



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF ARTIFICIAL
INTELLIGENCE

Semantic Matching in Human-Robot Collaboration

SCIENTIFIC STUDENT CONFERENCE

Authors:

Szilvia Szeier

Computer Science MSc

Benjamin Baffy

Computer Science MSc

Supervisors:

András Lőrincz

Senior Researcher
Department of Artificial
Intelligence

Budapest, 2022

Abstract

Human-robot interaction, collaboration, and joint planning have specific limitations, due to the constraints of human communications. The solution to the general problem of communication is distant, but there are tasks, such as navigation that are within reach. A necessary component is the semantic map, where occupied voxels of the environment have labels, such as “table”, “chair”, “staircase”, etc. Previously, we have developed (with colleagues) a real-time semantic mapping system. It was evaluated on small robots. A general issue appeared: the labels differed for the robot, since the deep learning labelling algorithm could not provide outputs due to the very different viewpoint. The same problem should occur for drones, too. This calls for semantic matching between the human and the robot, where human-defined labels should be transferred to the robot’s perspective. In addition to correct labelling, understanding the geometry of any environment is essential for a deeper comprehension of the human’s environment. We present a system that can perform such semantic matching efficiently, using segmentation from any viewpoint utilizing superpixels and the 3D reconstruction obtained from a human perspective.

1 Introduction

Intelligent agents have been assisting us in our daily lives for the past several years. They help us out within a diverse set of tasks ranging from autonomous driving to guidance systems in healthcare. These systems perform specified tasks quickly and precisely, without much ability to generalize. To be able to adapt tasks, a certain level of understanding needs to be established.

As of today, many systems lack the comprehension of the context surrounding different scenarios. An essential piece of information when dealing with rooms or similar indoor environments is its semantics. The semantics describe our surroundings, we can deduct what kind of room the given area is, what purpose it can serve, and most importantly, what can be done in it. In this study, we focus on indoor environments and their semantics to achieve a higher level of understanding of our surroundings.

The task of semantic segmentation has been around for quite a few years in the field of artificial intelligence. There are countless systems and algorithms which semantically segment two dimensional RGB images [6, 8] and even multiple datasets containing 3D models of various objects or environments with given annotations. Most of the semantic segmentation algorithms that can be found are trained on images taken from the perspective in which we perceive the world, we refer to this viewpoint as the human perspective. Since these methods lack training examples from unusual viewpoints such as those captured by drones or small ground agents, their predictions are unreliable. With 3D datasets, this problem does not arise, but only a limited number of such datasets exist, since creating them is a tedious task and therefore there is not much variety.

In this study we propose a method of obtaining semantic segmentation from any viewpoint starting from a partial human viewpoint reconstruction of the environment. This work is a continuation of the efforts made by [26], where the authors provided a pipeline for real time RGB or semantic 3D reconstruction of indoor environments. In their work, the authors focused on developing a modular architecture. This allows components to be easily replaced with new state-of-the-art alternatives as they become available. With our contributions we maintain the modular approach.

An alternate method to semantically segment images from unusual viewpoints is to train a network on annotated examples. This requires examples of such data, the collection of RGB images is the simpler task at hand, the real challenge is acquiring ground truth annotations. One option would be to hand annotate them, but this approach is costly and time-consuming. Another possibility would be to utilize 3D RGB/semantic datasets by rendering 2D images from them as training data. This procedure would cut down on the time and cost constraints, but due to the lack of such datasets these networks would have very limited variation.

As a consequence, we aimed to build on the strong capabilities of human perspective semantic segmentation algorithms, while trying to extend their range to unusual viewpoints. Our work is based on [26] as we extend its potentiality to create an accurate semantic reconstruction even when run on various robotic agents. Initially we developed our method within a simulator using a dataset with semantic annotations provided. Due to the available ground truth data, we are able to quantitatively evaluate our results. The original pipeline proposed by [26] works well in real-world environments and our method is no different. As opposed to the simulated environments, we do not have ground truth

semantic annotations for these rooms, therefore the evaluation is done qualitatively. In both the simulated and real-world scenarios we maintain real time capabilities.

2 Related Works

In this section we review contributions related to our research topic. Our method provides a modular approach, which allows the replacement of pipeline components as new state-of-the-art becomes available.

2.1 Pose Tracking

The task of obtaining the position and orientation of sensors or any defined entity is called pose tracking. These poses can be described with 6 degrees of freedom (DoF) and are usually represented with a 4×4 homogeneous transformation matrix. The technology best suited for this task varies by use case. In outdoor environments and changing weather conditions, LiDAR-based sensor fusion methods are the optimal choice, while for indoor environments SLAM approaches are a more common choice.

In robotics, simultaneous localization and mapping (SLAM) algorithms are used to construct maps of unknown environments and simultaneously track the agent’s pose with respect to the surroundings. There are different types of SLAM applications, but in this study we focus on visual SLAM technology, which utilizes 3D computer vision to recognize and track points throughout a sequence of frames.

The ORB-SLAM methods [19, 18, 3] are the most popular visual SLAM approaches, since they are able to tackle this issue while maintaining real-time performance. The first version of the algorithm focused on monocular cameras, but as the newer variants appeared, the developers solved the scale ambiguity problem with the use of stereo or RGBD sensors and later added visual-inertia support.

2.2 3D Reconstruction

The goal of 3D reconstruction is to recreate the shape and appearance of real world objects and/or scenes with low cost and memory requirements. In the scope of this study, by 3D reconstruction we are referring to the 3D modelling of indoor environments. This can be achieved by traversing the area with an RGBD or stereo camera while tracking its location and orientation. The recreation of a scene can be done in an online or offline manner.

Offline reconstruction methods are used to model large 3D environments. These methods often use structure from motion (SfM), where the algorithm is given a large set of RGB images and the goal is to reconstruct the space by finding correspondences within the images by projecting them into a 3D world. In this case, the camera parameters or the geometry of the reconstruction are not known, so it becomes the algorithms goal to estimate them. One of the most known contributions in this field is *Building Rome in a Day* [2].

In real-time 3D reconstruction algorithms the camera pose has to be tracked. The camera pose, colored image, and depth image are all used independently to update a global

reconstruction frame-by-frame. In most robotic applications real-time 3D reconstruction is necessary, for example a navigation robot needs to be aware of immediate changes in its environment to be able to generate a collision-free trajectory. The geometric representation of the environment can greatly affect the precision of the reconstruction, the memory needed to store it, and even the computation efficiency. Maps can be represented with a 3D voxel grid, Signed Distance Function (SDF), or Truncated Signed Distance Field (TSDF), which is a more memory efficient version of SDFs, among many others. Voxel-hashing is another efficient method for free-space storing; this representation can be seen being used in Voxblox [21]. The Kimera [24] framework is built on Voxblox to maintain good performance.

2.3 Semantic Segmentation

Semantic segmentation refers to the pixel-wise clustering of image regions that belong to the same object. Once these regions have been identified each one is classified according to a category. The first advances in semantic segmentation were made on 2D images. Large scale annotated datasets like ImageNet [11] or CoCo [13] already exist.

To semantically segment 3D geometries there are 2 commonly used approaches. The first approach uses complete 3D model as input. If we consider the final goal being to segment a 3D reconstruction, it would make sense to give that as input since we could utilize the 3D geometry of objects. This would certainly be beneficial, but the problem can be found within the lack of available datasets due to the complexity and time needed to annotate them. Some, like [7], do exist, but most of these models only contain a few object categories.

The second approach includes projecting the labels obtained from a 2D image onto the 3D model. The Kimera [24] framework that we use within our pipeline also creates the 3D reconstruction in this manner. 2D semantic segmentation datasets are much more common and are significantly easier to create, therefore this is why we utilize a 2D to 3D segmentation approach.

3 Tools

During the scope of this study we initially developed and tested in a simulated environment then recorded data and qualitatively evaluated the results in real-world environments. In this section we describe the tools used for the two types of experiments.

3.1 Simulation

Before experimenting in real world scenarios, where experiments are more tedious to conduct and often times we run into the issue of not having ground truth measurements to quantitatively evaluate results, we make use of simulated data. Working with simulated data has a lot of advantages, but it also introduces certain problems. Since we initially intended to use a 3D reconstruction dataset rather than testing in the real world, we risk introducing a challenge called the *sim-to-real gap*, which originates from the limitations of reflecting the real world within the datasets.

3.1.1 Replica Dataset

There are multiple high quality 3D reconstructions of indoor spaces like Matterport3D [4], Gibson [33], and the Replica [32] dataset. Each of these datasets have their own advantages and disadvantages, a comparison between them can be seen in Table 1.

		Replica	Matterport3D	Gibson
Scale	Nr. Scenes	18	90	572
	Nr. Rooms	35	2056	-
Quality	Color res. $[\frac{pixel}{m^2}]$	92k	97k	97k
	Geometry res. $[\frac{points}{m^2}]$	6k	0.7k	0.7k
Fidelity	HDR Textures	✓	×	×
	Reflections	✓	×	×
	Semantic Classes	88	40	×

Table 1: Comparison of the 3D reconstruction datasets, by different metrics. [25]

The Replica dataset consists of 18 highly photo-realistic 3D indoor scenes at both room and building level, a few of which can be seen in Figure 1. The scenes are made up of dense meshes, high-resolution texture, semantic class and instance level information, and mirror/glass reflections as well. The dataset was created to enable machine learning research for a variety of tasks ranging from segmentation in 2D or 3D to the training and development of embodied agents. The dataset was created with the hope that algorithms trained on it would successfully transfer to real world applications as well. A custom built RBGD capture rig was used to capture the raw data to build the dataset that utilized a SLAM system for localization. The poses obtained from the SLAM algorithm together with the depth images were fused into a TSDF. From this, meshes were extracted using the standard Marching Cubes [14] algorithm. Semantic annotation was performed by first rendering 2D images and applying a 2D instance-level masking tool, then fusing the annotations back onto the mesh and using a voting scheme. Finally, the 3D annotation are refined with the help of a superpixel-like segmentation algorithm.

Keeping in mind that we eventually want to apply our algorithm to real-world environments, we try to overcome the difficulties stemming from the *sim-to-real gap*. The environments will not have ideal lighting conditions and the reconstructions will not be unrealistically clean and noise-free. When choosing a dataset to work with we kept this challenge in mind and therefore an important factor was using a photo-realistic dataset. Additionally, in our pipeline we utilize RGB projections, depth measurements, and ground truth semantic labelling of the environment. The Replica dataset was an obvious choice as it met all the necessary criteria. Additionally, it is supported by the employed simulation



Figure 1: A few examples of the models in Replica [32]. From left to right: apartment 1, room 2, and office 3.

environment by default (discussed in the next section), so no problematic adaptation was needed for us to be able to use it.

3.1.2 Habitat Simulation Environment

As a first step of developing most pipelines it makes sense to test in simulated frameworks to reduce the time needed to conduct real world experiments. Ideally, we would train agents in the real world, but many challenges arise. Besides the testing being slow, it is resource-sensitive, difficult to control and reproduce test cases, and most importantly there is a lack of ground truth measurements. With a modular pipeline it is possible to test the components separately, but then there is a significant chance of complications arising, therefore we aim to test the entire system. Taking this into consideration, we decided to run experiments and evaluate our methods quantitatively using a simulator. The iGibson [28] and Habitat [27] are current state-of-the-art simulator engines that can be used with multiple datasets to provide realistic measurements. Out of these two, we chose to work with the Habitat framework by Facebook Research, because of the automatically available datasets associated with it. The environment gave us great flexibility with the setup and allowed us to measure the error of our semantic segmentation from any viewpoint compared to ground truth labels.

Habitat is a simulation platform that aims to enable research in embodied artificial intelligence (AI). The system allows the training of various virtual robots and egocentric agents in a 3D simulator before applying the learned skills in the real world. The simulator supports computer-aided design (CAD) models of spaces and objects, configuration of sensors, rigid-body mechanics, and many others. Habitat is made up of two main components: Habitat-Sim and Habitat-Lab. The prior is the high-performance physics-enabled 3D simulator that we use within our pipeline. Typically, Habitat-Sim is used with Habitat-Lab, which is a modular high-level library that enables end-to-end development of embodied AI. It can be used for training agents via reinforcement learning or no learning at all to complete tasks like navigation, instruction following, or even question answering. The creators even provide some pre-trained agents for these tasks. For our system Habitat-Sim is sufficient, since we are not implementing a reinforcement learning agent.

By default, Habitat has three different datasets that can be loaded, these are: Matterport3D [4], Gibson [33], and Replica [32]. This variety of datasets makes Habitat an appealing choice for 3D reconstruction applications. Out of the possible choices, Matter-

port3D has the most number of scenes and rooms, this diversity makes it a perfect choice for training agents on RGB inputs. In our pipeline, we utilize depth measurements so we would prefer them to be of high quality, but Matterport3D lacks this in many areas. We also require the chosen dataset to contain at least class level semantic segmentation. Within the Gibson dataset, even though there are a lot of scenes, semantic object annotations are only provided for a fraction of the spaces. The geometry quality for both Gibson and Matterport3D are well behind Replica. Ultimately, we chose the Replica dataset which provides a high variety of indoor spaces to explore. Not only does it consist of photo-realistic scenes, but semantic labels are also provided. And the most appealing feature is that Replica can be loaded into Habitat by default, sparing us any implementation and data processing challenges. A few example sensor measurements from the simulator using Replica can be seen in Figure 2. We extract RGB frames, depth measurements, ground truth semantically annotated frames, along with exact camera poses from the simulator.

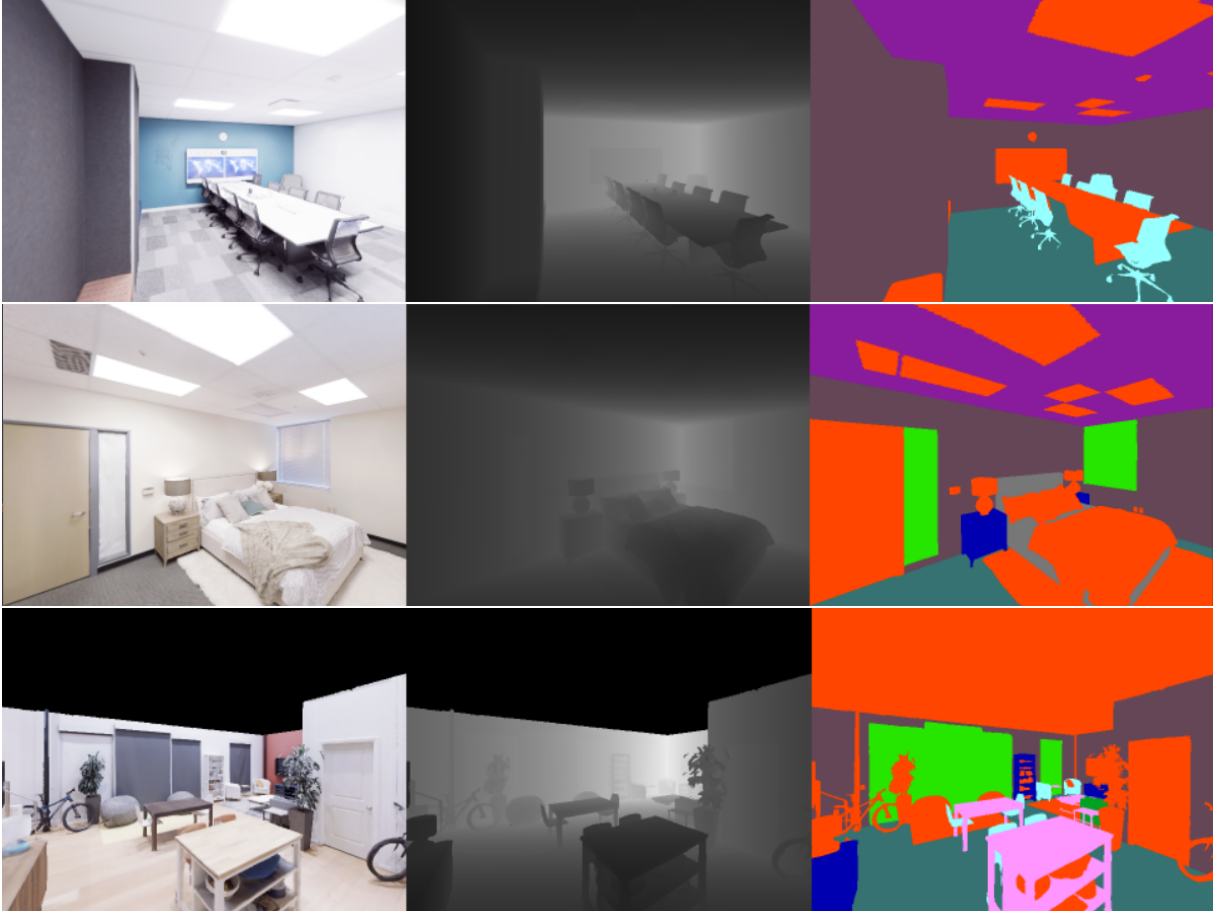


Figure 2: A few rendered images from the simulated sensors within the Habitat simulator. Each row is a different environment, the rendered frames (from left to right) are the RGB image, depth image, and the ground truth semantic segmentation obtained from the used dataset.

3.2 Real World

The long term goal of this study is to develop an application and/or software that can be used in most, if not any, indoor environment. After testing in a simulation with photo-realistic data, we planned to experiment in real-world environments. Here, we lack ground truth annotations, which would be time and resource consuming to gather, therefore we evaluate the results qualitatively. In this section we describe the tools needed for the real-world experiments, which includes a small robot that can be seen in Figure 3.



Figure 3: The ground robot used in our real-world experiments. The body is 3D printed and the brain of the agent is given by an embedded computer known as the NVIDIA Jetson Nano. We experimented with multiple cameras, the one visible is the ZED2 [31] stereo camera, but we used the Kinect for Azure camera during experimentation which can be seen in Figure 5.

3.2.1 EIT Digital Environment

We conducted the real-world experiments at the university’s EIT Digital building. Within the building, we selected two rooms which contained the types of objects that we were focusing on. The two areas chosen were two meeting rooms, a smaller and a larger one. Figure 4 depicts the selected environments.

We recorded multiple videos in each room from various perspectives. Originally, the data was recorded for our study, but since multiple projects can benefit from the experiments we shared the recordings as a custom dataset. For each room we recorded 2-3 videos from a human perspective and 2-3 videos from a low perspective (approximately 20 cm from the ground). For each video we collected the RGB frames, depth frames, and transformation information which include the position and orientation of the sensor.

3.2.2 Ground Robot

Semantic segmentation algorithms are usually trained on human viewpoint images. This means that they fail on images taken from unusual perspectives, which could come from agents such as small ground robots or even drones. We assembled a ground robot which was used to collect data from the low perspective and in the future assist in human-robot collaboration.



Figure 4: When testing our method in the real world, we used these two rooms within the EIT Digital building.

Robots are expensive, some even costing as much as \$20'000 like the Husky A200. This is why the body of our agent is given by the OpenBot [16]. The creators of this agent aimed to develop a small electric device capable of completing multiple tasks, such as person following and real-time autonomous navigation while keeping a low budget. The total estimated cost of their design is around \$50. A software stack for smartphones is also included to enable mobile operations and prove that the system is powerful enough for more advanced tasks as well. We 3D printed the frame of the OpenBot, collected and assembled the electrical components from scratch, and adapted the Arduino code to fit our needs.

The software stack was originally designed to be used with smartphones, but we switched the smartphone for a more powerful computer. The brain of our agent is the NVIDIA Jetson Nano [20] embedded computer. With the support of an active community and due to its computing capabilities relative to its size, the Jetson Nano is the perfect tool to add to our robot. It is equipped with a Quad-core ARM Cortex-A57 MPCore 1.43 GHz CPU and a 128-core NVIDIA Maxwell GPU. Having two such components gives us the freedom to distribute processes among the resources to achieve optimal performance.

3.2.3 Kinect for Azure

For reconstructing a 3D model of the environment we need depth information, so we have the option to use either a stereo or RGBD camera. Stereo cameras have two or more lenses that are used to estimate the depth with triangulation techniques. These methods work well in structured environments, but are otherwise unreliable. RGBD cameras on the other hand, measure the depth with designated sensors and are therefore dependable in most scenarios.

The RGBD camera we went with is the Azure Kinect [15], which can be seen in Figure 5. This device has a 1MP Time-of-flight depth camera, 12MP rolling shutter RGB camera, 3D digital accelerometer, and a 3D digital gyroscope. We used the RGB camera with a resolution of 1280×720 (HxW) and a 16:9 aspect ratio. The depth camera was used in the narrow field-of-view (NFOV) operation mode. The camera was fixed to the OpenBot robot body and was powered by the Jetson Nano.

3.2.4 UcoSLAM

The final component to be able to conduct real-world experiments is a pose tracking algorithm. UcoSLAM [17] is a reimplementation of ORB-SLAM2 [18], which uses sparse keypoints to optimize the movement of the camera with the help of ORB features. Similar to ORB SLAM, the algorithm works with monocular, stereo, and RGBD cameras as well, therefore it could be paired with our selected camera.

The UcoSLAM that we use in this study is an extension of the original version. Contributions added by [26] include a wrapper that allowed the algorithm to be used within the Robot Operating System (ROS) framework which is a fundamental component of the research and which we detail later on in Section 4.2. Additionally, [26] implemented a functionality to share the constructed SLAM maps among other agents for co-localization.



Figure 5: The Kinect for Azure RGBD camera developed by Microsoft that was used in the real-world experiments to gather and test our pipeline.

4 Methods

In this section we will detail the system and the methods used in our pipeline. First, we describe the proposed architecture and then introduce the Robot Operating System [30] (ROS), the core component, that orchestrates communication between the sub-modules, handles message passing and serves as the encapsulating framework for the Habitat simulator [27], which is used with the Replica [32] dataset. Then we discuss the components of the processing pipeline. Mask R-CNN [6] is used as the 2D semantic segmentation module; Kimera [24] as the volumetric reconstruction framework; a cloud filtering module [26], which was implemented for an earlier version of the pipeline, but we utilize it within ours as well; Fast-SLIC [9], our most recent contribution, for superpixel segmentation of 2D frames; and finally a label projection procedure used for a unique solution to address the problem of semantically segmenting objects from unusual viewpoints.

4.1 Proposed Architecture

In this section, we outline the design of the proposed pipeline and detail the control of information. The components mentioned here will be further elaborated in the proceeding sections.

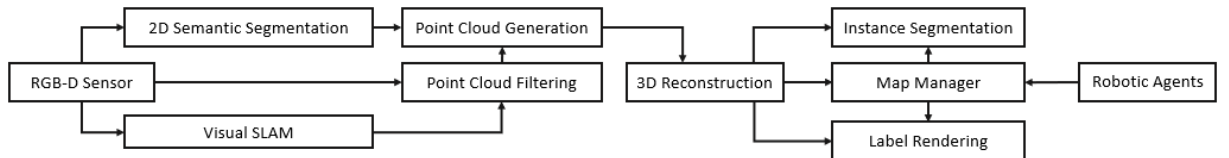


Figure 6: The originally proposed pipeline from [26]. The modular design allows for easy replacements of components. This was used in our pipeline to create the partial reconstruction from the human viewpoint.

In our work we used the pipeline from [26], which we extended with the use of superpixel technology. Our goal is to enable semantic matching between a high perspective and

an unusual, low one from which, for example, smaller agents perceive the world. With this, we hope to propose a potential method of dealing with a significant lack of semantic segmentation from non-human viewpoints. As to have a starting point, since semantic segmentation is viable from human perspectives, we present the robot or small agent with a partial reconstruction of the environment. To create the partial reconstruction, we employ the original, unmodified pipeline which can be seen in Figure 6.

The structure of our pipeline is held together by ROS [30], which we will detail in Section 4.2. Before applying an application in the real world, it can be profitable to test it in a simulated environment such as the one we used. We load a room from the selected 3D indoor environments dataset and place an agent into the scene.

The agent explores the environment, streaming its pose, RGB camera frame, and a corresponding depth map. We want to calculate the 3D position of each pixel of the given 2D frame. In the case of simulated data, the pose of the agent is extracted from the simulator; whereas during the real-world experiments this data was obtained from UcoSLAM [17]. The agent’s pose tells us its spatial position and orientation within the environment and also within the 3D reconstruction since they use the same coordinate system. After positioning the agent in the reconstructed model, we apply our label projection procedure that we discuss in Section 4.3.5.

4.2 Robot Operating System

As opposed to its name, the Robot Operating System [30] (ROS) is not an operating system, but rather an open source framework and a set of software libraries and tools that can be used in many robotic applications. One of the most attractive features of ROS is how the software is run and how processes within it communicate. This communication is peer-to-peer and allows the developer to design complex systems, without needing certain hardware knowledge. Additionally, ROS provides automatic testing tools, device drivers, communication between process running on different machines and tools for visualization as well. One of the biggest advantages of ROS is its community. It was made by the community, for the community, resulting in several reusable packages and already implemented components like trajectory planning or collision avoidance which are simple to integrate due to the architecture of the ROS.

Within the system, it is possible to work with many different processes implemented in various programming languages, allowing components written in C++ and Python to be used together. This allows us to implement deep learning based algorithms in Python, which has many libraries for precisely these type of tasks or to use C++, where faster interpretation is needed. This flexibility enables us to run processes on different hardware (GPU, CPU) as well. Within ROS, these code components are organized into packages. Each package provides specific functionalities making it a good opportunity to separate pipeline components.

The system provides a ways to connect processes, called *nodes* with a central unit often referred to as the *master node*. This master node is responsible for connecting nodes and organizing the peer-to-peer communication. Communication is done by defining publisher and subscriber connections between nodes. A publisher sends data, while a subscriber receives data over specified *topics*. Topics are communication channels where the data gets published; they are required to have a unique name and transport only specified

type of messages. A node can broadcast arbitrary number of topics and subscribe to any of the advertised topics. The before mentioned message types describe the structure of the conveyed data. There are predefined types, but custom types can be constructed from primitive types (integers, floating points, booleans, etc.) that can also be nested. Messages consist of a header and a body. The header contains the message type and a timestamp, which can be from the machine’s clock or simulated time. Defining custom message types allows us to communicate more efficiently between processes as we do not have to implement the encoding, transportation, and decoding of the sent information; this is done automatically within ROS.

We use a unique transformation management system within ROS called *frames*. When working with mobile robots we often define coordinate frames that describe absolute or relative transformations between other frames. These frames are important for example when working with a robot which has a visual sensor on top of its body and an arm with which it can manipulate its environment. Lets assume the visual sensor detects the target object. In this case the object is located relative to the coordinate frame of the visual sensor, but where is this relative to the arm that aims to interact with it. If we define a coordinate frame for the arm relative to the visual sensor, then we can simply calculate the position of the target from the arm. ROS provides a transformation management tree that shows the relationship between frames. As the tree architecture suggests, each frame can have one parent, but multiple children. All these coordinate frames can seem difficult to manage, but luckily ROS has a dedicated package to handle all transformations. The frames are described in a 6 degree of freedom (DoF) space and structured within a homogeneous transformation matrix, which is a 4×4 matrix containing rotational and translation components.

4.3 Processing Pipeline

4.3.1 Kimera

Our modular architecture is a continuation of the work presented in [26], therefore this work was also inspired by and based on the Kimera [24] pipeline. This is a real-time RGB or semantic mesh reconstruction pipeline implemented in C++. The framework makes use of simultaneous localization and mapping (SLAM), which is an algorithm used for positioning an entity that is equipped with a camera, with the help of the recorded frames. Kimera is a modular framework, which has been already configured to work with ROS. It consists of four individual modules: Kimera-VIO, a visual inertial odometry pipeline, Kimera-Mesher, a 3D mesh generator, Kimera-RPGO, which is a SLAM implemented for loop closure detection, and Kimera-Semantics, that is responsible for generating the semantically labelled 3D reconstructions. This last module is which we incorporated into our pipeline.

In the original Kimera framework their VIO module is given the task of tracking the pose of the camera. This pipeline utilizes inertial measurements, which allows pose tracking even in structure-less environments, like a completely empty, homogeneous room. Even though Kimera provides the VIO module, in [26] it was replaced with UcoSLAM [17], a SLAM variant that can provide more accurate pose estimation. It was replaced, because the application of inertial measurements still produced significant drifts in the pose, which

is detrimental to the 3D reconstruction. Furthermore, the goal was to gain from the benefits of RGBD sensors, which provide higher quality depth measurements compared to other methods that must estimate this quantity. In our pipeline, we focus on simulated experiments within Habitat, therefore for simplicity, we can make use of the exact poses that can be queried.

We used the Kimera framework within our pipeline for generating the 3D semantic mesh. Given the pose, RGB frame, and the corresponding depth measurements, this framework generates a point cloud from the depth measurement with the help of the pose and the known camera intrinsic parameters. The given RGB frame is projected onto this point cloud. The per-frame point clouds are integrated into the so-far created TSDF volume using a slightly modified version of the Kimera-Semantics [24] module. When updating TSDF scans, it is a common practice to update every voxel crossed by a ray projected from the sensor with a weighting function. [26] implemented a more general equation for updating these voxel weights. Additionally, they truncate the updates to lower memory requirements and increase overall speed. As a final step, the marching cubes algorithm is applied to transform the volume into a mesh resulting in a colored 3D model. To create semantic 3D reconstructions all that needs to be modified is the 2D image we pass to the framework. The 2D image is only used to color the generated point clouds, so if we pass 2D semantic segmented images, we will end up with a semantic 3D reconstruction of the environment. Whatever 2D image we pass will be what is projected onto the reconstruction, giving meaning to the various regions. The original map reconstruction algorithm provided by Kimera was extended by [26], these contributions include modifications that guarantee a more accurate reconstruction of the environment; We detail these improvements in Section 4.3.2.

4.3.2 Cloud Filter Node

This essential module is a contribution by [26]. As stated in Section 4.3.1, the inertia-based pose tracking was replaced with a more accurate method. To further increase the accuracy of the 3D reconstruction, it is necessary to detect false or low confidence measurements. The reconstruction errors can originate from multiple sources. When using stereo instead of RGBD cameras, the estimated depth confidence can be low. Dynamic objects within the presumed static environment can also introduce unwanted artifacts into the reconstruction. A third potential source is error within the pose estimation.

The process implemented to solve the third potential error source is called the cloud filter node. Pose estimation error can stem from drift or false relocalization from the displacement of objects or due to some redundant regions within the environment. To resolve this issue while maintaining the modular architecture, a motion model was used to detect improbable transitions between the sensor poses. Both the rotational and translational components are checked for discontinuities using the smoothness of the predicted sensor trajectory. When a faulty pose is suspected, point cloud integration is paused until a valid estimation is received.

The addition of this module is of greater significance in [26] than within our pipeline since we extract ground truth poses from the simulator. The pipeline that we extended, originally used UcoSLAM for localization. UcoSLAM estimates the pose with the help of sparse landmarks, but this can potentially introduce the previously mentioned errors.

4.3.3 Mask R-CNN: Semantic Segmentation

As described in Section 4.3.1, the Kimera framework colors the final reconstruction with the passed 2D image, which can be both RGB and semantic. This means that any 2D semantic segmentation method that gives pixel level predictions can be used within the modular pipeline. Our goal is to obtain a very high accuracy semantic reconstruction of the indoor environments to the extent that we are willing to trade computation time in return. For this reason, we decided to use the well known Mask R-CNN [6] for semantic segmentation.

Mask R-CNN is an object instance segmentation network, based on Feature Pyramid Network [12] and a ResNet101 [10] backbone. Its architecture is similar to Faster R-CNN [22], which is an object detection network that consists of two stages. First, a Region Proposal Network [23] (RPN) proposes object bounding boxes, then feature extraction takes place using Region of Interest Pool [5] (RoIPool) from each bounding box. Mask R-CNN reflects this two stage construction. Differences can be found within the second stage, where in addition to predicting class and box offset, a binary mask is generated for each RoI. During training, a multi-task loss was defined on each RoI that consists of the classification loss, bounding-box loss, and a mask loss, that is defined as the average binary mask error inside regions.

To achieve our main goal of being able to semantically segment an image taken from unusual viewpoints, Mask R-CNN could not be used, because segmentation fails from low perspectives due to being trained on images taken from human viewpoints. This is the case with most, if not all 2D semantic segmentation networks. It might be suggested to train an already existing architecture on images from a diverse set of perspectives, but this is infeasible. Training data would need to be generated, since the lack of data is not limited to semantic annotations, but also viewpoints from unusual viewpoints taken with RGB cameras. To train such a network, a custom RGB dataset from low perspectives would need to be generated with pixel-wise annotations. A task like this is extremely time consuming and requires more resources than most individuals have. For this reason, we use Mask R-CNN within our pipeline to generate the partial reconstruction of the environment from the human perspective. An example of the output on simulated and real world images can be seen in Figure 7.

4.3.4 Fast-SLIC: Superpixel Segmentation

Superpixel segmentation is a clustering algorithm designed to group regions of an image by some similarity metric such as color, texture and proximity. They have been used in many computer vision research tasks like object detection, video tracking, and even in selective search frameworks. Over the years, multiple superpixel segmentation techniques have been developed, out of which SLIC [1] (Simple Linear Iterative Clustering) is one of the most common and frequently used due to its simplicity and low computational cost. The existing superpixel methods fail to run in real time, because they usually have high latency or trade accuracy for speed. Fast-SLIC [9] is an optimized variant of SLIC, which aims to achieve significantly low runtime on a CPU to allow application without hardware limitations. Compared to other SLIC implementation, Fast-SLIC can run up to 20 times faster without sacrificing accuracy. It is able to process 1280×720 resolution images at

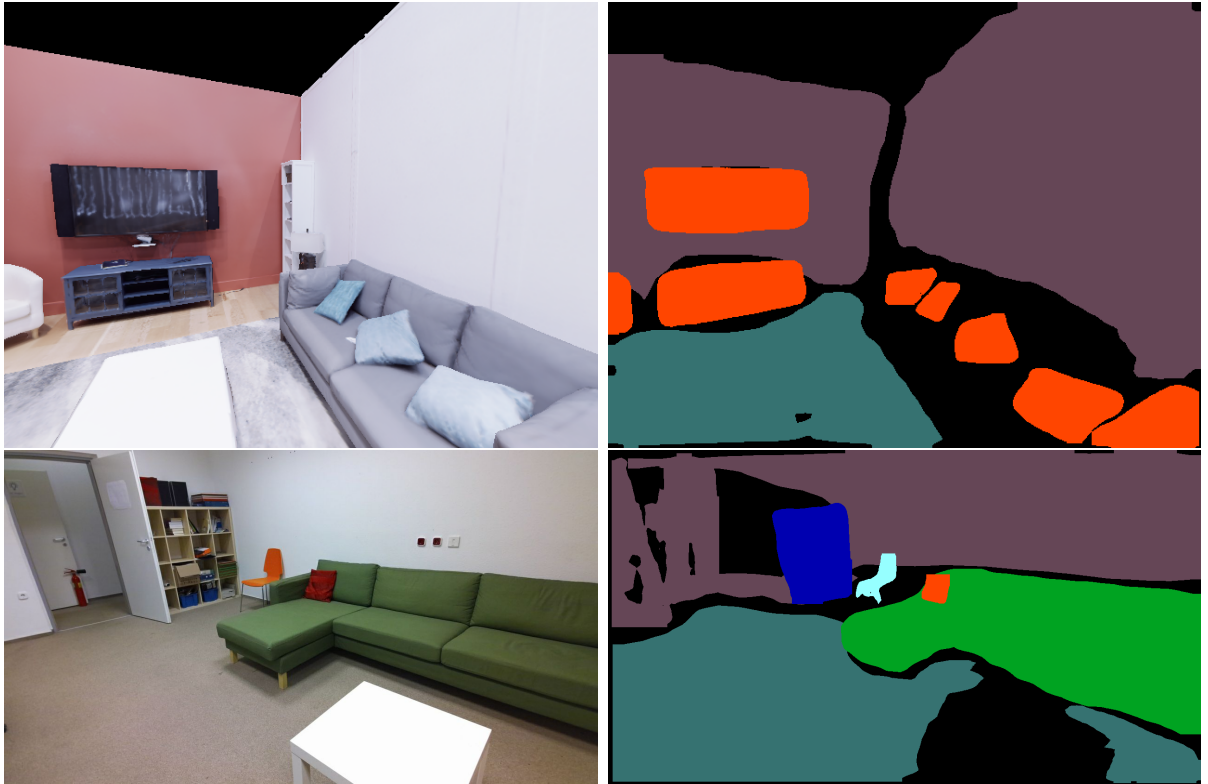


Figure 7: Two examples of the Mask R-CNN semantic segmentation, one on an image rendered from the simulator (top) and another one taken in the real world (bottom). The different colors represent different categories, while black areas signify that no label was given to that area.

60 frames per second.

The significant decrease in runtime can be attributed to multiple factors. A quantization scheme and new distance metric was introduced that is based on 16-bit and 8-bit integer-only calculations, allowing for more efficient parallelization. To reduce latency without trading accuracy, a row subsampling scheme was used. Additionally, a tiling technique was devised to enable parallel cluster assignment.

The inspiration behind Fast-SLIC was to implement a real-time superpixel segmentation algorithm. We selected this implementation for this exact reason. The original 3D reconstruction pipeline can be used in real time, so we aimed to retain this feature. Within our pipeline, Fast-SLIC is used to group the pixels in a 2D frame from the low viewpoint into regions. The idea was to identify areas which belong to the same object or semantic class, since these areas are usually similar in some way. A few representative examples of the Fast-SLIC output on frames extracted from the simulator and real-world experiments can be seen in Figure 8. Without the application of a superpixel algorithm, semantic matching would be done on pixel level, which would result in noisy and incoherent predictions.



Figure 8: Few examples of the superpixel segmentation on images obtained from the Habitat simulator using the Replica dataset (top row) and from real-world environments (bottom row).

4.3.5 Label Projections

In this section we detail our main contribution to the pipeline. We try to tackle the the problem that stems from a lack of semantic segmentation from various, unusual viewpoints like that of small ground robots or flying drones. To do this we project the labels of the semantic reconstruction created from the standard human viewpoint.

Our initial idea was to project the labels of the semantic reconstruction onto the plane of the ground. This step would be done in a preprocessing step, enabling the process to run in real time. As we were working with small ground robots, this approach was the most straightforward one, but upon further evaluation we concluded that for other type of agents this kind of projection does not make sense and would not give good results. We aimed to find a general approach applicable for any type of robotic agent. For this purpose we decided that a good choice would be to project the labels onto the camera plane. This is an optimal choice if we consider the many different builds that robots can have. It can be applied to both ground units and flying ones.

After deciding on a general approach, a further issue was identified. The problem originates from the way the 3D semantic model is reconstructed. We utilize the depth measurements from the camera, but these only tell us the surface of the objects and the environment. This means that when, for example, a cabinet is recorded we obtain the points along its surface, but in the 3D reconstruction the object will be hollow. Consider the case of a table for which the legs did not get included into the reconstruction, because it was not visible from the human viewpoint. The legs could be visible from the lower perspective. If the labels are projected onto the ground, the legs would receive the correct label. This is also the case when projected onto the camera plane. Now if in the original environment there is a vase or any other object placed on the table then the outcome may differ. For the case of the ground projection, let's say the vase is just over the legs of the table that are not visible from the upper view. In this scenario, the projection would result in the legs of the table to be assigned "vase" labels. The same spectacle can be observed when the labels are projected onto the camera plane. To achieve it, the vase would need to be placed so that during projection the table leg is between the vase and the camera.

This means that we cannot use the proposed method to segment the image from the ground agent's perspective. We can however, use this technique to extend the human viewpoint reconstruction. Therefore we suggest a procedure with which we do not solve the entire problem, but it is a step in the right direction. In the remaining parts of this section we describe the methodology in detail.

For the ground projection case, we have a preprocessing step where we project the labels onto the ground. To be more specific, when projecting the labels we want to keep the vertically highest label that is not ceiling. This means that if there is another label above the "floor" label, it can be overwritten, but only by labels that are not "ceiling". This preprocessing step means that during the actual application of the algorithm, the method is mostly reduced to a number of queries. Real time processing is achievable.

In the other, more general version that we discussed there is no need for any preprocessing steps. With the given location and orientation of the agent's camera, we place and position a virtual sensor within the 3D reconstructed model which reflects the pose of the agent and render an image. Since we are working with a semantic reconstruction, the images generated will be semantic images.

We start out from a human viewpoint partial semantic reconstruction that we assume is high quality (in reality this assumption may not hold). We say that reconstruction is partial, because when recording an environment from a single height there will be areas not visible due to occlusion. If we assume that the labelling is correct, then viewed from

any new perspective the labels within the reconstruction would be the same and therefore correct. This means that the areas we want to focus on are the ones which are visible from the new, unusual viewpoint, but not part of the reconstructed model. To obtain the unseen regions we use the depth image (we will call this the measured depth image) obtained from the camera and the fact that we know the location and orientation of the agent. We place the agent within the 3D reconstruction and render a depth image (rendered depth image). From the rendered depth and measured depth images we calculate point clouds and find areas where the values differ, meaning that they are visible from the lower view but not the higher one. We do not expect an exact matching of the points since measurements can slightly differ so we set a threshold for the difference that we do allow.

After comparing the two point clouds the task gets split in two. As we stated before, from the assumption that the given human viewpoint partial reconstruction is high quality, those areas with the point clouds that are closer than the specified threshold get their labels from the reconstructed model. Seeing the same thing from two different perspectives does not change what type of object it is and because we assume the input reconstruction to be correct we assign these labels to the new perspective.

In the other case when the points within an area differ more than the specified threshold the procedure is the following. First we identify the points that were not seen in the human view reconstruction (areas where the two point clouds greatly differed). For the floor plane projection, we project these points onto the ground plane and see what label the human perspective reconstruction projection has at the given coordinate. This means that each pixel in the original image gets a label assigned to it, unless some error has occurred and the point falls outside of the given reconstruction bounds. When using the camera plane projection method, pixels that correspond to those areas that are not visible from the human perspective reconstruction get assigned the label at the category seen in the rendered semantic image at the same coordinate. In this method, we wanted to focus more on getting higher precision with the labels. This is why we only give labels to the occluded areas when we are fairly certain that it is the correct one. We introduced a new hyperparameter which regulates how certain we want to be, we call this the “ground truth ratio”. This ratio specifies what percentage of the labels within a superpixel should originate from the human perspective reconstruction in order for the unlabelled regions to receive semantics.

As a final step in both approaches, the superpixel technology is utilized as it creates some form of clustering of the input RGB image without necessarily having to know semantics. Within each superpixel, we identify the most common label and assign this value to all pixels within the superpixel. For the camera plane projection method, this is only done if the specified superpixel meets the “ground truth ratio” condition.

When creating the reconstruction from the ground agent’s viewpoint, these semantic images are passed to the Kimera algorithm in the same manner as the Mask-RCNN semantic segmentation when the human perspective reconstruction is being generated.

5 Results

In this section we briefly describe how the experiments were conducted and present their outcomes. The process was developed in the Habitat simulation environment and later

adapted in real-world environments. To argue the validity of our research we tested the algorithm in both simulated and real-world environments. In the following sections, we detail the procedure in which the experiments were conducted. First we explain and show the results within the simulated environment and then describe the real-world tests.

5.1 Simulated Experiments

The Habitat simulation environment was used to conduct the simulated experiments. The dataset used in the simulator has instance level ground truth semantic labels for all environments. We adjusted the segmentation to class level to fit the categories within the SUNRGBD [29] dataset. Among the many possible changeable parameters, we focused on 3 main ones: the depth noise added to the simulated depth sensor, the point cloud distance threshold as described in Section 4.3.5, and the “ground truth ratio” parameter, which is also detailed in Section 4.3.5.

The final output of our pipeline is a 3D semantic reconstruction from the ground agent’s camera, which is located 20 cm from the floor, therefore mimicking an unusual viewpoint. To use Kimera to reconstruct a 3D semantic model of the environment, a 2D semantic image must be passed. During the experiments we want to evaluate our superpixel and label projection combination methods to a semantic segmentation algorithm, which in our case is Mask R-CNN. We refer to the superpixel and label projection combination methods as “vertical” for the case when we project onto the ground plane and “camera” when onto the camera plane. These methods depend on a human perspective partial semantic reconstruction of the environment. The input human viewpoint reconstruction is also generated with Kimera, but with the help of the ground truth semantic segmentation offered by the dataset. Since we trust the Mask R-CNN from this higher perspective, we recreate the 3D model using semantic segmentation obtained from it as well. This means that for specified values for the 3 examined parameters we have to perform 5 runs from which we deduct metrics. The 5 runs derive from the two projection methods, that are given either the human view reconstruction from the ground truth segmentation or the Mask R-CNN segmentation; and finally a low perspective reconstruction that utilized Mask R-CNN, therefore no starting reconstruction was needed.

To ensure that the agents traverse the same trajectory, therefore guaranteeing no advantage among the procedures. An initial step of generating paths within the environment was created. This trajectory was recorded and replayed for each run. Working within the simulation gives us the advantage that we can query the translation and the orientation of the robot agent’s camera. This convenience enables us to trust the pose of the agent and not have to worry about and take inaccurate measurements or drifts into account during the reconstruction.

We show some metrics used to evaluate the implemented procedures; In Table 2 we show quantitative metrics for our proposed ground plane projection method; Table 3 shows the evaluation for the camera plane projection approach, and finally Table 4 displays reconstruction metrics for models created using Mask R-CNN from the low perspective. In all three cases, we attempt to close the *sim-to-real gap* by introducing a parameter to add noise to the rendered depth images; this parameter adds Redwood noise. For our techniques, we experimented with different values for the before-mentioned position difference and ground truth ratio parameters. The final output of the pipeline is a triangle

mesh of the environment, this is what we compare. As we have mentioned before, the simulator comes with ground truth semantic labels for each environment, so we generate a reconstruction from the low perspective using this segmentation. In this manner, we obtain a so-called “ground truth reconstruction” from the unusual viewpoint, which we use to compare the other methods to. For evaluation we review the following metrics: Jaccard index, Sorensen similarity index, and a color accuracy similarity metric.

Room	Depth Noise	Starting Seg.	Pos. Diff	Jaccard	Sorensen	Color Acc.
<i>frl_apartment_4</i>	0	Habitat	0.05	0.7414	0.8059	0.9124
		Habitat	0.1	0.7506	0.8123	0.9116
		Mask R-CNN	0.05	0.4664	0.5055	0.5680
		Mask R-CNN	0.1	0.4656	0.5049	0.5689
	1	Habitat	0.05	0.5358	0.6770	0.8925
		Mask R-CNN	0.05	0.3370	0.4198	0.5660
	10	Habitat	0.05	0.4016	0.5535	0.8752
		Mask R-CNN	0.05	0.2150	0.2950	0.5496
<i>room_2</i>	0	Habitat	0.05	0.8481	0.9008	0.9612
		Habitat	0.1	0.8445	0.8985	0.9599
		Mask R-CNN	0.05	0.5163	0.5494	0.5874
		Mask R-CNN	0.1	0.5184	0.5515	0.5890
	1	Habitat	0.05	0.5209	0.6566	0.9466
		Mask R-CNN	0.05	0.3525	0.4493	0.6302
	10	Habitat	0.05	0.3706	0.5326	0.9069
		Mask R-CNN	0.05	0.2511	0.3630	0.6270

Table 2: Quantitative metrics of two rooms within the simulated environment using the ground plane projection method. Since the simulator gives unrealistically precise measurements, we add some Redwood noise to the rendered depth image, its distortion can be manipulated with a scalar value, where larger values correspond to noisier depth measurements and zero means no noise. The starting segmentation defines whether the given human perspective partial reconstruction was created with Habitat or Mask R-CNN semantic segmentation. Position difference defines how far apart point cloud points need to be in order to consider a certain area occluded from the higher viewpoint.

When comparing two result meshes we first transform them into semantic voxel grids then compare them with the specified metrics. For the Jaccard index, we calculate an intersection over union (IoU), where the union is the total number of voxels within the two voxel grids that are occupied within at least one of them. The intersection is the number of voxels for which the position and the color match in the two voxel grids, we call this the “color intersection”. The ratio of these two values give our Jaccard index. The Sorensen similarity index is calculated as

$$\frac{2 \times |X \cap Y|}{|X| + |Y|}, \quad (1)$$

where the numerator specifies the double of the color intersection and the denominator is the total number of voxels within the compared voxel grids. The final metric that

<i>Room</i>	<i>Depth Noise</i>	<i>Starting Seg.</i>	<i>Pos. Diff</i>	<i>GT Ratio</i>	<i>Jaccard</i>	<i>Sorensen</i>	<i>Color Acc.</i>
<i>frl_apartment_4</i>	0	Habitat	0.05	0.6	0.7431	0.8073	0.9108
		Habitat	0.05	0.2	0.7426	0.8050	0.9106
		Habitat	0.05	0	0.7516	0.8140	0.9139
		Habitat	0.1	0.6	0.7501	0.8117	0.9114
		Mask R-CNN	0.05	0.6	0.4671	0.5055	0.5671
		Mask R-CNN	0.05	0.2	0.4537	0.4947	0.5655
		Mask R-CNN	0.05	0	0.4515	0.4939	0.5673
		Mask R-CNN	0.1	0.6	0.4598	0.5007	0.5690
	1	Habitat	0.05	0.6	0.5338	0.6757	0.8941
		Mask R-CNN	0.05	0.6	0.3225	0.4093	0.5655
	10	Habitat	0.05	0.6	0.3802	0.5390	0.8748
		Mask R-CNN	0.05	0.6	0.2463	0.3384	0.5489
<i>room_2</i>	0	Habitat	0.05	0.6	0.8109	0.8780	0.9586
		Habitat	0.05	0.2	0.8453	0.8994	0.9609
		Habitat	0.05	0	0.8301	0.8895	0.9590
		Habitat	0.1	0.6	0.8370	0.8936	0.9596
		Mask R-CNN	0.05	0.6	0.5014	0.5384	0.5817
		Mask R-CNN	0.05	0.2	0.5166	0.5497	0.5872
		Mask R-CNN	0.05	0	0.5112	0.5463	0.5863
		Mask R-CNN	0.1	0.6	0.5164	0.5496	0.5875
	1	Habitat	0.05	0.6	0.4853	0.6338	0.9418
		Mask R-CNN	0.05	0.6	0.3468	0.4445	0.6255
	10	Habitat	0.05	0.6	0.3618	0.5191	0.9151
		Mask R-CNN	0.05	0.6	0.24725	0.35791	0.6433

Table 3: Evaluation of our proposed camera plane projection method on two rooms within the simulated environment. To give more realistic depth measurements, we add some Redwood noise to the rendered depth image, the larger this value the more distortion there is in the measurements, while zero corresponds to no noise. The starting segmentation defines whether the given human perspective partial reconstruction was created with Habitat or Mask R-CNN semantic segmentation. How far apart point cloud points need to be in order for us to further work with them are defined by the position difference parameter, specified in meters. The ground truth ratio (GT ratio) parameter is used to tune how many pixels within a superpixel should be visible from the provided human view reconstruction.

we investigated is the color accuracy, where we divide the color intersection with the intersection of occupied voxels.

Within one of the environments the problem that we are focusing on can be demonstrated. In the low viewpoint reconstruction when only the Mask R-CNN semantic segmentation was used, the table was not recognized and therefore wrongfully labelled within

Room	Depth Noise	Jaccard	Sorensen	Color Acc.
frl_apartment_4	0	0.3945	0.4444	0.5601
	1	0.2791	0.3503	0.5763
	10	0.2219	0.3020	0.5868
room_2	0	0.4976	0.5432	0.5963
	1	0.3479	0.4388	0.6479
	10	0.2450	0.3513	0.6546

Table 4: Besides our proposed methods, we also evaluated the Mask R-CNN reconstructions to confirm that our methods improve semantic segmentation and therefore semantic reconstruction of the environment. Like previously, we assess different noise levels by modifying the depth noise parameter, where zero corresponds to no noise.

the reconstructed model. When applying either of our proposed methods, the reconstruction accurately labels the table, no matter which type of human partial reconstruction it starts from. This phenomenon is displayed in Figure 9.

5.2 Real-world Experiments

The other set of experiments were conducted in the real world. As opposed to the simulated environments, real-world surroundings generally do not come with ground truth annotations. Generating annotations is a possibility, but it would require countless hours of tedious work. To avoid this uninteresting task, we use the real-world experiment results for qualitative evaluation.

In the simulation, the camera pose is given by the simulator, but in the real world this information is not immediately available. To estimate the translation and orientation of the agent’s camera we use UcoSLAM, which we introduced in Section 3.2.4. When conducting the experiments, we first traverse the environment with the SLAM running to create a high quality SLAM map that we use to localize the robot during reconstruction. Generating this map in advance is an important step as it will give more accurate pose estimations, which in-turn results in a higher quality reconstruction of the environment. This observation is detailed and verified by [26]. The main conclusion was that with online map creation, the final reconstruction might have drifts resulting from wrong pose estimation. These drifts are usually fixed with so-called loop closures, which occurs when the algorithm recognizes a match between the old and a new keyframe and adjusts the map accordingly.

After creating a high quality SLAM map, we use it to localize the handheld Kinect for Azure camera while recording the area and generating a reconstruction. Multiple human viewpoint and lower (robot) perspective recordings were taken. The higher vantage point recordings were taken with reconstruction of the entire room in mind, while the robot recordings only cover a part of the area. As we did not have ground truth semantic annotations of the environment and only evaluate the results qualitatively, any (partial) lower view reconstruction is sufficient.

When recording the low perspective, we attached the camera to the OpenBot robot base with a NVIDIA Jetson Nano mounted. The camera was powered by the Jetson Nano, which also ran UcoSLAM. The pose estimation algorithm must be run on the robot agent,

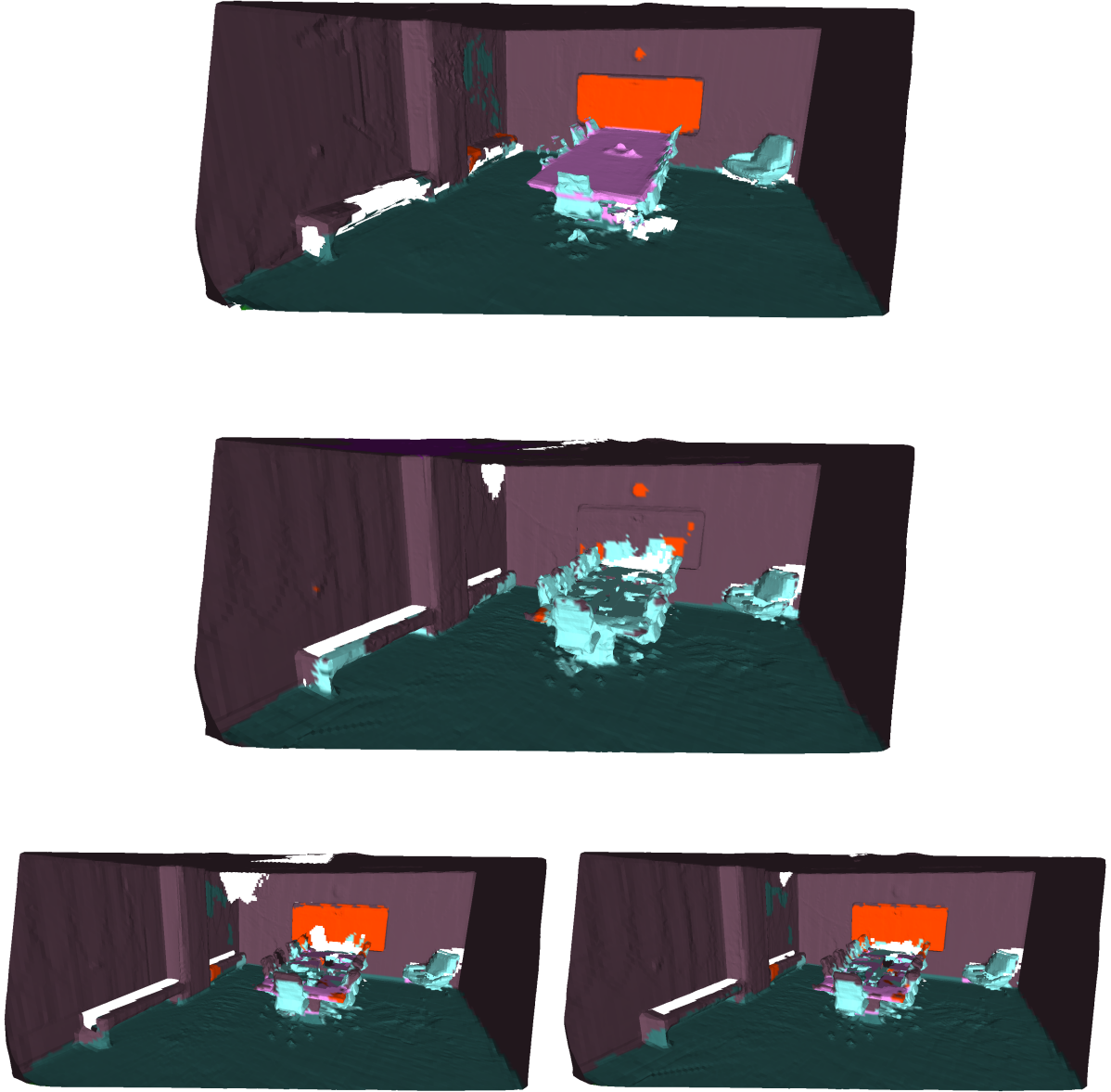


Figure 9: An example of the problem we attempt to tackle. Top image is the reconstruction of the room using Mask R-CNN semantic segmentation from on images captured from the human perspective. The objects are correctly labelled, even the table which is represented by the color pink. Middle image is the same environment reconstructed with Mask R-CNN semantic segmentation on images captured from a low viewpoint. The top of the table is not modelled since it is not visible from the low perspective, this is not a direct issue. The problem is the labelling of the table; from this viewpoint Mask R-CNN was not able to recognize and accurately label it. The images in the bottom row are the reconstructions created with our label projection methods (camera plane and ground plane respectively) starting from the human perspective Mask R-CNN reconstruction visible in the top image. In these reconstructions, the table is colored pink signifying that it is correctly labelled.

since latency of the streamed RGB images must be minimized. The recording was run on a separate device, which used to control the navigation of the robot. The agent streamed the RGB image, depth measurements, estimated camera pose from the UcoSLAM through a local network to the recording device. Recording these pieces of information allowed us to create a smaller custom dataset of the two real-world environments and to also manipulate certain parameters, such as playback speed, when later processing the data. After all the data was recorded, the 3D reconstructions were generated and evaluated.

Figure 10 displays a few rendered images from the reconstructions, which nicely demonstrate how each of our proposed methods approach the task of missing labels. Additionally, Figure 11 displays a few instances of our camera plane projection approach taken at angles in which Mask R-CNN would not be able to correctly segment the object visible on the image, but our technique manages to segment with some exaggeration.

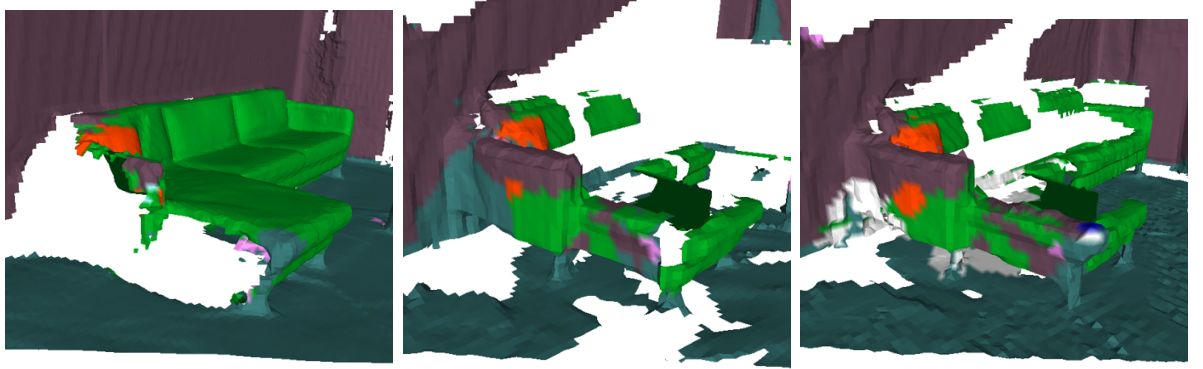


Figure 10: Reconstructions of a sofa within one of the real-world environments. On the left image, the human perspective reconstruction is visible. Here the Mask R-CNN had a difficult time recognizing the side of the sofa so we speculated that examining this area might be interesting. The results of the ground plane projection method is displayed in the middle image, where it can be seen that the sofa is extended and part of it gets labelled as "floor" (greenish-blue color) since in the human reconstruction the ground under it does get labelled. The image on the right shows the camera plane projection method's output, here we can see that the "floor" labels do not get extended onto the sofa, but instead the small mislabelled region (exhibited by the color brown) is spread over the newly observed area.

6 Discussion

In this section we go in depth regarding the results provided in the previous section. The evaluation is divided into two groups: quantitative results on scenarios ran in the Habitat simulator and qualitative results from the real-world experiments. First we iterate through the metrics obtained from the simulator and then detail our observations in the real world.

During the testing we evaluated three different metrics, which are detailed in the previous section. These metrics were used to evaluate different environments from the used dataset. Within a single room we manipulated various parameters such as which semantic segmentation is used to create the input human perspective partial reconstruction

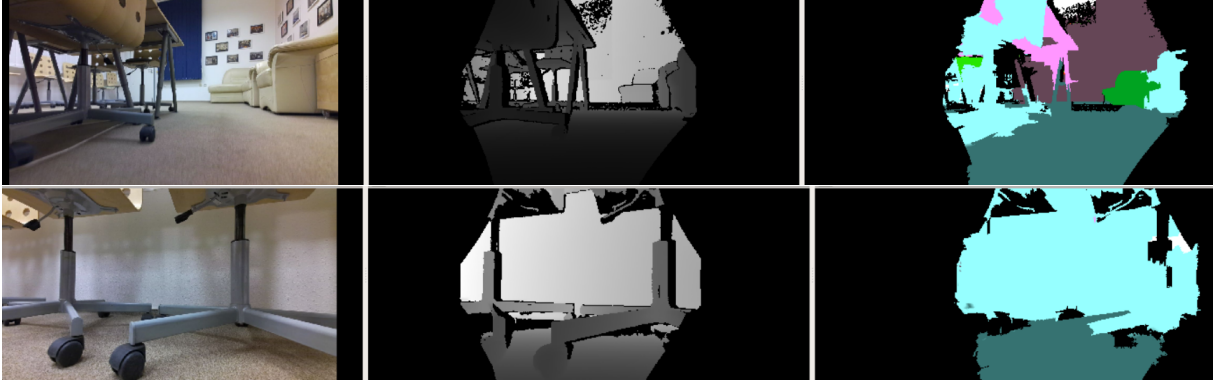


Figure 11: Two examples of viewpoints within the real-world environment when the Mask R-CNN segmentation would fail, but our proposed camera plane method correctly segments the objects with some surplus.

(Habitat ground truth labels or Mask R-CNN segmentation). Furthermore, to close the *sim-to-real gap* resulting from the employment of a simulation we tried adding different amount of noise to the otherwise perfect depth measurements obtained from the simulator. And finally, two additional parameter unique to our approach were manipulated to gauge their effect on the results. Besides our two techniques, we also ran the Mask R-CNN segmentation from the lower perspective to further get an understanding of the improvements our system enables.

We can see that the results are very intuitive. No matter which approach we use out of the three that we tested, as we increase the depth noise, the metrics decrease. Another expected observation is how the Mask R-CNN human perspective partial reconstruction runs had on average a lower score than the Habitat ground truth tests. We cannot get better results than the ground truth and the lower view reconstructions is only as good as the input reconstruction it is given. Setting the position difference parameter to a higher value means we allow a larger margin of tolerance, which in turn increases the metric score, this property is reflected in our results as well. Similarly, decreasing the value for the ground truth ratio will result in a higher score. This makes sense, because if we increase the number of labels within a superpixel that need to be visible in the inputted model, the amount of superpixels within which we extend the labels decreases.

We can also notice that both of our proposed methods show significantly higher scores for the metrics compared to the lower perspective Mask R-CNN reconstruction. This proves our main idea, that typical semantic segmentation algorithm are not fit to segment from unusual viewpoints. Furthermore, we observe that there is no compelling difference in score of our proposed methods. Although we would have expected a symbolic distinction, this can be explained with the example given at the end of Section 4.3.5, where discrepancies would occur if object would be vertically stack on each other, but there were not too many such cases.

One of the evaluated environments, namely office_4 which can be seen in Figure 9, has a prime example of when the Mask R-CNN segmentation fails. From above, the table (colored pink) is correctly labelled, but from below the table does not get recognized. Meanwhile, both of our methods, starting from the Mask R-CNN partial reconstruction manage to accurately segment the table.

The other set of experiments were conducted in real-world environments, for these qualitative evaluation is used. In Figure 10 an interesting occurrence can be observed: Mask R-CNN was not able to segment the side of the sofa. This incident allows us to more closely examine what our two approaches do in such a case. It is important to point out that the reconstructions that were generated from the lower viewpoint see less of the environment than what we would, this is also the reason why much more of the sofa is not included in the final reconstruction. For the ground plane projection method we can see that part of the sofa receives “floor” labels, which are marked with a greenish-blue color. This comes from how we project the starting reconstruction onto the ground. In the human reconstruction, the sofa was not included, but the ground underneath is, this is why the sofa receives some of its labels. For the camera plane projection case, this does not happen, but instead the wall (colored brown in the figure) gets extended onto the sofa. This happens due to those labels being in majority and a small mislabelled areas also contributes to the larger part having “wall” labels.

Finally, in Figure 11 we display a few frames of the ongoing reconstruction taken from the perspective we aimed to tackle. These frames also come from a real-world environment while using the camera plane method. The RGB image, depth image, and the semantic segmentation are visible. There are depth measurements for only a portion of the image, this property originates from the Kinect for Azure camera. Since we utilize the depth to gain semantic segmentation, only the region with depth measurements are segmented. From these viewpoints, Mask R-CNN would have a difficult time recognizing the objects, but with our approach this is possible. Most noticeable is how the chair legs are correctly annotated with some exaggeration. This exaggeration stems from the superpixel segmentation is the starting reconstruction.

7 Conclusion

In this paper we presented a modular pipeline for extending semantic segmentation maps using different viewpoints to enable semantic matching by means of superpixel technology. The modular architecture allows for effortless replacement of components as new and improved versions are developed. The pipeline was developed and initially tested in a simulated environment, then after achieving results, we moved on to testing and qualitatively evaluating in real-world environments. The simple modification of using UcoSLAM instead of the ground truth poses from the simulator for pose estimation, as was the case in [26], enables the reconstruction and semantic matching in any indoor environment.

In future work it would be interesting to explore possible improvements in the pipeline. One possibility would be introducing temporal correlation between the frames within the superpixel segmentation. Currently, the grouping is done on a frame-to-frame basis, but by incorporating superpixel tracking we may be able to reconstruct more accurate models.

A further limitation of the system is the assumption that the human perspective, partial reconstruction is static. A second camera would be needed to update the model. Implementing this feature would open a new door into the world of accurate human-robot collaboration within dynamic environments.

8 Contributions

This research is based on the work of [26]. We extended the pipeline with a superpixel algorithm that is capable of real time segmentation. Furthermore, we added a label projection procedure which aims to bridge the gap stemming from a lack of semantic segmentation algorithms from non-human viewpoints. The label project method utilizes the superpixel algorithm to obtain a relevant segmentation. We also modified the source code to improve the ease of usage. This research consists of many different components, each of which have unique hardware requirements or software prerequisites. At such a scale, managing all the version requirements become a tedious and never ending task, therefore we encapsulated the source code within a Singularity container.

9 Acknowledgements

We would like to thank Dávid Rozenberszki and Gábor Sörös, who were involved in the development of the system that we extended. Not only were they part of constructing the original pipeline, but they were also constantly available to offer a helping hand and inspired us with their ideas.

References

- [1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. Technical report, 2010.
- [2] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Rick Szeliski. Building Rome in a day. *Communications of the ACM*, 54, 2011.
- [3] Campos, Carlos AND Elvira, Richard AND Gomez, Juan J. AND Montiel, Jose M. M. AND Tardos, Juan D. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *arXiv preprint arXiv:2007.11898*, 2020.
- [4] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision*, 2017.
- [5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.
- [6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *IEEE International Conference on Computer Vision*, 2017.
- [7] Yong He, Hongshan Yu, Xiaoyan Liu, Zhengeng Yang, Wei Sun, Yaonan Wang, Qiang Fu, Yanmei Zou, and Ajmal Mian. Deep learning based 3D segmentation: A survey. *CoRR*, abs/2103.05423, 2021.

- [8] Lei Ke, Martin Danelljan, Xia Li, Yu-Wing Tai, Chi-Keung Tang, and Fisher Yu. Mask transfiner for high-quality instance segmentation. In *CVPR*, 2022.
- [9] Alchan Kim. Fast-slic. <https://github.com/Algy/fast-slic>. Last accessed: 2021-11-01.
- [10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.
- [11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 2017.
- [12] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, 2017.
- [13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *European Conference on Computer Vision*, ECCV, 2014.
- [14] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [15] Microsoft. Kinect for azure. <https://www.microsoft.com/en-us/d/azure-kinect-dk/8pp5vxmd9nhq?activetab=pivot:overviewtab>. Last accessed: 2022-05-08.
- [16] Matthias Müller and Vladlen Koltun. OpenBot: Turning smartphones into robots. In *International Conference on Robotics and Automation*, ICRA, 2021.
- [17] Rafael Muñoz-Salinas and R Medina-Carnicer. UcoSLAM: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognition*, 101, 2020.
- [18] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5), 2017.
- [19] Mur-Artal, Raúl, Montiel, J. M. M. and Tardós, Juan D. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5), 2015.
- [20] NVIDIA. Jetson nano. <https://developer.nvidia.com/embedded/jetson-nano>. Last accessed: 2022-05-08.
- [21] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3D euclidean signed distance fields for on-board MAV planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS, 2017.

- [22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015.
- [23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 2015.
- [24] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, 2020.
- [25] David Rozenberszki. Building semantic maps of the world. MSc Thesis.
- [26] David Rozenberszki, Gabor Soros, Szilvia Szeier, and Andras Lorincz. 3D semantic label transfer in human-robot collaboration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2602–2611, 2021.
- [27] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *IEEE/CVF International Conference on Computer Vision, ICCV*, 2019.
- [28] Bokui Shen*, Fei Xia*, Chengshu Li*, Roberto Martín-Martín*, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D’Arpino, Sanjana Srivastava, Lyne P Tchammi, Kent Vainio, Li Fei-Fei, and Silvio Savarese. iGibson, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint arXiv:2012.02924*, 2020.
- [29] S. Song, S. P. Lichtenberg, and J. Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2015.
- [30] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. <https://www.ros.org>. Last accessed: 2021-12-08.
- [31] StereoLabs. Zed2. <https://www.stereolabs.com/zed-2/>. Last accessed: 2022-05-08.
- [32] Julian Straub, Thomas Whelan, and Lingni Ma et al. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.
- [33] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2018.