a class. We determine the ratios of these classes with a user-defined ratio. This ratio is selected to be either `removed` or `new` with the remainder being `correct`. Objects



(a) Input (b) Ground truth

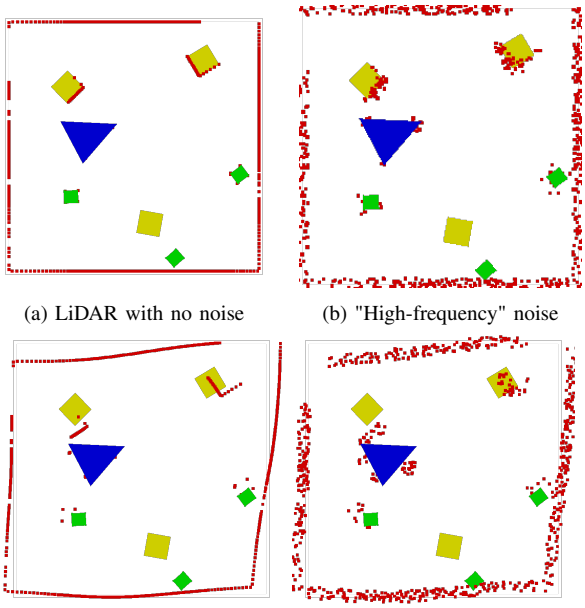
Figure 6: Example of input and ground truth. With (a) input image containing, red: LiDAR, green: chair, yellow: pillar and (a) ground truth with red: bounding box, purple: segmentation mask, and labels.

set as `removed` during data generation are set to be invisible to the LiDAR but visible on the prior map. The objects tagged as `New`, are set to be visible on the LiDAR detection but not on the prior map. The objects that did not receive a `new` or `removed` tag are set to the `correct` tag. With these tags, *bpycv* automatically outputs an array matching the input image's size. Each point in this $n \times m$ array is assigned a unique instance ID. This is effectively an image with pixels replaced by the instance IDs with every pixel with no object being the background class.

We now have an image where each pixel has an associated object class and ID. However, the instance segmentation model needs a mask, bounding box and label to train. Therefore we convert the matrix that results from the data generation into these data forms. This conversion occurs before training:

- **Bounding boxes:** For each object (instance) in an image, a bounding box is generated. This is achieved by finding the outermost coordinates of each instance in the image masks. This results in a list of z bounding boxes, where z is the number of instances.
- **Instance IDs:** Another list of z elements is created, each containing the corresponding instance ID.
- **Boolean mask matrix:** The image masks are converted into a 3D matrix with dimensions $z \times n \times m$. Within each of the z layers a value of "True" indicates the pixel belongs to that instance, with "False" indicating it doesn't. This results in a matrix where each instance has a unique mask made of True/False values, defining the shape of each instance. This is essentially a 3D multihot encoded array.

4) LiDAR Simulation: The simulation of the 2D LiDAR is a critical part of the data generation, as it simulates real-world interaction. The LiDAR data is generated as follows: **1.** The simulated LiDAR is placed randomly in the room, from this position rays are cast in a circular pattern (Figure 7a) every object it hits becomes a square block of user-defined size in 3D space, indicating a LiDAR hit. **2.** Every block has an XYZ coordinate, noise is only added in the XY directions. To simulate High-Frequency noise, uniform noise is added on top of the LiDAR data, the maximum distance this noise may be offset from the true location is defined by the user, for example, if the user defines a maximum offset distance of 0.1 m then any given LiDAR block will have a maximum distance from the true hit location of 0.1 m. (Figure 7b). **3.** In addition to High-Frequency sensor noise, we found noise and inaccuracies caused by robot localisation inaccuracies and other LiDAR scanning artefacts. To get an approximation of these noise types, we add a Perlin Noise [19] texture to the position of each LiDAR point. Perlin noise is a gradient noise function that generates smooth continuous patterns. For more details on the LiDAR data generation see appendix E. **4.** Combine the Low-Frequency and High-Frequency noise types into the final LiDAR image (Figure 7d).



(a) LiDAR with no noise (b) "High-frequency" noise

(c) "Low-frequency" noise (d) Combined noises

Figure 7: An exaggerated example of different types of LiDAR noise.

C. Uncertainty Area Detection

A limitation when using 2d LiDAR data is that there isn't always enough data to classify the object type. We mitigate a part of this problem by using data from prior data from a building digital twin. However, when a new object is detected by the LiDAR but not present on the prior map, there isn't always enough data to classify it. We found that the model frequently outputs multiple high-confidence labels for the same object in this situation. Consider Figure 8, at the bottom of this image we can observe two points of the LiDAR detection, these points are a chair partially occluded by a wall showing only two of the four legs. Because the model can only see two legs it cannot know which way the chair is oriented, in this case, the model outputs two high-confidence predictions. One or both of these predictions are wrong. We know that when the model does not have enough information to make a good guess, it makes multiple high-confidence predictions for the same object. Therefore, we want to determine where the model outputs multiple confident predictions for the same object.

The most obvious method to detect the confident overlap areas would be to detect where any overlap is and combine these areas. This is however not completely suitable because some minor overlap may mean that two objects are close together. We, therefore,

want to threshold a minimum overlap percentage. An additional issue is that occasionally we have more than two overlapping bounding boxes. To handle this, we extend the more common Intersection over Union (IoU) from

$$IoU = \frac{A_1 \cap A_2}{A_1 \cup A_2}, \tag{1}$$

to

$$IoU_{multiple} = \frac{\sum_{i \neq j} A_i \cap A_j}{\sum_{i \neq j} A_i \cup A_j}, \tag{2}$$

where A is the set of all bounding boxes. We then consider these predictions to be uncertain if $IoU_{multiple}$ higher than 0.2. If this overlap threshold is reached we combine all the overlapping bounding boxes to create an *area of uncertainty*. We call all the bounding box labels contained in an *area of uncertainty* the hypothesis.

We want to know whether the correct class is included in that *area of uncertainty*. We therefore calculate a correct hypothesis rate. Let $\mathcal{P}$ be the set of all predictions p, and the ground truth g, we say that there is a correct hypothesis if:

$$\exists p \in \mathcal{P} \text{ such that } p = g. \tag{3}$$

In other words, if for the ground truth, there is any overlapping correct hypothesis label, then the hypothesis is correct.

D. Real-world dataset

We need a real-world dataset to verify whether the generated data represents the real world and whether the domain randomisation is sufficient to capture real-world data. This dataset is not used to train models but to test their performance.

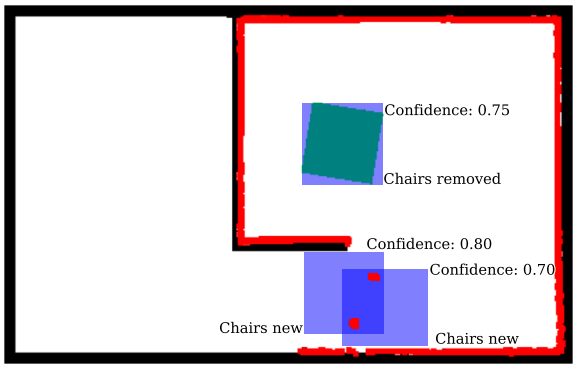


Figure 8: Example of uncertainty in model output. At the bottom, note two LiDAR points that could be a new chair in either direction, outputting two confident but conflicting masks.

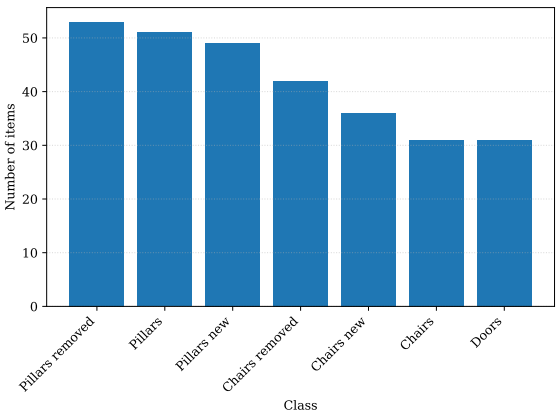
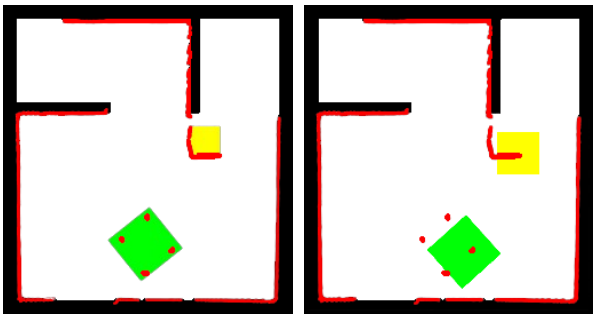


Figure 9: Number of objects of each class in the real-world dataset.



(a) (b)

Figure 10: Two variations for the same real-world test environment detection. With (a) containing detections where the LiDAR detection (red) and the prior map match. In (b) the objects are moved on the LiDAR detection compared to the prior map.

The dataset is generated in a controlled lab environment using a modified ROSbot2.0 [20] equipped with an RPLiDAR A2, the ROSbot is modified by removing the RGBD camera and its bracket to give a unobstructed LiDAR scanning image.

This dataset consists of multiple images recorded from four unique room layouts (Figure 12). For each room layout, 15 scenes are recorded. In each of these scenes, we placed objects randomly in a room, in figure Figure 11 an example of a test setup is shown. The objects placed in the room include boxes, round pillars, chairs and doors. For each scene, LiDAR data is recorded. After the data is recorded the LiDAR detections are overlaid with the prior maps manually.

As discussed previously the data is also embedded with semantic information. This is accomplished by giving

the prior map shapes with different colours indicating different objects as seen in the image. These shapes were added manually after recording the real-world LiDAR information in the lab. The distribution of items class items is shown in Figure 9.

As shown in Figure 10, this allows for multiple variations per data recording. For each of the data recordings, two variations on the prior map are made. The LiDAR detections are the same for the two variations, but the assumed object positions in the prior information change. In this way, we go from 60 recorded LiDAR detections to a total of 120 images.

After the creation of these images, they need to be labelled. This labelling is done using Make Sense [21]; a data labelling tool. The labelling is accomplished by drawing polygons around objects and assigning them the appropriate semantic classes.

Note that this is a limited dataset, intended to indicate real-world performance and should not be considered exhaustive or definitive.

E. Computer Vision Model

We use an unmodified version of Mask R-CNN in this thesis. We use Mask R-CNN for its stability, documentation and ease of implementation. We use the built-in PyTorch Mask R-CNN implementation [22]. As we use an unmodified Mask R-CNN implementation we will not explain the model functioning here, please refer to [10] for further details. We use the PyTorch ADAM [23] optimiser. We conducted manual hyper-parameter tuning to determine the hyper-parameters as depicted in Table I, which we use in all experiments. We initiate weights with the PyTorch default [22] and set the random seed to 42 for consistency. All parameters not specifically mentioned are left as the PyTorch default values.

For more details please refer to our repository [24].

F. Metrics

To evaluate the performance of the instance segmentation model we require metrics. These metrics are not only intended to measure performance but also to optimise the models.

While training the mask R-CNN model, the loss as described in [10] is computed. While the loss metric is very suitable for training, it can vary widely between different datasets and does not account for model robustness. Therefore, we use the commonly used metric, the mean Average Precision (mAP) [25]. This metric measures Average Precision across various thresholds of Intersection over Union, providing a consistent and robust means to evaluate and compare different models. A higher mAP indicates better performance in detecting and classifying objects.

Table I: Model hyperparameters used for experiments.

Epochs	Lr	$\beta 1$	$\beta 2$	ϵ	W-Decay
10	$1e^{-4}$	0.900	0.999	$1e^{-8}$	$5e^{-4}$

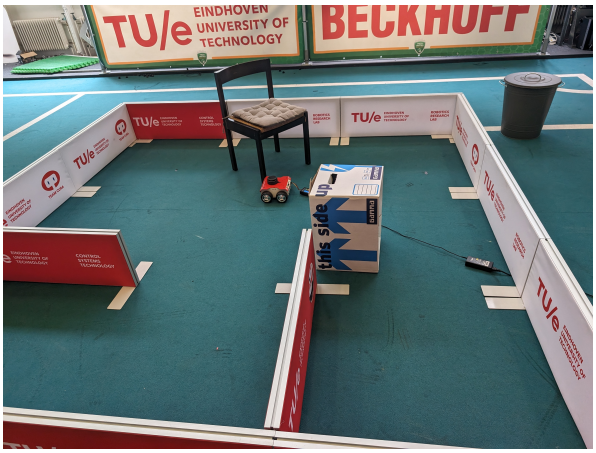
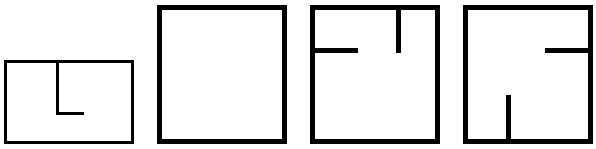


Figure 11: Lab test setup, this is the room layout from Figure 11c. The red robot in the centre is where the LiDAR data is captured from.



(a) (b) (c) (d)

Figure 12: The four different room layouts for the real-world dataset.

The mAP is very suitable for comparing different models, it however does not give an intuitive feeling of how well the model works in the real world. For that purpose, we use the Micro Average True Positive Rate (TPR) also known as sensitivity or recall:

$$\text{Micro-average TPR} = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} (TP_k + FN_k)}, \quad (4)$$

where K is the set of all classes k, TP_k is the True Positives (TP) per class and FP_k is the False Positive (FP) per class. This metric measures the proportion of actual positive instances correctly identified by the model.

To calculate the TPR we need to find which model prediction corresponds to which ground truth label. We achieve this by calculating the overlap between ground truth and prediction labels. Specifically, we accomplish this by calculating the IoU of each prediction with each ground truth mask, with a $IoU > 0.5$ indicating overlap. This method is similar to the approach described in [26].

G. Experimental Setup

As discussed in subsection I-B, we want to determine whether a model trained exclusively on synthetic data can detect mismatches between environmental data queried from the digital twin and the robot's perception Additionally, we want to explore how this performance is influenced by the generation of datasets tailored to a specific environment. To determine whether this is possible we try to answer the following questions:

- Is the performance of a model trained on synthetic data comparable to the same model evaluated on real-world data?
- Is a model trained on a customised dataset better than a model trained on a general dataset?
- How do the parameters of the data generator influence the model performance?

All experiments use Mask R-CNN instance segmentation models trained exclusively on synthetic data, using the same parameters as mentioned in Table I and verified on the real-world dataset subsection II-D) and a new synthetic dataset created with the parameters from dataset I (Table IX), unless otherwise specified.

In the experiments, all generated images are 280×280 pixels and every training dataset consists of 4000 images with an 80/20 train/validation split.

1) Outline experiment 1: To determine whether a model trained on synthetic data works similarly on real-world data as it does on synthetic data, we train a model on synthetic data and test it on both newly generated synthetic data and the real-world dataset. The synthetic data generation method adequately represents the real world if the performance on both datasets is comparable. Conversely, if the model performance is worse than the real-world dataset this indicates a domain gap, where the synthetic data doesn't fully capture the real world.

2) Outline experiment 2: We test this by training two models on two unique datasets, where in 1) we manually pick the parameters for the dataset to represent the types of objects and sensor heights we know are in the real-world dataset, and in 2) we pick parameters in a wide range changing the robot sensor height, object types and sizes to be as broad as possible. We then compare the performance of these two models and evaluate the impact.

3) Outline experiment 3: We wish to determine which parameters influence the model performance. Most of the parameters we define in our data generation are a range. For example, we vary the width of a chair between (0.5 m-1 m). We want to discover how much this randomisation influences the model performance. We therefore perform ablation experiments where we set one or more parameters to fixed values to determine their importance on model performance. on the following parameters: 1) LiDAR height, 2) LiDAR low-frequency noise, 3) LiDAR high-frequency noise, 4) Chair Width, 5) Pillar Width and 6) a combination of Chair Width, Pillar Width and LiDAR block size, where the LiDAR block size is the pixel size of the LiDAR detection. The parameters we use for the ablation experiments can be found in Table VIII.

III. Results

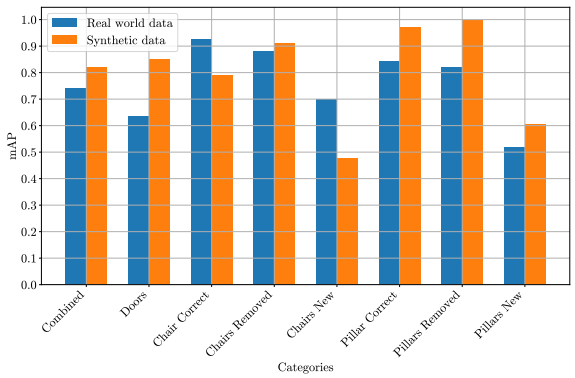
In this section, we present the outcomes of our experiments. We test our method by comparing it to the real-world dataset and a new synthetic dataset by performing the experiments mentioned in subsection II-G.

A. Experiment 1 Results

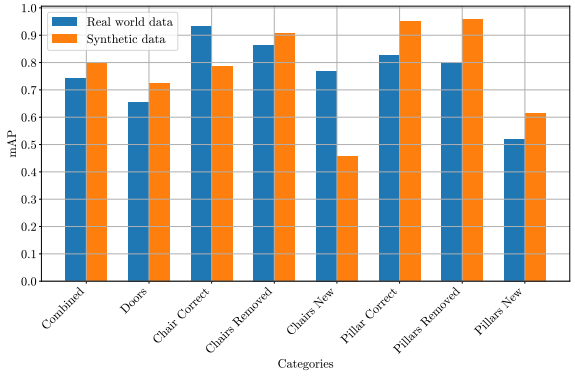
We perform the experiment described in subsubsection II-G1. The parameters for this dataset can be found in Table IX. We use the mAP to compare the performance of the bounding box and the segmentation mask. We can observe in Figure 13a and Figure 13b that while the model generalises to the real world, the model performs better on synthetic data. This is most evident in the Combined column in this figure, where we see a significant difference between the real-world data and the synthetic data. In this column, the bounding box mAP is 0.74 for the real-world data and 0.82 for the synthetic data. From this, it can be concluded that while the real-world performance is decent there exists a domain gap between the synthetic and the real-world data. In addition to this, we can observe that Figure 13a and Figure 13b have very similar performance metrics, we therefore from now on only show the bounding box results. For more examples of the model output see Figure 22.

When examining the per-class performance, the figures show that the `chairs new` and the `pillars new` classes are challenging for the model to capture.

In addition to the mAP we evaluate our model parameters (Table IX) on different metrics to improve our understanding of real-world performance. We observe



(a) Bounding box performance.



(b) Segmentation masks performance.

Figure 13: The mAP for the bounding boxes and segmentation masks for the class categories combined and separated per class for the real-world and synthetic data. Higher is better.

that our model correctly predicts 80% of the labels. Another important metric is how well the model predicts masks. For this we calculate the mask IoU, this gives a good indication of how well the predicted and ground truth masks overlap. Interestingly, the model performs better on real-world data than on synthetic data. We theorise this is caused by synthetic data creating more difficult situations, particularly with the `chairs new` class which we can observe in Figure 13 the model struggles with. We can test if this is the case by calculating the mask IoU again, but this time without the `chair new` class. This results in the IoU on the real-world dataset increasing slightly from 0.86 to 0.87, while on the synthetic data, it increases more significantly from 0.78 to 0.85 confirming the hypothesis. Finally, as mentioned in subsection II-C, we also attempt to detect areas of uncertainty for the model. To determine whether the model predicts the correct class, we use the hypothesis TPR metric. As shown in Table II, the model predicts these quite well, achieving 0.93 TPR for the real-world data and 0.97 for the synthetic data.

Table II: Evaluation of model I (Table IX) on micro-average True Positive Rate (TPR) and mask Intersection over Union (IoU), on real-world and synthetic datasets. Higher is better.

Metric	TPR	Mask IoU	Hyp. TPR
Real world data	0.80	0.86	0.93
Synthetic Data	0.85	0.78	0.97

A benefit of using an instance segmentation model is shape completion. Figure 14 demonstrates this shape completion where a pillar visible as half a circle at the bottom of Figure 14a is converted to a full circle in Figure 14c. Demonstrating the instance segmentation model completes the shape from an incomplete input.

B. Experiment 2 Results

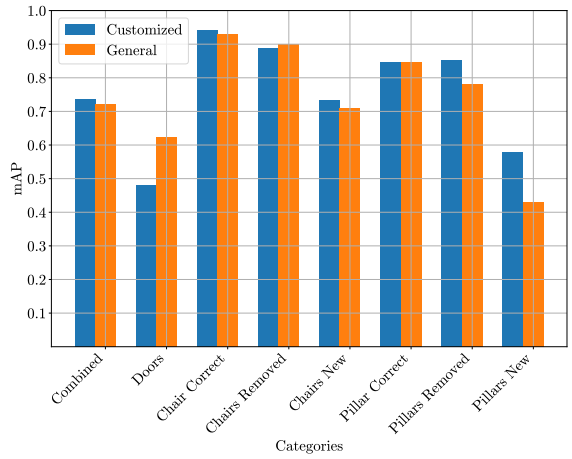
As discussed in subsubsection II-G2 we compare two models: a customised dataset, and a generalised dataset (Table IX for parameters). In Figure 15, it can be observed that there is only a small difference between the customised dataset and the general one. The real-world dataset (Figure 15a) has inconsistent results, with slightly better results with the customised data for most classes except the `doors` and `pillar new` classes. The results become more consistent but still minor when testing on the synthetic dataset (Figure 15b).

C. Experiment 3 Results

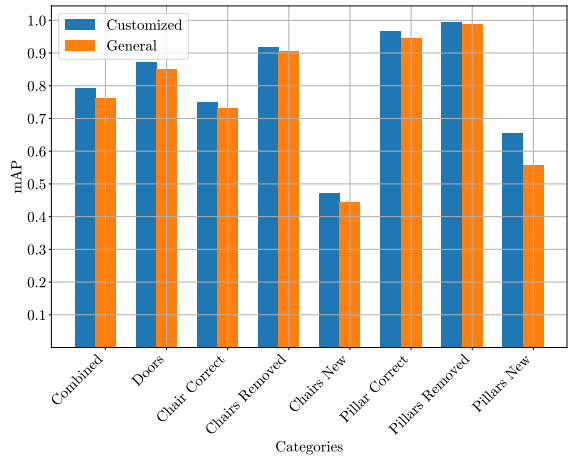
In this experiment, we perform an ablation study on the synthetic data generation parameters to assess their impact on model performance as discussed in subsubsection II-G3. The parameters for this experiment can be found in Table VIII. We can examine the results in Figure 16a and Figure 16b. We ablate over the two

(a) Input (b) Ground Truth (c) Model Output

Figure 14: Comparison of Input, Ground Truth Mask, and Model Output Mask. Note that the pillar on the bottom of the image (a) is only detected from one side by the LiDAR (red outline), but the model completes the circular shape (c).



(a) Tested on real-world data.



(b) Tested on synthetic data.

Figure 15: Figures each comparing the mAP for the bounding boxes between two models tested on different datasets: a general dataset with diverse parameters and more variation, and a customised dataset with parameters tuned for the test dataset, tested on the synthetic and real-world datasets. Higher is better.

types of noise we introduced in subsubsection II-B4. We can observe that the low-frequency noise does not have a significant effect, high-frequency noise seems important for specifically the `chair new` class, the Pillar Width and Chair Size parameters do not significantly contribute to the model performance in the real-world data, with only the `pillars new` class performing worse with a Fixed Pillar width. Interestingly, when we fix the values for Chair Width, Pillar Width *and* LiDAR block size, performance decreases significantly, especially in the `new classes`.

The performance difference while testing on the real-world dataset (Figure 16a) and the synthetic dataset (Figure 16b) are equivalent to the results in experiment 1 and show the same per parameter performance impact except for the `doors` class where no parameters seem to impact the synthetic performance significantly, while the doors class is impacted significantly by multiple parameters in the real-world.

IV. Discussion

This thesis aims to prove that a parameterised synthetic data generator can be used to train an instance segmentation model to detect mismatches between a prior map and a robot's detections. Therefore, the experiments aim to evaluate the performance of using a Mask R-CNN model trained on our novel parametric data generation method.

Experiment 1 shows that a Mask R-CNN model trained on synthetic data can generalise to the real world. However, a performance gap exists between the real world and the synthetic data seems to exist. This performance difference can be attributed to the "domain gap" between the synthetic and real-world data. Additionally, the model performs worse for detecting new objects, where no prior information is present. While we believe the domain gap can still be made smaller by trying more parameter combinations in the dataset generator there is a limitation on what can be done with only 2D LiDAR detections when there is no prior information.

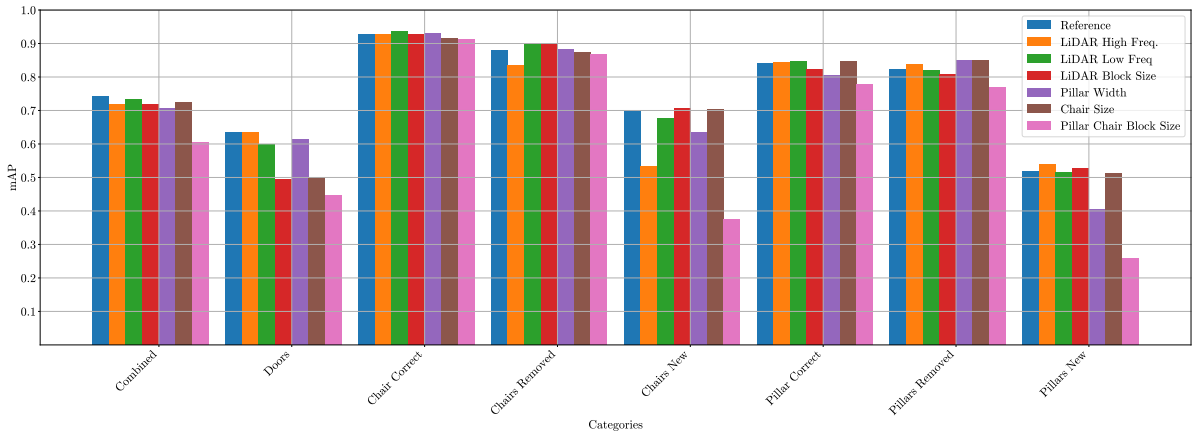
We try to mitigate these issues by detecting areas of uncertainty. In our experiments, we show that the uncertainty area prediction method shows potential, and the model frequently includes the correct label even if it includes multiple predictions for the same object. Future research could focus on using other sensor types to confirm these hypotheses.

This experiment also shows that the segmentation mask IoU unexpectedly performs worse on the synthetic data compared to the real-world data. We attribute this to the synthetic dataset presenting more challenging situations and layouts than the real-world dataset. This is primarily evident in the `chair new` class, where the model struggles to predict the shapes of chairs placed close together, a scenario that is uncommon in the real-world dataset. We confirm that the `chair new` class causes this by removing the `chair new` class from the evaluation. This shows that indeed this class is the reason for the discrepancy. This highlights the need for a more extensive real-world dataset in further research.

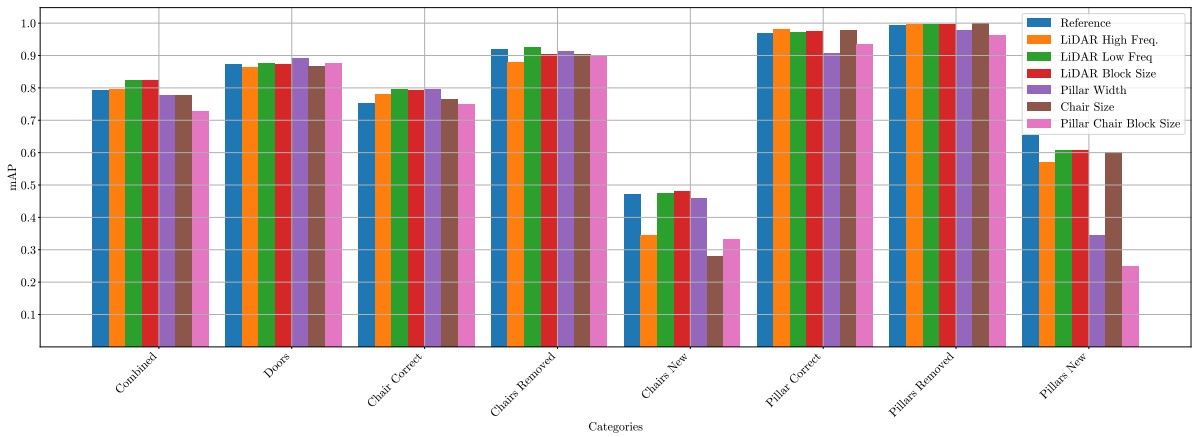
Additionally, we show that instance segmentation has the benefit of shape completion, where an object shape not fully visible with a LiDAR can be completed. This could be useful for robotic path planning, where an object not fully visible is still mapped entirely.

In experiment 2, we compare a customised model against a more general model. The results show that in our experiments there is no significant improvement when customising a dataset to our room environment. While there is a more consistent improvement over the classes when testing on the synthetic dataset, this difference is still not very large. Therefore we cannot conclusively determine whether customising a dataset for a specific environment does or does not add to performance. To draw a conclusion, further research is needed, particularly the generation of more varied datasets with additional classes, and real-world data for testing.

The ablation study in Experiment 3 shows that parameter randomisation has the greatest impact on the `new` classes. For example, fixing the pillar width parameter lowers the performance of the `pillars new` class. We theorise this is because the `chair new` contains multiple small features, making it susceptible to High-Frequency noise disturbances. Additionally, combining pillar width, chair size and LiDAR block size impacts performance even more. This makes sense, as Mask R-CNN is a size invariant model [10] which means that the absolute size of objects is not important but the relative size is.



(a) Tested on real-world data.



(b) Tested on synthetic data.

Figure 16: mAP bounding box results of the ablation experiments on the Real-world and Synthetic validation datasets. Each color represents a model trained on the same parameters for dataset creation, with a fixed value (Table VIII), except for Pillar Width, Chair Size, and Block Size, which are combined in one parameter.

While comparing the results for the ablation study tested on real-world data the same parameters seem impacted as when testing on the synthetic data except for the `doors` class. We attribute this to the domain gap between real-world and synthetic data, but can not explain exactly why different parameters impact this class more.

Altogether this shows that in the 2D data generation domain, randomisation is useful for robust model performance and fixed parameters in some values can be compensated by randomising other values. In future research, more parameter combinations could be tested to see which parameters work synergistically and which parameters do not need variation for performance. We also show that our novel 'low frequency' noise method does not seem to add to performance significantly.

While this work shows promise, a significant limitation is the small number of classes in the dataset generation method, which may not adequately represent a real-world environment. We believe it is likely that model performance will degrade when adding more classes, especially when these classes are similar in appearance. Our approach could also be extended to include updating wall geometry, potentially benefiting the maintenance of BIM models. Furthermore, exploring more lightweight instance segmentation networks or Generative Adversarial Networks (GANs) might offer promising alternatives to the relatively older Mask R-CNN model, especially the lightweight models that may be useful for running locally on robots.

Another use case for the synthetic data generation method is fine-tuning, where a model initially trained on a large dataset of synthetic data is subsequently fine-tuned with a smaller amount of real-world data. This approach allows the model to learn general patterns from the synthetic data before adapting to the specifics of real-world scenarios.

V. Conclusion

In this work, we propose a novel parametric data generation method to train an unmodified Mask R-CNN model for updating digital twins using 2D LiDAR perceptual data. We complement the sparse data from 2D LiDAR sensors by creating a prior map from data retrieved from alternative sources. The research demonstrates that synthetic data can be used to train a model that detects mismatches between a simulated digital twin and real-world detections. We also show that a key advantage of this method is the unmodified Mask R-CNN model's ability to complete the shapes of objects, even when the LiDAR detection captures only a partial view, such as detecting just one side of an object. Additionally, we use uncertainty area detection to determine areas where the Mask R-CNN model outputs multiple confident predictions. Further research is necessary to conclusively determine the effectiveness of dataset customisation for specific environments. Finally, we provide evidence that multiple parameters, when varied together in data generation, can compensate for the negative impact of not randomising individual parameters.

VI. Acknowledgements

I would like to thank my supervisors Elena Torta and Ghijs van Rhijn for their valuable feedback and support during this thesis. Additionally, I'm grateful to my fellow student on the fifth floor of Flux for their support, assistance and many laughs.

REFERENCES

[1] R. de Koning, E. Torta, P. Pauwels, R. Hendrikx and M. J. G. van de Molengraft, 'Queries on semantic building digital twins for robot navigation,' in *9th Linked Data in Architecture and Construction Workshop*, vol. 3081, 2021, pp. 32–42.

[2] S. K. Punia, M. Kumar, T. Stephan, G. G. Deverajan and R. Patan, 'Performance analysis of machine learning algorithms for big data classification: Ml and ai-based algorithms for big data analysis,' *International Journal of E-Health and Medical Communications (IJEHMC)*, vol. 12, no. 4, pp. 60–75, 2021.

[3] H. Durrant-Whyte, D. Rye and E. Nebot, 'Localization of autonomous guided vehicles,' in *Robotics Research: The Seventh International Symposium*, Springer, 1996, pp. 613–625.

[4] A. Adkins, T. Chen and J. Biswas, 'Obvi-slam: Long-term object-visual slam,' *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2909–2916, 2024. DOI: 10.1109/LRA.2024.3363534.

[5] J. Qian, V. Chatrath, J. Yang, J. Servos, A. P. Schoellig and S. L. Waslander, 'Pocd: Probabilistic object-level change detection and volumetric mapping in semi-static scenes,' *arXiv preprint arXiv:2205.01202*, 2022.

[6] J. Qian, V. Chatrath, J. Servos *et al.*, 'Povslam: Probabilistic object-aware variational slam in semi-static environments,' *arXiv preprint arXiv:2307.00488*, 2023.

[7] M. Hussain, 'Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection,' *Machines*, vol. 11, no. 7, p. 677, 2023.

[8] W. Gu, S. Bai and L. Kong, 'A review on 2d instance segmentation based on deep neural networks,' *Image and Vision Computing*, vol. 120, p. 104 401, 2022, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2022.104401. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885622000300.

[9] M. Gualtieri and R. Platt, 'Robotic pick-and-place with uncertain object instance segmentation and shape completion,' *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1753–1760, 2021. DOI: 10.1109/LRA.2021.3060669.

[10] K. He, G. Gkioxari, P. Dollár and R. Girshick, 'Mask r-cnn,' in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[11] S. Borkman, A. Crespi, S. Dhakad *et al.*, 'Unity perception: Generate synthetic data for computer vision,' *arXiv preprint arXiv:2107.04259*, 2021.

[12] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, 'Domain randomization for transferring deep neural networks from simulation to the real world,' in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 23–30.

[13] S. Hinterstoisser, O. Pauly, H. Heibel, M. Martina and M. Bokeloh, 'An annotation saved is an annotation earned: Using fully synthetic training for object detection,' in *Proceedings of the IEEE/CVF international conference on computer vision workshops*, 2019, pp. 0–0.

[14] G. O. Flores-Aquino, J. Duvier Díaz Ortega, R. Y. Almazan Arvizu, O. Octavio Gutierrez-Frias, R. L. Muñoz and J. Irving Vasquez-Gomez, '2d grid map generation for deep-learning-based navigation approaches,' in *2021 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, 2021, pp. 66–70. DOI: 10.1109/ICMEAE55138.2021.00018.

[15] D. Feng, Z. He, J. Hou, S. Schwertfeger and L. Zhang, 'Floorplannet: Learning topometric floorplan matching for robot localization,' in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 6168–6174. DOI: 10.1109/ICRA48891.2023.10160977.

[16] B. Kaleci, K. Turgut and H. Dutagaci, '2dlasernet: A deep learning architecture on 2d laser scans for semantic classification of mobile robot locations,' *Engineering Science and Technology, an International Journal*, vol. 28, p. 101 027, 2022, ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2021.06.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098621001397.

[17] Blender Foundation, *Eevee rendering — blender manual*, Accessed: 2024-08-22, 2023.

[18] DIYer22, *Bpycv*, https://github.com/DIYer22/bpycv, Accessed: 2024-06-03, 2024.

[19] A. Lagae, S. Lefebvre, R. Cook *et al.*, 'A survey of procedural noise functions,' in *Computer Graphics Forum*, Wiley Online Library, vol. 29, 2010, pp. 2579–2600.

[20] S. Macenski, T. Foote, B. Gerkey, C. Lalancette and W. Woodall, 'Robot operating system 2: Design, architecture, and uses in the wild,' *Science Robotics*, vol. 7, no. 66, eabm6074, 2022.

DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[21] P. Skalski, *Make Sense*, https://github.com/SkalskiP/make-sense/, 2019.

[22] PyTorch, *Mask r-cnn*, https://pytorch.org/vision/main/models/mask_rcnn.html, Accessed: 2024-05-31.

[23] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106.

[24] P. van den Akker, *A parametric dataset generator for updating digital twins through 2d lidar perceptual data*, 2024. [Online]. Available: https://gitlab.tue.nl/et_projects/graduationprojects/pa-dtupdate.

[25] B. Carterette and E. M. Voorhees, 'Overview of information retrieval evaluation,' in *Current Challenges in Patent Information Retrieval*, M. Lupu, K. Mayer, J. Tait and A. J. Trippe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 69–85, ISBN: 978-3-642-19231-9. DOI: 10.1007/978-3-642-19231-9_3. [Online]. Available: https://doi.org/10.1007/978-3-642-19231-9_3.

[26] A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollar, 'Panoptic segmentation,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[27] T. Bektas, 'The multiple traveling salesman problem: An overview of formulations and solution procedures,' *Omega*, vol. 34, no. 3, pp. 209–219, 2006, ISSN: 0305-0483. DOI: https://doi.org/10.1016/j.omega.2004.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048304001550.

[28] R. Bridson, 'Fast poisson disk sampling in arbitrary dimensions.,' *SIGGRAPH sketches*, vol. 10, no. 1, p. 1, 2007.

[29] B. Foundation, *Noise texture node*, https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/noise.html, n.d. (visited on 29/05/2024).

APPENDIX

A. Wall creation method

For the creation of the 3D geometry, first, the walls are created. The parameters that determine the geometry shape for the walls are summarized in Table III. The room layout is determined by a 2D grid, with vertices (points) v_i depicted as yellow in Figure 17. This grid forms the general layout of the room in XYZ-Cartesian space. A grid has four variables that determine its shape: Size X, Size Y, Vertices X, and Vertices Y. Size X and Size Y determine the Total size X (W_x^s) and Total size Y (W_x^s) of the room. Vertex X and Y are calculated by Vertex X $= W_x^{nr} + 2$ and Vertex Y $= W_y^{nr} + 2$. In this way W_x^{nr} and W_y^{nr} represent the number of wall segments inside the room excluding the outer wall.

The vertices ($v_i(x, y)$) are connected, this connection is called an edge $e(v_i, v_j)$ where v_i and v_j are arbitrary vertices. This grid is split into two parts as depicted in Figure 17: the outer wall (red) which remain unchanged and the inner wall (green) which will modified further.

To form unique random layouts, inner wall sections are deleted, creating hallways and rooms (Figure 17b). Each edge, $e(v_i, v_j)$ that is not part of the outer wall is assigned a uniform random number U in the range $(-1, 1)$. Subsequently an edge is deleted when

$$U \leq 1 - W^d. \tag{5}$$

Here $(1 - W^d)$ is the deletion threshold. Using this method the variable Wall density (W^d) controls the total average percentage of wall segments out of the maximum. For example, if $W^d = 0.5$ on average half the walls inner wall segments are deleted.

The previous step results in a semi-random room layout with perfectly perpendicular walls. However, since walls are not always perpendicular in reality, therefore, additional randomness is introduced.

Assume that:

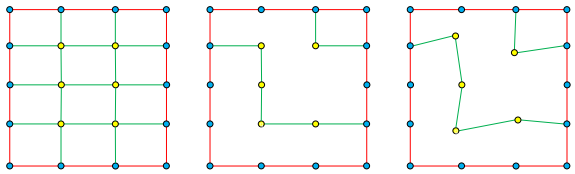
- V: Set of all the vertices
- $v_{in} \in V$ Set of all inner wall vertices, depicted as yellow dots in Figure 17

For v_{in} random noise is depicted in Figure 17c with the rule:

$$v_{in} = v_{in} + v_{random} \cdot W^R, \tag{6}$$

where v_{random} is a uniformly distributed random vector bounded by $(-1, 1)$ and W^R the max wall randomness multiplier parameter. With this method, the randomness can be determined by setting W^R to a value. For example, setting W^R to 0 will keep all the walls in a grid (Figure 17b), while for example setting W^R to 1 will add a maximum offset of $\pm 1m$ (Figure 17c).

Now that a 2D room outline has been created, this can be transformed into 3D geometry. This process involves starting with a profile, a 2D shape of the wall, and "sweeping" this shape along each edge of the room outline. The 2D shape of the wall defines its width (W^w) and height (W^h). Once this step is completed, a 3D model of the room, consisting only of the walls, is created. In (Figure 18) the 3D profile derived from Figure 17c can be observed.



(a) Original grid (b) Remove edges (c) Add noise

Figure 17: Grid layout steps for random room creation. Yellow and blue dots are the inner and outer wall vertices respectively. Red lines indicate the outer wall edges and green lines the inner wall edges.

Table III: Parameters and their interpretation for room layout generation.

Input	**Value interpretation**	**Range**	**Abbreviation**
Total size X	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	W_x^s
Total size Y	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	W_y^s
Wall Width	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	W^w
Wall Height	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	W^h
Wall Nr X	Nr of X wall sections	$\{n \in \mathbb{N} \mid x \geq 0\}$	W_x^{nr}
Wall Nr Y	Nr of Y wall sections	$\{n \in \mathbb{N} \mid x \geq 0\}$	W_y^{nr}
Wall Density	Percentage of filled walls	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	W^D
Seed	Random number	$\{n \in \mathbb{Z} \mid n \geq 0\}$	S
Max wall randomness	Randomness multiplier	$\{x \in \mathbb{R} \mid x \geq 0\}$	W^R

Table IV: The parameters that govern the door placement and geometry.

Input	Value interpretation	Range	Abbreviation
Min door width	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	D^w_{min}
Max door width	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	D^w_{max}
Max door rotation	Angle (radian)	$\{x \in \mathbb{R} \mid x \geq 0\}$	D^m_{rot}
Door density	Probability of a wall section having a door	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	D_{den}
Door height	Meter	$\{x \in \mathbb{R} \mid x > 0\}$	D_{height}
Max door location offset	Percentage of wall length where door can be located	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	D_{offset}
Door closed probability	Probability of each door to be closed	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	D_{closed}

B. Door Creation

After creating a room, doors can be placed in the inner wall segments. The doors are fully parametric, including adjustable width, height, placement, and rotation. The parameters for the door creation can be found in Table IV.

Doors are placed along wall segments (green lines in Figure 1). To determine potential door locations, each wall segment is divided into points centred along its edges. The max door location offset (D_{offset}) parameter controls the door centre location along the wall segments. This parameter decides for each wall segment what percentage of the wall length a door is allowed to be placed. The door location is uniformly distributed along the wall segment bounded by:

$$\frac{W_{len}}{2} - D_{offset} \leq D_{cen} \leq \frac{W_{len}}{2} + D_{offset} \quad (7)$$

Where W_{len} is the length of the wall segment, D_{cen} is the distance from the centre of the door to the wall.

For example, if a wall segment is $2m$ long and $D_{offset} = 0.5$ the door centre has a random chance to be located anywhere between $0.5m \leq$ door centre $\leq 1.5m$ of the wall segment.

Each of these centre points is assigned a uniform distributed value U in the range $(-1, 1)$ a door is placed when:

$$U \leq 1 - D_{den} \quad (8)$$

Where D_{den} is the user-defined parameter that governs the probability of a wall section containing a door.

After the door locations are determined the doors can be placed. The door width (D^w) is uniformly distributed and bounded by:

$$D^w_{min} \leq D^w \leq D^w_{max} \quad (9)$$

D^w_{min} and D^w_{max} are the minimum and maximum door width parameters defined by the user.

The door height is calculated using the user-defined D_{height} parameter. The door height becomes:

$$\text{Total door height} = W^h \cdot D_{height} \quad (10)$$

Where W^h is the wall height.

Doors can be rotated to open or close. The rotation axis of a door is $\frac{\text{door width}}{2}$. Each door is assigned a uniform distributed value P in the range $(-1, 1)$ a door is closed when:

$$P \leq 1 - D_{closed} \quad (11)$$

If a door is open the angle is randomised in a uniformly distributed range of:

$$-D^m_{rot} \leq D_{angle} \leq D^m_{rot} \quad (12)$$

D^m_{rot} is the user-defined parameter to determine the maximum rotation and D_{angle} is the door angle relative to the wall, both in radians.

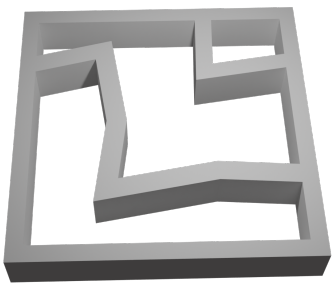


Figure 18: A 3D room generated from the 2D layout shown in Figure 17.

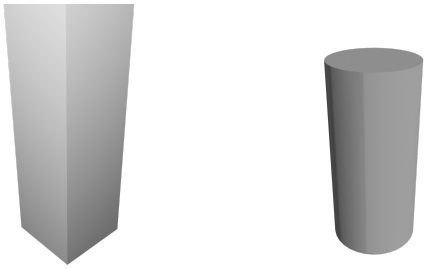
C. Objects

Now that the room layout has been fully defined including doors, objects are needed to populate it. In this thesis three objects are used: chairs, pillars and tables. For some examples of chairs see: Figure 20 and Figure 19 for pillar examples. The parameters for these objects can be found in Table V and Table VI. For implementation details please see [24].

D. Object distribution

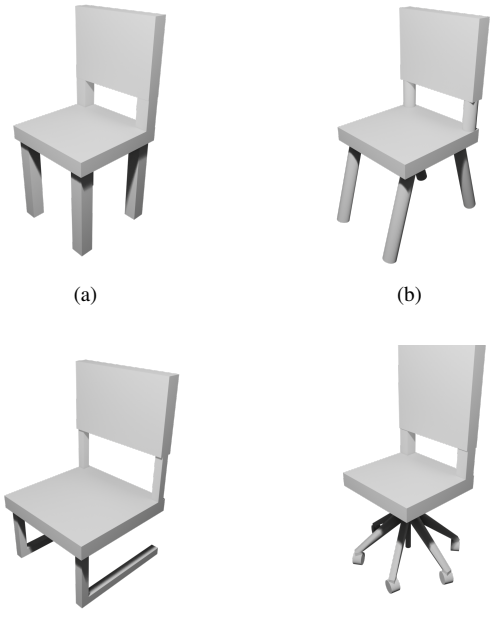
After the creation of the walls and objects these objects need to be distributed in the room without intersecting with objects. Generally, this problem is solved with a travelling salesman algorithm [27]. However, implementing this in Blender presents challenges. We won't go into the specifics of these challenges here, but the primary reason comes from the use of instancing in Blender, which prevents us from knowing the precise shapes of objects during placement.

When placing objects we can only know the bounding box shape and the centre location of the object. Knowing this we designed an algorithm to place objects in the scene with minimal intersecting.



(a) (b)

Figure 19: Comparison of two pillars generated with our data generation method.



(a) (b)

(c) (d)

Figure 20: Examples of different types of chairs generated with our data generation method.

Table V: Parameters for the Chair Model

Input	**Value Interpretation**	**Range**
Chair Width	Width in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Chair Seat Height	Seat height in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Chair Back Height	Back height in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Chair Leg Width	Leg width in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Circular Legs	Circular or square legs	{True, False}
Leg type	Choose from types: vertical, angles, office chair, double angled	$\{n \in \mathbb{N} \mid x \leq 4\}$
Leg center offset percentage	Relative location of the chair leg relative to the outside.	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$

Table VI: Parameters for Pillar

Input	**Value Interpretation**	**Range**
Pillar Height	Height in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Pillar Width	Width in meters	$\{x \in \mathbb{R} \mid x > 0\}$
Square	Square or circular pillar	{True, False}

Assume B_i^{max} is the maximum x or y length of the bounding box of any object $z \in Z$. And U_z^{max} is the minimum distance set between objects i as set by the user. The basis of this algorithm is the Poisson Disk distribution [28]. This distribution method creates a semi-random distribution of points, a useful property of this method is that we can specify a minimum distance (r) between samples. We set this minimum distance as

$$d_{min} = max(B_z^{max}, U_z^{max}) \tag{13}$$

making sure that any object is unlikely to intersect with another object of the same type, and also giving the user control over how the objects are scattered. This will result in occasional overlap because we do not take angles into account. For our purposes minor errors are acceptable.

Assume we make this Poisson Disk Distribution for every object i. We then iterate through each point ($p_i \in P$) and calculate whether there exists a point within the specified minimum range d_{min}

$$\begin{aligned} &\text{For each } p_i \in P, \quad \text{if there exists } p_j \in P \\ &\quad \text{with } p_i \neq p_j, \quad \text{such that } d(p_i, p_j) < d_{\text{min}}, \\ &\qquad \text{then } P := P \setminus \{p_i\}. \end{aligned} \tag{14}$$

With $d(p_i, p_j$ is the distance between p_i and p_j. This results in a distribution of points with a minimum distance that does not exceed the bounding box distributions or the user-defined minimum range.

Table VII: Parameters for the LiDAR

Input	Value interpretation	Range	Abbreviation
Max offset high freq noise	Maximum offset per point	$\{x \in \mathbb{R} \mid x \geq 0\}$	L_{hf}
Max offset Low Freq noise	Maximum offset per point	$\{x \in \mathbb{R} \mid x \geq 0\}$	L_{lf}
LiDAR block size	Size of points shown on the image visualization	$\{x \in \mathbb{R} \mid x \geq 0\}$	L_b
LiDAR height	Height in meter	$\{x \in \mathbb{R}\}$	L_h

E. LiDAR generation

The LiDAR is simulated as an object in 3D space that generates LiDAR points. LiDAR points are generated where rays from the simulated LiDAR sensor intersect with environmental objects. The placement of the LiDAR is defined as pos $= (x, y, z)$ in the room is given by:

$$\begin{aligned} x &\sim \mathcal{N}\left(0, \left(\frac{W_x^s}{6}\right)^2\right) \\ y &\sim \mathcal{N}\left(0, \left(\frac{W_y^s}{6}\right)^2\right) \\ z &= L_h \end{aligned} \tag{15}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a normal distribution with mean μ and variance σ^2, W_x^s and W_y^s are the the x and y sizes of the room and L_h as the user defined LiDAR height.

As mentioned in Appendix II-B4, we introduce two types of noise: High Frequency and Low Frequency that are applied to LiDAR points. The high-frequency noise is relatively straightforward. For each LiDAR point, we generate a new individual two-dimensional random vector $U \in \mathbb{R}^2$ bounded by $(-1, 1)$. The new position for each LiDAR point $P(x, y)_{new} \in \mathbb{R}^2$, constrained in the XY plane, is determined as follows:

$$P(x, y)_{\text{high freq noise}} = U(x, y) \cdot L_{hf} \tag{16}$$

Here, $P(x, y)_{new}$ is the new position, $P(x, y)_{orig}$ is the original position and L_{hf} is a user-defined variable that specifies the maximum allowable offset. This method provides a convenient and intuitive way for the user to control the maximum offset.

After determining the High Frequency noise the Low Frequency noise is added. To do this we use Perlin noise (Figure 21), for specifics on Perlin noise see [19]. In short Perlin noise creates textures that are smooth continuous patterns. We use Blenders Perlin noise texture function [29] for this purpose. In Blender Perlin noise has many inputs, these values were selected through visual inspection of the outputs: Scale: 2.08, Detail: 1.3, Roughness: 0, Distortion:0. Additionally we use the 4D variety of the noise texture, this allows us to set the W value which is the texture coordinates to evaluate the noise at. We randomise the output by setting this equal to the random seed S. For every LiDAR point, the Perlin noise outputs a value $p(x, y)$ between $0-1$. We map this value from $[0, 1]$ to $[-1, 1]$ by:

$$P(x, y)_{\text{Low Freq noise}} = -L_{lf} + 2P(x, y)_{\text{perlin}} L_{lf}, \tag{17}$$

for both the x and y coordinates. Where $P(x, y)_{\text{Low Freq noise}}$ is the output mapped Perlin noise L_{lf} is the maximum distance by which a LiDAR point may be offset in the x and y direction and P_{perlin} is the Perlin noise value (the x and y coordinate value). Resulting in a Perlin noise for each point of $P(x, y)$ with a maximum x or y offset equal to the user-defined L_{lf}.

Finally, all the noises are added together:

$$\begin{aligned} P(x, y)_{\text{final}} = P(x, y)_{\text{orig}} + P(x, y)_{\text{Low Freq noise}} \\ + P(x, y)_{\text{high freq noise}}, \end{aligned} \tag{18}$$

Figure 21: Perlin noise.

Table VIII: Parameters for ablation experiment and their associated fixed values.

Parameter(s)	Value(s)
LiDAR Freq High	0
LiDAR Low Freq	0
LiDAR Block Size	0.01
Pillar Width	1.5
Chair Size	1.5
Pillar Width, Chair Size, LiDAR Block Size	(1, 1, 0.15)

where $P(x,y)_{\text{final}}$ is the output position of a LiDAR point, $P(x,y)_{\text{orig}}$ is the true original position of the LiDAR point and $P(x,y)_{\text{Low Freq noise}}$ and $P(x,y)_{\text{high freq noise}}$ are the two noise types respectively.

Using this method we have a highly customised method of simulating LiDAR noise in a 3D world, including both high-frequency noise and a low-frequency smooth displacement.

Note, that it is possible for the LiDAR to be placed in another object. This will result in an empty image. This is not a large problem because the error will be compensated by the number of images generated.

Table IX: Parameters used in the generation of the datasets. Where applicable all dimensions are in meters. Any value with (x,y) indicates a uniformly distributed range between these values.

Parameter	**Model I** (Customized)	**Model II** (Generalized)
Image resolution	[280, 280]	[280, 280]
LiDAR height range	(0.2, 0.21)	(0.2, 1.5)
Minimum overlap percentage for visible objects	0.001	0.001
Probability that object is class `new`	0.333	0.333
Probability that object is class `remove`	0.333	0.333
Wall width Range	(0.1, 0.3)	(0.1, 0.3)
Nr of wall sections in X direction	(0, 2)	(0, 2)
Nr of wall sections in Y direction	(0, 2)	(0, 2)
Wall density	(0.7, 1)	(0.7, 1)
Seed	(0, 10000)	(0, 10000)
Min door width	0.7	0.7
Max door width	1.3	1.3
Max wall randomness range	(0, 1.6)	(0, 1.6)
Max door rotation	$(\pi/2, \pi)$	$(\pi/2, \pi)$
Door density	(0.5, 1)	(0.5, 1)
Room height	3	3
Pillar minimum space to closest object p_{min}	3	3
Chair minimum space to closest object p_{min}	1	1
Chair size (width) range	(1.2, 2.5)	(1, 3)
Leg height range	(1.2, 2.5)	(1, 3)
Distance from seat to bottom backrest range	(0.3, 0.8)	(0.3, 0.8)
Back rest height range	(0.3, 0.8)	(0.3, 0.8)
Leg width	(0.05, 0.1)	(0.01, 0.2)
Circular legs	(True, False)	(True, False)
Leg type	(0, 1)	(0, 3)
Leg center offset percentage	(0.5, 1)	(0.5, 1)
Chair Width	(1, 2)	(0.5, 4)
Chair Round/square	(True, False)	(True, False)
Leg radius	(0.05, 0.1)	(0.05, 0.12)
High freq noise variance	(0, 0.1)	(0, 0.1)
Low Freq noise variance	(0.0, 1.0)	(0, 1)
Lidar block size	(0.1, 0.2)	(0.05, 0.2)
Nr lidar points	512	512
Walls Display Color	(255, 255, 255, 255)	(255, 255, 255, 255)
Chair Display Color	(0, 255, 0, 255)	(0, 255, 0, 255)
Pillars Display Color	(255, 255, 0, 255)	(255, 255, 0, 255)
Doors Display Color	(255, 0, 255, 255)	(255, 0, 255, 255)
LiDAR Display Color	(255, 0, 0, 0)	(255, 0, 0, 0)

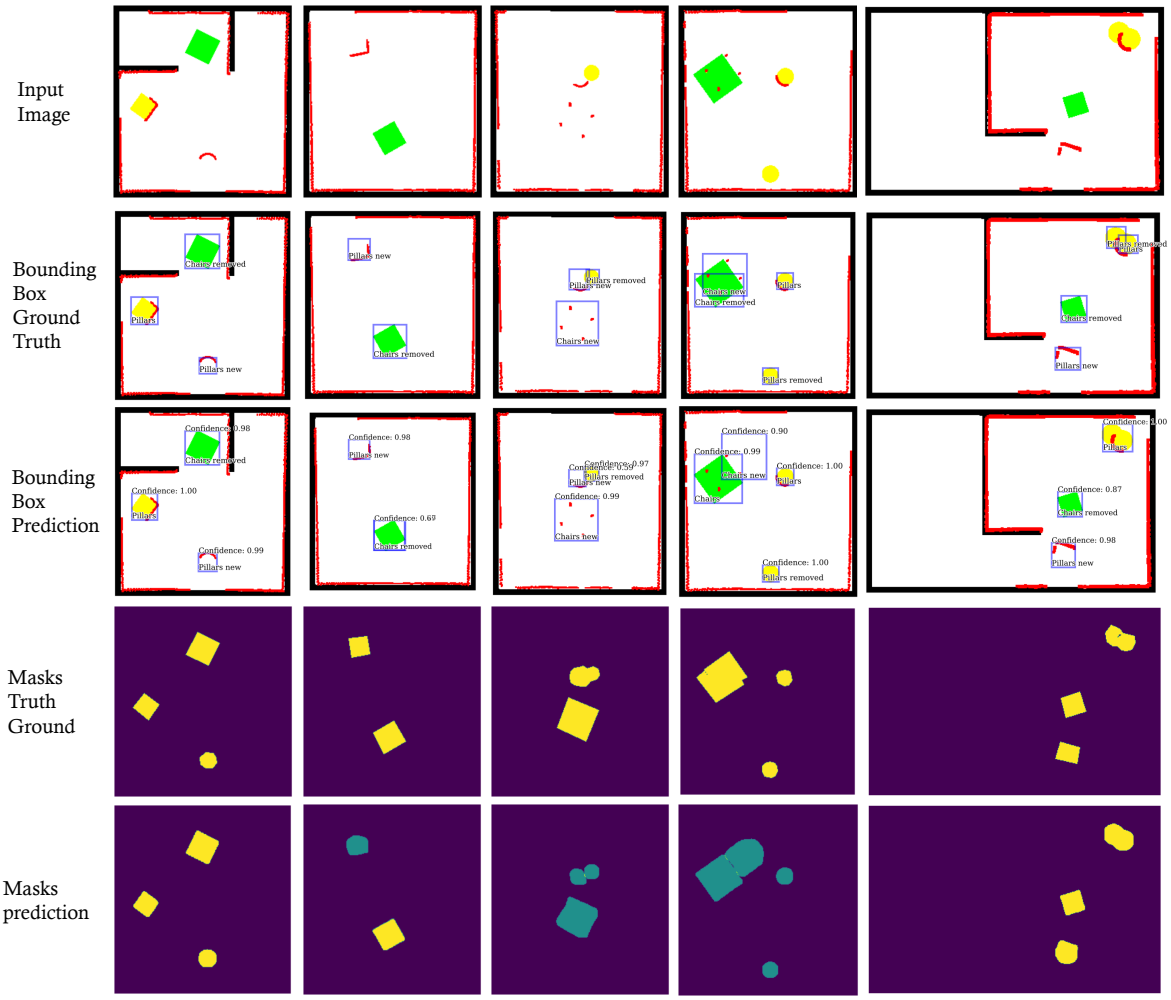


Figure 22: Comparison of Object Detection and Segmentation Results: The top row shows the Input Images, followed by the Ground Truth and predicted bounding boxes for the objects. The bottom two rows display the Ground Truth and predicted masks, highlighting the model's performance in detecting and segmenting various shapes within the images.