UNIVERSITY OF TWENTE

Master Thesis

CNN based dense monocular visual SLAM for indoor mapping and autonomous exploration

Anne Steenbeek

Supervisor: Dr. Francesco Nex

2020-05-26

Abstract

UAVs for indoor mapping applications are normally equipped with bulky and expensive sensors such as LiDAR or depth cameras. However, this task should be performed using light, small and inexpensive platforms, more agile to move in confined spaces. An additional challenge is given by the absence of the GNSS signal that limits the localization capabilities of the UAV. In this research, the real-time indoor mapping capabilities using only a monocular camera installed on a commercial low-cost UAV (DJI Tello) are investigated. The limitations of traditional monocular SLAM approaches are the lack of scale of the scene and the reduced density of points in the generated map. Deep learning methods are nowadays able to estimate depths from single images, although these products are often affected by large outliers. The proposed method integrates SLAM algorithms and CNN-based single image depth estimation algorithms in order to densify and scale the data and deliver a map of the environment, suitable for exploration, in real time. The details of the implemented algorithms, the training strategy of the network as well as the tests on each element of the proposed methodology are reported in this work. The results achieved in a real indoor environment are also presented, demonstrating the potential of this solution in the rapid exploration of unknown environments.

Contents

List of Figures

List of Tables

Abbreviations

ABSREL Absolute Relative Error

BRIEF Binary Robust Independent Elementary Features

CNN Convolutional Neural Network

CPU Central Processing Unit

FAST Features from Accelerated Segment Test

GAN Generative Adversarial Network

GPS Global Positioning System

GPU Graphical Processing Unit

IMU Inertial Measuring Unit

LIDAR Light Detection and Ranging

MAE Mean Absolute Error

MAV Micro Air Vehicle

NN Neural Network

ORB Oriented FAST and Rotated BRIEF

RGB Red Green Blue

RGBD Red Green Blue and Depth

RMSE Root Mean Square Error

ROS Robotic Operating System

SFM Structure From Motion

SLAM Simultanious Localisation and Mapping

UAV Unmaned Aerial Vehicle

vSLAM visual Simultanious Localisation and Mapping

1 Introduction

1.1 Motivation and problem statement

Until a few years ago, UAVs (Unmanned Aerial Vehicles) were mainly seen in the military domain as expensive pieces of equipment, while their cost has dropped significantly in the last few years, becoming commonplace in the consumer market. People are discovering more and more possible use cases of these consumer drones. This is largely thanks to UAVs becoming cheaper and easier to operate than ever before, resulting in a lower entry barrier.

The UAV market is even estimated to reach 45.8 billion by 2025, according to [7]. Over this period, the small UAV segment is expected to grow at the highest CAGR (Compound Annual Growth Rate) with an increase in demand from both military and non-military applications. This is largely thanks to the demand for inspection, surveillance, and reconnaissance applications, both military and enterprise.

New markets and use cases are being found for UAVs at a very rapid rate. For example, farmers are using aerial photography for high precision farming. By taking multi spectral images from the air, farmers can detect which crops need the most fertilizer and which don't, allowing for very efficient use of resources. UAVs are being used for inspecting hard to reach areas. A good example of this is wind turbine inspection. Previously this was a very time consuming task, requiring long shutdowns of the turbine, and the inspection would have to be done manually. However, with the rising popularity of UAVs, this task has almost completely been replaced, requiring only a fraction of the time and cost it takes to send in workers.

Being able to set out a mapping mission beforehand and let the drone fly autonomously to collect data and give useful insights about the targeted area with minimal operator interaction has made UAVs an invaluable tool. The ease of use and low price point of current UAVs have been in large responsible for this tremendous market growth.

However, these applications still require human intervention, limiting the scalability of these solutions. In order for these systems to be easily scalable, a high level of autonomy is very important. Having to send out and train people on how to use the systems can be a bottleneck for growth for certain applications. This increased demand for autonomy in UAVs has attracted a lot more attention [8], [9], [10], [11], [12].

An area where autonomous drones have not yet become commonplace, are indoor environments. Companies like Flyability are already working on indoor inspection UAVs[13], using cages around the drone so it can safely bump into obstacles. These drones are used to inspect the insides of caves, oil and gas storage units, and sewers, to name a few. The UAVs however, are still controlled manually, requiring trained pilots to be sent out, or requiring the training of local employees before the UAVs can be deployed. This lack of indoor autonomy has several reasons. It is a lot harder to autonomously (and manually) fly UAVs indoors due to there being a lot more obstacles than in the outdoor airspace. Where outdoors, GNSS is used for localization and navigation, this is unsuitable for indoor usage due to its low accuracy and inability to function indoors. The GNSS signals do not always penetrate into the houses, and the satellite signals bounce around on surfaces, reducing the accuracy even further if they do manage to reach the GNSS receiver.

To circumvent these problems, UAVs are often outfitted with a suite of sensors that aids them with indoor navigation. For high accuracy indoor localization, external motion-tracking systems can be used, however, these require the environment to be modified beforehand, making it less versatile. In order for the UAV to localize itself completely autonomously, without external sensors, the go-to solution has been SLAM (Simultaneous Localization and Mapping). SLAM algorithms use external sensors to create a map of the environment and are able to localize within this map.

In early research on indoor navigation and localization for UAVs, LIDAR sensors were used to perform the task of both mapping and localization [14] [15]. LIDAR sensors use rotating laser sensors to measure the distances surrounding the sensor. An advantage of using these 2D LIDAR sensors was that they could utilize the navigation and path planning algorithms that were developed for ground robots, where these sensors were already commonly used. The sensor would be attached to a gimbal below the UAV in order for it to stay level. The LIDAR would be used to generate a 2D occupancy grid in which the robot can then localize using laser-based SLAM algorithms. This 2D approach was less then ideal, as one of the main features of UAVs is that they can move around in 3D space. This was also due to the limitations of the LIDAR at the time, mainly providing scans in a single plane. A problem with implementing LIDAR on UAVs is that these sensors are relatively expensive, heavy and large, especially compared to cameras. Making them unsuitable for smaller UAVs.

Improving on these solutions, RGBD (RGB+Depth) and stereo camera systems are utilized[16] [12]. These systems allow the construction of 3D occupancy maps. Relying on visual SLAM instead of laser-based SLAM, making them a lot more versatile, however, the systems often require more computation power.

Aiming to develop a fully autonomous UAV, capable of indoor flight is SKYDIO, utilizing machine learning and stereo cameras [17]. Due to its size however, it is not yet suitable for general indoor use. Where conventional stereo cameras use two forward facing cameras, these systems use multiple fish-eye stereo cameras with overlapping fields-of-view, spread out over the UAV. Their system enables them to generate 360°collision maps and perform navigation tasks. This same trend, in using machine learning for object avoidance and tracking is also done at DJI, however, they do not yet have a fully autonomous UAV commercially available.

The problem with all these approaches is that adding all these sensors not only increases their costs up into the thousands of dollars but because of the vast array of sensors required, these drones need to be larger, louder and heavier, not to mention more dangerous, all things which negatively impact their use in indoor environments.

Therefore, making UAVs as inexpensive and lightweight as possible is vital for making them suitable for autonomous indoor use. This would open up a whole set of use cases, such as drones, or even swarms of drones exploring damaged buildings that have suffered structural damage or other environments that are too dangerous for humans to explore. All this while not having to worry about losing thousands of dollars of research equipment.

In addition to this, being lightweight is advantageous for battery life and thus the range of flight. The weight of the UAV is also important for legal reasons as UAVs that weigh under 250 grams are exempt from registration with the FAA (Federal Aviation Administration) in the USA.

Therefore, solving the problem of reducing the size, weight, and cost of the UAV while still maintaining as much of the capabilities provided by the more cumbersome and more expensive sensors like LIDAR, depth cameras and GPS is vital for increasing the applicability and integration rate of autonomous UAVs for use in indoor environments.

1.2 Research identification

Using simple, small and inexpensive monocular cameras to achieve the same functionality provided by GPS, depth cameras or LIDAR for localization and obstacle avoidance would allow for a drastic reduction in cost, weight and size of UAVs. This is the minimal sensor suite a lot of lower-end consumer drones are equipped with. Monocular cameras are also capable of being used for visual SLAM, though with certain limitations, providing localization information. However, even though visual SLAM produces maps, these maps are designed to be used for localization and are not suitable for path planning or obstacle avoidance. The maps produced by visual SLAM generally consist of low density pointclouds. These pointclouds, due to the nature of visual SLAM, lack information about low texture areas like blank walls. Therefore another approach must be taken in order to obtain maps suitable for navigation and obstacle avoidance. In the before mentioned research, these obstacle maps, which can be used for navigation, are constructed using the dense depth data provided by the depth sensors, fused with the position information from SLAM.

However, a limitation of these single monocular cameras, as opposed to RGBD cameras, stereo cameras, or LIDAR, is that they lack the ability to obtain this depth and scale information, limiting their use for constructing a metric 3D map of the environment that could be used for navigation purposes.

These are all problems that need to be solved efficiently before these cameras can be used for autonomous exploration.

Therefore this research will focus on designing a system that can be used by a low cost UAV equipped with a single monocular camera to construct navigable, metrically scaled 3D maps, of indoor environments that can be used for navigation and obstacle avoidance.

1.3 Research objectives

The main objective of this thesis is to develop and evaluate a completely monocular single camera system that can be implemented on low-cost UAVs to allow for real-time autonomous indoor exploration and navigation. Adding the capabilities of dense map generation and scale estimation on top of monocular SLAM.

To achieve this, several research objectives have to be addressed:

1. Generate a navigable map of an area using only a single optical camera in real-time.
   (a) What are the existing approaches to generating navigable maps using monocular cameras?
   (b) How can a single optical camera be used to localize within an unknown area?
   (c) How can depth information be obtained from this camera?
   (d) How can the scale of the scene be determined?
   (e) What is the best method to construct a navigable map using the depth, scale, and localization information?
2. Determine the quality of the designed system.
   (a) How accurate is the localization?
   (b) What is the accuracy of the obtained scale?
   (c) How accurate are the obtained depth images?
   (d) How accurate is the final obstacle map?

1.4 Thesis structure

First the current state of the art is discussed, providing insight in current similar research, highlighting the current solutions comparing their strengths and weaknesses. This is followed by the methodology, describing the developed solution and discussing the system structure and design choices. Going more in depth into the inner workings of the system. In order to evaluate the performance of the designed system, several experiments are then conducted, aiming to give insight into the different strengths and weaknesses. Finally, the results are discussed, looking into any possible improvements that could be made in future research.

2 State of the art

In this section the current state of the art is researched. Looking into what is already available and how similar problems are being solved at the moment. First the localization problem is discussed, looking at the different methods that are used for tracking using monocular camera systems. Then different methodologies of obtaining the scale and depth information in these systems is reviewed. The different methodologies used for constructing and storing obstacle maps are then presented. Finally, works are reviewed that use different methodologies for combining SLAM and obstacle map generation.

2.1 Localization

In order to localize the UAV using only the camera data, visual SLAM (Simultaneous Localisation And Mapping) is used. SLAM is the process of creating a map of an area and localizing oneself in this map. This map can be a model or a representation of aspects of interest describing the environment where the robot is operating. The different aspects of the map differ per algorithm, in the case of visual SLAM, the map is often represented as a 3D pointcloud.

The SLAM problem has seen a rise in attention in the last decade, as, for example, the self-driving car industry has shown a vast demand for robust and accurate localization solutions. Also, the emergence of indoor applications like augmented reality, which require robust localization in GPS denied environments has pushed the field further in recent years. SLAM offers an appealing solution to localization in any environment, even if there is no preexisting localization infrastructure in place, making it very versatile.

SLAM systems generally fall into one of two categories, laser-based, and visual-based. The laser-based systems rely on LIDAR to create laser scans, which are used to create a map of the area. Recently, most of the research is directed at visual SLAM, which uses camera images to localize and map the area. Visual SLAM (or vSLAM) can be much cheaper compared to laser-based SLAM since inexpensive, off the shelf cameras can be used, and you no longer need relatively expensive LIDAR equipment.

Currently, the two major state of the art methods for visual monocular SLAM[18], direct based, and feature-based. Early examples of feature based vSLAM include PTAM [19], implementing keyframes and parallel tracking and mapping threads. Meanwhile in the direct field, one of the main works that popularized it was DSO (Direct Sparse Odometry)[20].

Feature-based methods function by extracting a set of unique features from each image. These features are unique points (also called keypoints) that can be identified in an image. By matching the same points in multiple views, these algorithms can determine the different positions from which the points were observed. An example of this so-called feature matching can be seen in figure 1.

Direct methods do not rely on a sub representation of the image, but compare the entire images to reference them, and use image intensities to obtain information about location and surroundings. An advantage of these direct methods also is that they generate a denser representation of the environment then feature-based methods that only generate

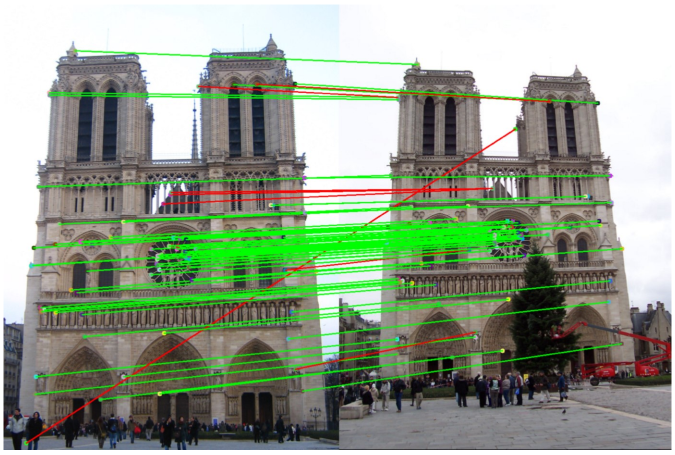


Figure 1: Example of feature matching. By matching the same features in multiple views the change in camera position can be determined.

sparse pointclouds[21]. As mentioned, SLAM creates a map, this is its own internal representation of the environment. In the case of vSLAM these maps can be represented as point clouds, where, in the case of feature based SLAM for example, each point in the pointcloud is a triangulated keypoint. These point clouds can vary in density but mainly fall into one of three categories:

Sparse pointcloud Often produced by feature-based SLAM, these point clouds contain only a small subset of the environment. Since they are mostly generated by using extraction of corners and blobs, they often only represent the regions around the boundaries of objects. Textureless areas like walls, ceilings, and floors often contain very few points as fewer keypoints are detected here.

Semi-dense pointcloud These point clouds are denser than the sparse point clouds, often showing a lot more information about the area as more and smaller details are included. However, often textureless regions are still missing in these maps. This is the type of point cloud produced by direct SLAM methods.

Dense pointcloud These maps include every detail of the environment. However, visually pleasing, they are not as useful concerning SLAM as they are often very computationally intensive to generate, and since not every detail of the environment is of importance, less dense representations are often used. These dense pointclouds are produced using algorithms like semi-global matching or image matching algorithms, matching all the pixels of the image. Dense point-clouds are typically generated as an additional step after SLAM. They are not compatible with real-time needs in most cases.

Currently, one of the most robust and complete feature-based SLAM implementations is ORB-SLAM [22], which is based on PTAM [23] and implements ORB features[24], it uses these features for all core tasks like determining the movement, creating a 3D map representation, relocalizing when tracking is lost and detecting if areas are being re-observed (so-called loop closures), making the system more efficient, reliable and straightforward. ORB SLAM has further been developed to also incorporate depth images for better performance in ORB SLAM2[25]. A significant advantage of ORB SLAM is that it is open-source, and has a very active community working with the codebase. Because of this, there are many improved and modified versions available, along with a large amount of documentation and detailed descriptions of the inner workings of ORB SLAM. A downside, however, is that it produces very sparse point clouds since, as mentioned in [25], the goal is long term and globally consistent localization instead of building the most detailed dense reconstruction. The author of ORB SLAM does suggest a method to densify this sparse point cloud in [26], as to compete with the point clouds generated by direct methods.

LSD-SLAM[27] is one of the best performing, open-source, implementations of direct monocular SLAM. It shows very promising results, and being a direct method also produces denser point clouds. This method tracks the motion of the camera towards reference keyframes and at the same time estimates semi-dense depth at high gradient pixels in the keyframes. An advantage of this method is that the semi-dense depth maps are more easily augmented for navigation purposes. However, they still avoid low gradient areas, like flat walls, leaving room for improvements.

When looking at the benchmarks in [28], it shows that ORB SLAM is more accurate then LSD-SLAM. It is also noted that ORB SLAM is more robust, as LSD-SLAM is more dependent on the quality of the camera and more susceptible to illumination changes then ORB SLAM.

2.2 Scaling and depth estimation

A shortcoming of monocular SLAM is the lack of real-world scale. This is a massive drawback of using monocular systems, especially when concerning control and navigation. The robot is unable to take into account the distances between points, as the scale being used is arbitrary. For example, sending velocity commands to the robot can have unexpected results, requiring additional feedback control systems for the system to function correctly.

For the SLAM system to work with a real-world scale, it requires an additional measurement of the real world, which can be integrated into the resulting map. Commonly used methods are depth sensors like stereo cameras[29], which use the distance between the two cameras (stereo baseline) as a ground truth measurement; alternatively, structured light cameras like the Kinect project a pre-determined pattern from which distances can be calculated. Not only can these distance cameras be used to determine the scale of the scene, but they also help with SLAM performance, having pixel-wise depth information can help with triangulating features in the map. In ORB SLAM2[25], for example, using the distance camera allows feature points to be classified into nearby and far away points. Where the nearby feature points are used for accurate translation and rotation calculations as they are often more accurate, and the far-away points are used to provide accurate rotation information but weaker translation information.

Several alternative methods have been suggested and developed to obtain the scale. An often-used system in UAVs is Visual Inertial SLAM (VI-SLAM), where IMU measurements serve as odometry and are fused with the SLAM measurements, often utilizing a Kalman filter[30]. This has an added benefit that the system is also less reliant on vision for tracking. If the vision system loses tracking, for example, due to an unlit area, or a sudden camera jerk, this can be compensated for by the odometry. In [31] the authors of ORB SLAM suggest a modification to ORB SLAM as to scale the scene using measurements collected by the IMU of the UAV.

Alternatively, a simple, and frequently implemented solution is using ground truth markers or ground control points [32]. These markers are easily recognizable patterns (for example, a checkerboard) with a pre-known size and sometimes a known location. Image vision is then used to detect the marker in the scene and scale the world accordingly. A downside of this approach, however, is that it requires augmentation of the scene beforehand, placing the markers in the area, making its usefulness quite limited.

Another method is to utilize a single depth measurement from a laser or sonar sensor[33]. These sensors utilize properties like the speed of light and sound to determine the distance to a single point. Since the scale of the SLAM maps is equal in all dimensions, it is possible to use these altitude sensors to do a linear depth measurement and correct the scale using this measurement[33].

Nevertheless, these methods all require additional sensors. New researches have been conducted in leveraging depth estimating neural networks to improve the performance of monocular SLAM systems. In [34], a method is suggested to use a depth estimating CNN in order to determine the scale for each keyframe. Following this same approach, in the work of [35] this same concept is used with even further integration into the ORB SLAM system. Here the depth input of RGBD ORB SLAM2 is modified by substituting the depth camera information with a CNN, improving the initialization procedure of ORB SLAM2. By feeding the depth estimations, also the near and far away points can be differentiated and utilized. A similar method is developed in [2] for LSD-SLAM. In this research, the neural network is not only used to optimize the performance of the SLAM system with depth information but also shows the possibilities of fusing semantic labels (pixel-wise object classification) within the constructed 3D map.

2.3 Obstacle map generation

For robots to navigate around an area in 3D, they require a map to use for navigation and obstacle avoidance. There are several methods to generate these maps from monocular images and also different methods of representing the space in 3D. ORB-SLAM2 only produces a sparse point cloud representation of the environment, which is not suitable for navigation since large portions of the environment could be missing.

In this section, different methods are discussed on how to incrementally generate a navigable map that can be used for exploration.

Mesh based One way of representing a complex geometric environment is using meshes. One advantage of these meshes is that they are less computationally intensive then point clouds, for example, since they only require a small subset of points to represent a large surface. An example of a mesh is illustrated in figure 2 In [36], a mesh-based

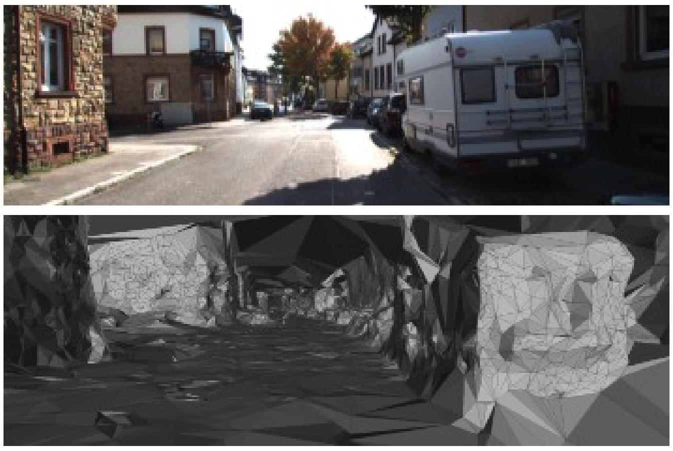


Figure 2: Example of a mesh reconstruction. Image source: [1]

approach is suggested, which would allow computationally constrained systems to reconstruct the environment using only the CPU using sparse features. This approach uses the sparse features and camera poses from either SLAM or a given ground truth to estimate the depth in each frame. This is done by triangulating the feature points.

In [1] the point cloud generated by ORB SLAM2 is used for mesh generation. Handling localization and mesh generation in the same pipeline makes the system a lot more computationally efficient. In the work of Rosinol et al. [37] this combination of SLAM and mesh generation is even further developed, where the generated mesh is also used to improve the positioning. The pose estimation is performed by ORB SLAM, and the generated point cloud is then used to construct the mesh in real-time. This resulting mesh structure is then used to detect regularities in the scene, like walls and other flat surfaces. These constraints of the scene are then used as boundary conditions to improve the pose estimation even further.

An advantage of these mesh maps is that they require relatively low computation power to generate. However, they do still have shortcomings as they are based on feature points. Requiring them to make assumptions on low texture regions where no points are detected, often assuming they are flat surfaces. They also don't address the scale ambiguity.

Direct semi-dense maps and sparse densification These are approaches that use either direct methods to produce a semi-dense point cloud or attempt to densify a sparse point cloud in order to obtain a denser representation of the environment. There are also implementations that combine both feature-based and direct methods[38]. The sparse point clouds generated by feature-based SLAM approaches like ORB SLAM are not dense enough to be used as a navigable map. In [26], the authors of ORB SLAM suggest a method to increase the density of the generated sparse point clouds using a probabilistic semi-dense mapping approach running in a parallel thread alongside their SLAM algorithm. This is based on the underlying approach of LSD-SLAM. In [39], a novel approach is implemented to enrich the sparse point cloud generated by ORB-SLAM. A K-Means clustering algorithm is used to determine the planes in which the keypoints reside. Additional points are then added within these planes. This, however, also has its limitations, as it's mainly focused on structured environments like corridors with mostly planar surfaces. A recurrent problem with the feature-based SLAM approach is their inability to obtain features in low-gradient regions. To circumvent this problem, most approaches either ignore these surfaces or assume that low-gradient regions correspond to planar surfaces, which is not always the case.

Dense pointcloud reconstruction A lot of research has already been done into multi-view stereo reconstruction [40] [41] [42] [43]. Most of the commercially available approaches for 3D reconstruction are described in (Remondino et al., 2014). However, the computational load needed by these techniques has always prevented real-time usage in robotics applications, which have been favoring RGBD cameras for this task.

More recent researches [44] [45] have opened the door to real-time capabilities due to their high parallelizability, allowing them to make use of powerful modern GPUs. This live dense reconstruction from a single moving camera is further developed in DTAM [46], VI-MEAN[47] and REMODE[48].

The construction of very dense representations of the environment is often not done in real-time due to the large amount of data that has to be processed. These dense point clouds are often more accurate but require a lot more computation power then sparser approaches.

In the survey research of [21], the state of visual SLAM and reconstruction is researched, mainly in dynamic environments. It is concluded that many advances have been made in this front, but that handling missing, noisy, and outlier data remains a future challenge for most techniques and that SFM approaches suffer from high computation costs due to their every pixel approach.

Machine learning and hybrid approaches Machine learning algorithms attempt to simulate the way that humans learn. Neural networks are trained using large datasets to recognize patterns in the data. In the case of images, Convolutional Neural Networks (CNNs) are often utilized. These types of networks analyze images and are able to recognize and obtain information from images that would be very difficult if done with conventional algorithms. Besides the algorithms of 3D reconstruction using CNN on stereopairs [17] [49], monocular depth estimation algorithms have been developed for the specific purposes of collision avoidance and depth estimation. These networks are trained to estimate depth from single RGB images. This, in combination with the position information obtained by SLAM, enables the possibility to construct dense 3D maps from these individual depth images.

Monocular depth estimation can be categorized into three learning approache: supervised, unsupervised, or semi-supervised. Supervised learning techniques train the network using labeled data (i.e depth maps), for example, obtained using depth cameras and using a loss function and regression techniques to train the network like DORN[50].

The works of [6] implemented a fully convolutional encoder-decoder architecture with novel UpProjection blocks. These UpProjection blocks enable the network to perform similarly to de-convolution blocks while requiring fewer parameters, and thus result in a smaller, faster, more efficient network.

Unsupervised deep learning approaches have the advantage of not requiring any labeled depth data, which is often more challenging to obtain, especially for large scale outdoor scenes where depth cameras are often unreliable. The work of [51] uses *epipolar geometry constraints* of stereo image pairs to generate disparity images and train the network using a reconstruction loss function. More recent works build upon this unsupervised trend, including *generative adversarial networks* or GANs, like in [52] where the network is training, and improving itself with self synthesized disparity images. The network is basically trying to continuously fool itself and then tries to learn from its mistakes.

Other semi-supervised methods like [53] have also been introduced, combining sparse depth measurements from LIDAR sensors with stereo image pairs for improved training results.

In addition to depth estimation, there is also depth completion, where sparse depth measurements are integrated into the depth prediction to, for example, fill in the gaps produced by sparse LIDAR sensors. This allows the system to attain a much higher level of robustness and accuracy. In [5], the work of [6] is improved by integrating sparse depth predictions into the network, requiring very few samples, claiming a 50% root-mean-square error improvement on indoor datasets with only 100 samples.

Recently, these neural networks are frequently being integrated into state of the art SLAM systems. In [2], LSD-SLAM, which is a direct monocular SLAM algorithm, is improved by using the CNN designed in [6], to provide scale correction. Additionally, a second CNN is running in parallel with the depth estimation network to provide semantic segmentation of the images. As shown in figure 3, these semantic labels give additional insight into the scene, differentiating between floors, ceilings, and other household objects.

In [35], the work of [6] is integrated to improve the startup procedure of ORB SLAM2 by substituting the depth images that are usually provided by a depth camera.

A recurring problem with these depth predicting neural networks is that occluding boundary regions tend to be overly smooth and shape details are lost in the process. In [54], a fascinating method is developed to solve this boundary problem, where sparse features obtained from SLAM and CNN-predicted dense depth maps are fused to obtain a more accurate dense 3D reconstruction. This approach converts the generated depth maps into a mesh for each keyframe, this mesh is then fused together with the map points of that frame to obtain a more accurate and scaled representation.

In the research of [55], a CNN is developed that uses pose information and multiple overlapping images to generate more accurate depth maps. Where the other networks only use single images, using multiple images allows the network to obtain more information about the scene to produce more accurate results.

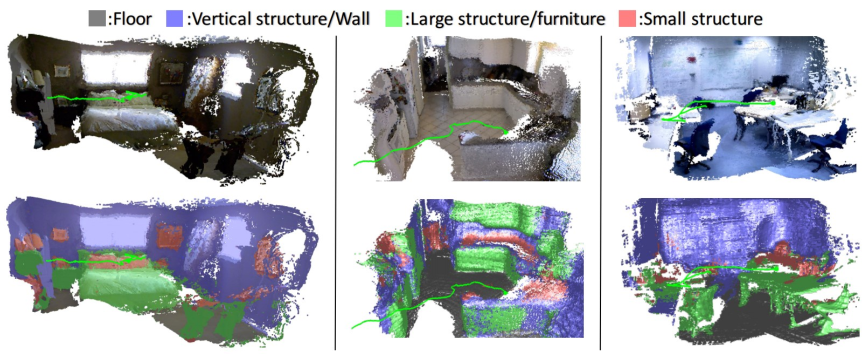


Figure 3: The 3D reconstruction data can be fused with semantic information to gain a deeper understanding of the scene. Image source: [2]

2.4 Localisation and obstacle map generation

In this section the research focusing on combining monocular SLAM and map generation with the aim of navigation and obstacle avoidance is given. Looking at the different methods that are used for localization, the different types of obstacle maps used, and the different methods used for determining the scale.

In [3] the point cloud generated by LSD-SLAM is used to construct a 3D obstacle map using OctoMap[56], which is a very memory efficient voxel mapping package. This map is then used in combination with the Move-it Package to control and navigate the UAV in 3D space. An example of the constructed obstacle map is shown in figure 4. It is noted, however, that the high processing power required by the CPU for both algorithms can be a limiting factor. Another noted limitation is the lack of scale for monocular vSLAM, which they suggest to be solved by using ground truth markers.

The developers of LSD-SLAM also developed a method for monocular autonomous drone exploration[57]. Building upon the semi-dense point cloud generated by LSD-SLAM, an OctoMap is generated. A specialized exploration algorithm is developed to take into account the fact that LSD-SLAM only determines the depth at high gradient pixels; texture-less areas are not directly observed. Due to these areas not being observed correctly, large portions of the map can remain unknown, which is something that other exploration and navigation systems do not take into account. This can cause them to become stuck on these unexplorable areas. An ultrasonic range sensor and air pressure sensor are used to determine the scale of the map. They do note on a problem with OctoMap and LSD-SLAM, the system is unable to handle loop closures. When a loop closure occurs, the poses at which the depth maps of keyframes were integrated into the OctoMap change and become outdated, requiring the whole map to be regenerated using the updated keyframe poses. This operation lasts for several seconds while the UAV is hovering on the spot, waiting until the exploration maneuver can proceed.

In [58] a slightly different approach is presented. Here VINS-Mono is used, which is a real-time SLAM framework for Monocular Visual-Inertial Systems[30], that fuses IMU and monocular image data to determine a scaled state estimation. For depth map generation, they are using a fisheye camera, from which they extract two distortion-free virtual pinhole images in order to provide a 180-degree horizontal FOV. These two images are then processed to extract a depth image. It is noted that this is a fragile and highly nonlinear process that requires accurate camera-IMU extrinsic calibration and proper estimator initialization, requiring specialized wide-angle cameras to function. A similar approach has been attempted in [47], which also utilizes VINS and TSDF-fusion[59] where spatial stereo images are used to generate a navigable map in real-time.

In [39], another obstacle avoidance method is shown. In this research, instead of using a semi-dense SLAM approach, they are using the feature-based ORB SLAM. In order to be able to use this spare point cloud for navigation, they densify it first. This is done by first fitting a plane to each cluster of points in the point cloud. The normal vector of these points is then classified using the K-Means clustering algorithm to determine which plane direction this plane belongs to, using this information, they line up groups of planes and add additional points according to the mean normal vector. The scale of the map is determined using the sonar of the UAV.

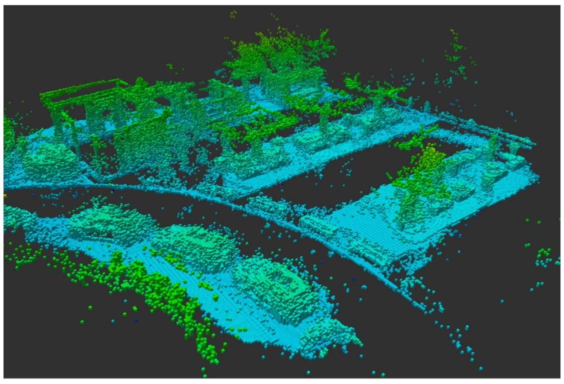


Figure 4: Example of an OctoMap obtained using the semi-dense ptointcloud generated by LSD-SLAM. Image source: [3]

2.5 Summary

For indoor localization, monocular SLAM is used in almost all research. Multiple different algorithms for SLAM are available and are either direct or feature-based. There is also a rise in using machine learning for pose estimation; however, these are not yet at the level of conventional SLAM algorithms. The two most frequently implemented are either ORB SLAM or LSD-SLAM. With the main difference between them being that ORB SLAM is feature-based, and produces sparse point clouds, and LSD-SLAM being direct based, producing semi-dense point clouds. In [3], an overview is given of different SLAM approaches being used by UAVs for indoor navigation. In this research, the results of ORB SLAM and LSD SLAM are compared, both state of the art vSLAM algorithms. Here it is concluded that even with the low-quality camera, ORB SLAM is more robust then LSD-SLAM.

The obstacle map generation shows a clear trend using the point cloud generated by the SLAM systems to generate a navigable map. These solutions either use the dense point cloud generated by direct methods to build a navigable map directly, or use a feature-based SLAM method and then densify the resulting sparse point clouds. There are also the approaches which use the SLAM system only for localization and then use a structure from motion algorithm to generate a map of the environment. These, however, struggle with real-time performance due to their high-density approach and are more often used for post-processing then real-time obstacle map construction.

The most commonly used method to obtain the scale of the map is to use external sensors like IMUs, distance sensors, or depth cameras. An added advantage of using IMUs is their ability to complement the localization provided by SLAM by providing an additional source of odometry, allowing these VI (Visual Inertial) systems to overcome temporary visual tracking failures. Recently, depth estimating CNNs, are being utilized more frequently to improve on SLAM systems, providing scale, assisting in initialization, and improving pose estimation, for example. The dense depth predictions of these networks can also be utilized to generate obstacle maps of the environment. These depth predictions can also be fused with semantic information, making the constructed maps more insightful.

For generating an obstacle map, either from semi-dense point clouds, densified sparse point clouds, or depth predictions, voxel maps are the most common solution. Mesh maps are also used in several cases. However, these are less suitable for path planning than voxel maps. The same goes for dense depth maps produced by SFM approaches like REMODE. Though visually pleasing, these dense point clouds, when used for navigation, are often converted into more efficient, lower resolution, voxel maps.

For voxel maps, OctoMap is by far the most commonly used solution. Not only is it used most often for storing the maps, but also when it comes to designing path planning algorithms.

3 Methodology

The generation of a navigable map using monocular visual SLAM algorithms is currently performed, either utilizing semi-dense point clouds generated by direct SLAM methods or by densifying feature-based methods and converting these into voxel maps. These solutions still have shortcomings, requiring external sensors for scale estimation and making assumptions about low-texture surfaces. To address these issues, a hybrid SLAM-CNN approach is developed with a single pipeline for generating obstacle maps and providing scale estimation, utilizing the sparse point cloud generated by ORB SLAM to improve the depth predictions. These depth maps are then fused, using OctoMap to generate a dense obstacle map that can be used for path planning.

In figure 5 an overview of the system design is shown.

The system is implemented using ROS, where each row is run as a node, and dataflows are managed using a publish/subscribe model which allows the individual nodes to communicate. The workings of ROS are further explained in section 3.1.

The UAV that is being used is a DJI Tello, it communicates over WiFi, and publishes a video stream. This video stream is then received by the SLAM node and the CNN node.

For positioning, ORB SLAM is used, due to its robust performance, even while using lower quality cameras and under different lighting conditions. The SLAM system takes the RGB images published by the UAV as input and uses these for localization. The localization is published as an unscaled pose, as the system is monocular in nature, it is using an arbitrary scale. The SLAM system is feature-based, so the map consists of triangulated 3D feature points. This sparse point cloud is unsuitable for obstacle avoidance, but does contain a lot of useful information about the environment. Just like the pose, the feature map generated by SLAM does not have a correct scale.

In order to obtain the scale and create dense depth maps, a Convolutional Neural Network (CNN) is used. This network takes individual RGB images as input, and uses them to compute a dense metric depth map, where the value of each pixel represents the estimated distance from this point to the camera in meters. This network first uses an initialization procedure to estimate the scale that SLAM is operating in by comparing the sparse depth map and the estimated depth map, and then uses this scale to correct for the unscaled pose and the unscaled spare depth map. After this procedure, the sparse depth map is used by the CNN to improve the accuracy of the depth images that are generated. The pose and depth estimates are then combined to generate a 3D occupancy map. For this, OctoMap is used. OctoMap provides a very memory efficient method of storing the occupancy of 3D spaces. In the following sections these components are explained in more detail.

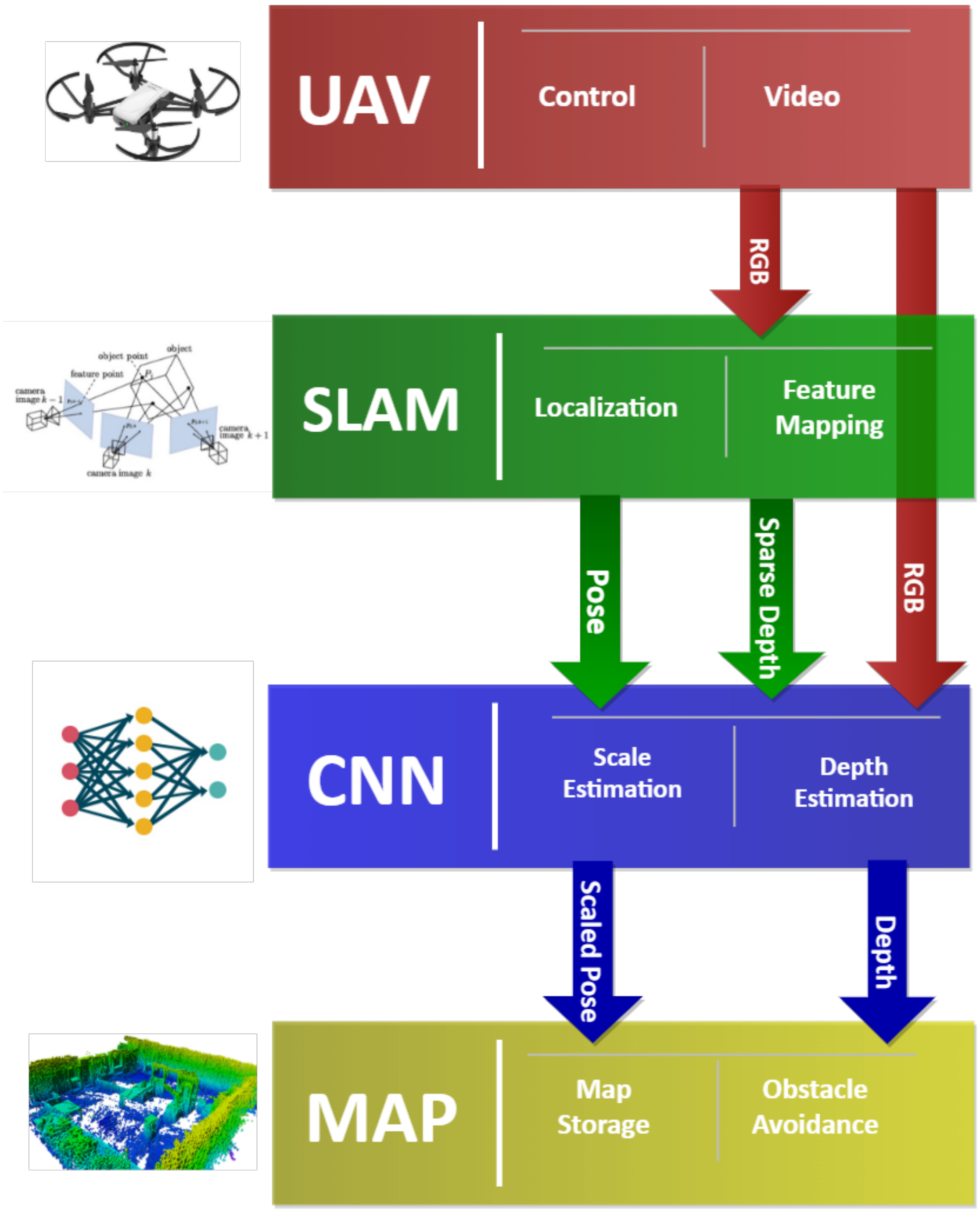


Figure 5: Simplified overview of the implemented architecture.

3.1 ROS

ROS (Robotic Operating System) is a robotics framework and toolkit, not a real operating system as the name would suggest. It provides a communication framework based on publishers and subscribers that allows different programs to transmit information like sensor data, either locally, or over the network. Each program can be run as an individual *node*. These nodes are a way to encapsulate each program and registers them with a master node. This way, individual nodes can find other nodes and communicate with them. Each node can subscribe to specific topics on which data, like camera images or other sensor readings, are published and also publish data to these topics. This allows for simple and reliable communication between different pieces of software and also makes it possible to visually inspect messages like point clouds and maps using separate helper tools.

3.2 UAV

To conduct the tests, the DJI Tello is used, shown in figure 6. This is a small, lightweight UAV, costing $99 at the time of writing. It uses WiFi for communication and an SDK that is provided to help developers. Several external libraries are available to control the Tello from a PC or laptop, allowing video streaming, receiving telemetry, and sending commands. It uses an internal positioning system, fusing IMU data and data the Vision System for more stable flight and to prevent drift. Even though the Tello has an internal IMU, the SDK does not allow reading the raw sensor data, only the positioning data. This instrument is, however, a good example of an inexpensive, off the shelf, commercial UAV.

DJI Tello Specifications	
Price	$99.-
Dimensions	$99mm \times 92.5mm \times 41mm$
Weight	80g
Propeller	3"
Sensors	Altitude sensor, barometer, IMU, 720p camera
Communication	2.4GHz WiFi
Camera	
Photo	5MP (2592 × 1936)
FOV	82.6°
Video	HD720P, 30FPS
Format	JPG(Photo); MP4(Video)
Image Stabilization	Electronic Image Stabilization (EIS)
Flight Performance	
Max Flight Distance	100m
Max Speed	$8m/s$
Max Flight Time	13min
Max Flight Height	30m

Figure 6: DJI Tello Edu edition.

3.3 SLAM

The SLAM system is based on ORB SLAM 2, this is a feature-based SLAM implementation for monocular systems but also supports RGB-D and stereo camera setups. It is built on the monocular feature-based ORB-SLAM and is adequately named after the ORB (Oriented FAST and Rotated BRIEF) descriptor. Currently, it is one of the best performing SLAM systems and is ranked as the third-best performing open-source visual SLAM implementation on the KITTI benchmark.

3.3.1 ORB feature descriptor

ORB SLAM is feature-based, meaning it uses feature matching to determine the position. There are several methods to find these features in an image; in this case, the ORB descriptor[24] is used. Oriented FAST and Rotated BRIEF (ORB) has been developed as an efficient and viable alternative to SURF. ORB is, in short, a fusion of FAST keypoint detector and BRIEF descriptor with several modifications to improve the performance.

ORB SLAM creates an image pyramid. The reasoning behind this is that it allows for feature extraction on different levels of scale. Due to the nature of image features obtained from FAST, they might miss out on features when only using a single level of scale. This can be solved by using an image pyramid, as shown in figure 7. An image pyramid is a multiscale representation of a single image where each layer consists of a downsampled version of the previous layer. ORB detects key points in each layer of the image using FAST, making it partially scale-invariant. The number of scale levels and the scale factor are tunable parameters in ORB SLAM, but commonly eight levels are used with a scale factor of 1.2.

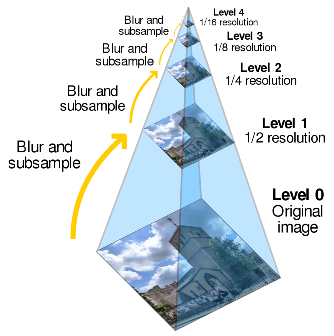


Figure 7: An image pyramid with five levels of scale. [4]

All the matched key points that have been found by the FAST algorithm are then converted into a binary feature vector that uniquely describes the feature. This is done using BRIEF[60]. The default implementation of BRIEF, however, cannot handle rotations, so ORB SLAM then uses an improved version, namely rBRIEF(rotation-aware BRIEF). rBRIEF uses an optimized pattern for picking the test points, which was learned from a large set of points by maximizing variance and minimizing correlation.
After all the features are extracted, ORB SLAM undistorts them using the provided camera parameters. This means ORB SLAM never has to undistort the whole image, only the features. This allow to reduce the computational time preventing the need to undistort the complete image.

3.3.2 Tracking, mapping and loop closure

The system of ORB SLAM consists of 3 main threads running in parallel, each handling a separate task in the process. The tracking thread takes in new images and uses them to estimate the new position, the local mapping and loop closing threads are responsible for building and optimizing the map of the area. Each of these processes is described in more detail below.

Internal map The map generated by ORB SLAM consists of *map points* and *keyframes*. Keyframes are special frames that store information about the camera pose, camera parameters, the extracted ORB key points for the frame, and the observed map points. The reasoning behind using keyframes is that storing every frame would be wasteful, only frames that contain a lot of information are worth storing. Each map points represents an observed feature in 3D world coordinates and stores information about every keyframe from which it was observed.
ORB SLAM makes use of graph theory. These graphs are mathematical structures used to model relations between objects. These graphs consist of nodes, and these nodes are connected by edges representing their relation. Keyframes and map points are used to build a *covisibility graph*. In this graph, keyframes are represented as nodes. If two keyframes share at least 15 map points, the nodes are connected.

Tracking The tracking thread takes in new camera frames and uses these for estimating the current pose. It is also responsible for deciding when an image should be used as a new keyframe. When a new frame is received, the ORB feature points are calculated. By using feature matching between the current frame and the previous frame, the camera movement is determined.

Then, a decision is made if the current frame should be used to create a new keyframe or not. ORB SLAM creates a lot of keyframes to be more robust to hard to track camera movements, especially rotations. However, they later go on to discard a lot of redundant keyframes to be able to keep the computational cost of the mapping low. In order for a frame to become a keyframe, the following criteria must be met:

- 20 frames must have been processed since the last time tracking was lost, in order be certain relocalization was successful.
- The local mapping thread must be idle. However, if more then 20 frames have been processed since the last keyframe, the local mapping thread is paused to allow for a new keyframe to be inserted.
- The frame must contain at least 50 feature point matches.
- The number of point matches between the current frame and the reference frame must be less than 90%.

Local Mapping The local mapping thread takes new keyframes generated by the tracking thread and is responsible for keeping the map up to date and optimizing it. When a new keyframe is received, it first checks if any map points should be removed if they are not reobserved often enough, even though they should be visible. This could be, for example, because objects were moved. After this, new map points are created by triangulating feature matches between the new keyframe and the keyframes with which it shares the most map points.

When no keyframes are waiting in the queue, local bundle adjustment is performed. Bundle adjustment is a method to jointly optimize the estimated camera positions and the triangulated position of the 3D map points. In order to determine which parameters and frames should be used for this bundle, the covisibility graph is used. This is done using the graph optimization library g^2o, which is able to deal with the large amount of parameters efficiently by restructuring the optimization problem.

Loop closure Over time, small errors in position can add up. This causes position drift. So when traveling large distances and then returning to the same position, chances are the start and end position do not exactly overlap due to all these small errors. Loop closure is used to solve this. When the same scene is observed after a certain traveled distance, the drift error can be determined and corrected.

To do this, the similarity between the most recent keyframe and its neighbors in the co-visibility graph is calculated. The lowest score is then compared to the rest of the keyframes. If the similarity score of another keyframe is higher than the lowest similarity of one of the neighbors, the keyframe is considered a loop closure candidate. The candidate is only used if three consecutive possible candidates have been found.

After a loop closure, global bundle adjustment is started to optimize all the keyframes within the loop that was closed.

3.3.3 Sparse depth map

In most state of the art depth estimation networks, only the RGB image is used. However, SLAM also produces a sparse depth map by triangulating the ORB feature points between frames. By combining this depth information with the RGB image, a more accurate depth estimation can be obtained.

In order to transfer this sparse depth image to the CNN, the map points in the keyframe are converted into a depth image which is published using ROS. The distance between each visible map point and the current camera position is calculated, then the x and y pixel coordinates of this point in the current frame are determined to construct a sparse depth image.

The image is then shared using ROS topics with other nodes that need it. The distance of each map point to the camera image is still unscaled at this point. This scale is later determined and compensated for in the depth estimation network.

3.4 CNN based depth estimation and densification

Single image depth estimation uses RGB images as input to estimate the distance from the camera to each pixel in meters. This is done using a depth estimating Convolutional Neural Network (CNN). These types of networks are trained using a large amount of images as input, and then continuously improving it by minimizing the error between the output of the network and the actual desired output.

SLAM also produces a depth map, however, this depth map only consists of a small amount of triangulated feature points, and has an arbitrary scale, making it unsuitable for obstacle avoidance. By fusing the sparse depth map generated by SLAM with the CNN, a denser, more accurate depth map can be obtained.

These sparse depth samples are used as an additional input for the CNN to obtain more accurate depth estimates. This is done by modifying the CNN to not only taking a RGB image as input, but also a sparse depth image, containing a small amount of depth samples triangulated by SLAM.

An example of the inputs, output and ground truth is shown in figure 8, where the RGB input of the network is shown in (a), the sparse depth input is shown in (b), the output of the CNN, the predicted depth image, is shown in (d) and the ground truth is shown in (c). The depth images in this figure are shown in a color format for clarity, however, in reality the produced depth map consists of only a single channel, where each pixel value equals the distance in meters.

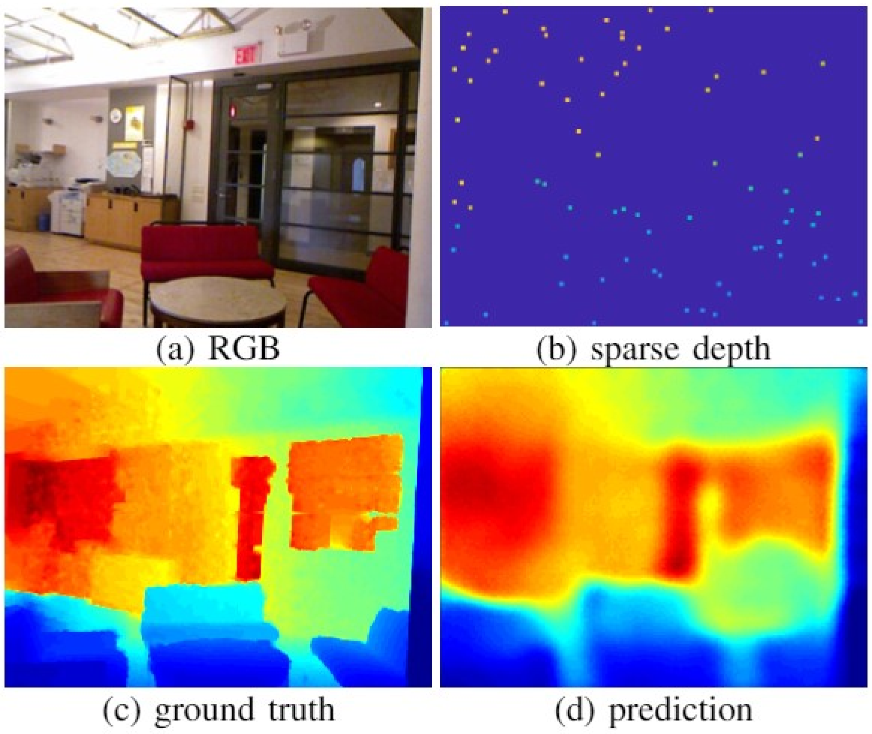


Figure 8: Sparse to dense Example. Image source: [5]

In this section, the inner workings and development of the depth estimation network are described.

- **Network Architecture** A description of the structure of the depth estimation network.
- **Data augmentation** How the images are augmented before training.
- **Sampling** How the sparse depth samples are obtained for training.
- **Training datasets** Description of the dataset used for training.
- **Scale estimation** Describes how the scale is determined from the depth samples.
- **Intrinsic parameters of the neural network** Goes into detail about how the network handles different camera parameters.

3.4.1 Network architecture

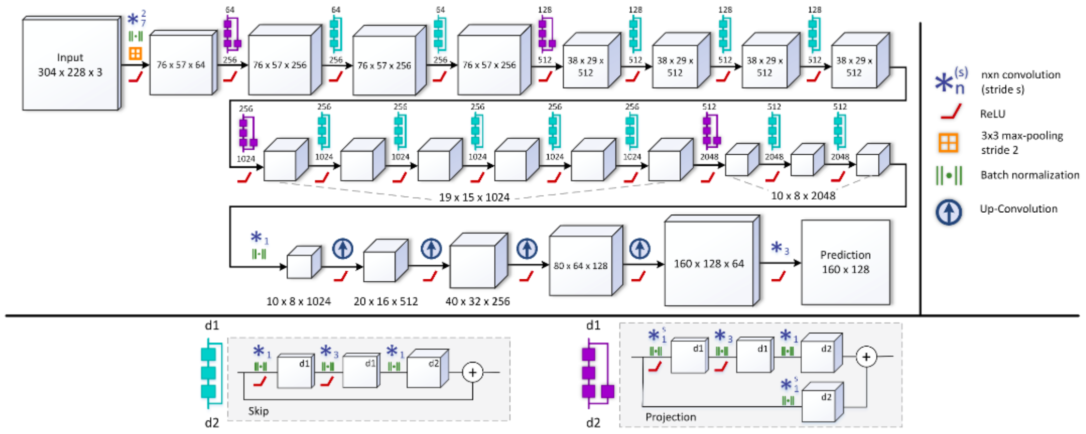


Figure 9: The network architecture builds upon ResNet-50, however, the fully-connected layer at the end of ResNet is replaced by a decoding layer (bottom row). Image source: [6]

The depth estimation network used to complete the depth predictions is based on [5], which uses sparse sets of depth samples and RGB images as input to predict a high resolution depth image. The architecture is based on the state of the art depth prediction system by Laina et al [6], shown in figure 9.

This architecture has an encoder, decoder structure, where the encoder process the input information and converts it into so called *feature maps*, these feature maps are then stacked together to produce the output of the encoder. These feature maps are a denser representation of the input information, storing more detailed information about the observed data in a format that can be processed by the decoder component of the network.

The goal of this encoding layer is to give the higher-level neurons larger receptive fields, thus capturing more global information about the image.

The encoding layer of the network is based on ResNet-50[61], which uses Residual connections that allow the network to retain information between layers, this allows for much deeper networks that would otherwise not be possible.

In the realized implementation, the weights resulting from training using imagenet [62], which is a large dataset consisting of images of rooms, animals, materials among other things, have been used as starting point for training.

Training such a complex network from scratch requires large amounts of time and computing power, unavailable to most. To avoid having to retrain the network from scratch for the task of depth estimation, a method called transfer learning is utilized. The same weights that were used from imagenet are used as a starting point for encoding data into the feature map. However, the last average pooling layer and linear transformation layer, found at the end of ResNet, used to attach the labels to objects in the images, is replaced with a new depth decoding layer designed by Laina et al[6]. This depth decoding layer uses an up-sampling strategy with *up-projection* blocks. Chaining these up-projection blocks allows high-level information to be more efficiently passed forward in the network while progressively increasing feature map sizes. This enables the construction of their coherent, fully convolutional network for depth prediction. This fully convolutional approach allows for an architecture that contains fewer weights while still providing good results.

A loss function is used as a metric for optimizing the network during training using regression. Experiments in both [6] and [5] have shown that $\mathcal{L}_1$ shown in equation 1 produced the best results on the RGB-based depth prediction problems. This is the Least Absolute Deviations function, which tries to minimize the error of all the absolute differences between estimate and the ground truth.

$$\mathcal{L}_1 = mean(|y - \hat{y}|) \tag{1}$$

3.4.2 Training datasets

For training, the NYU Depth v2[63] dataset is used. This dataset contains 48521 indoor images, of which 655 are used for validation, the rest is used for training. These images are all of the indoor scenes and have been recorded with a Kinect camera at a resolution of 640×480 pixels. This dataset also includes labeled depth images, which would allow for training semantic information into our network in the future. The dataset includes different types of rooms, like basements, bathrooms, bedrooms, offices, and dining rooms, all taken from 3 different cities. An example is shown in figure 10, which shows different rooms with their depth images and class labels. The kinect uses a structured light sensor to estimate the depth. By projecting a pre-determined pattern of infrared light, and observing the deformation of this pattern, the depth of the observed scene can be calculated. However, the random error of depth measurements increases quadratically with range, reaching 4 cm at 5 meters [64], which is the described maximum range of the sensor.

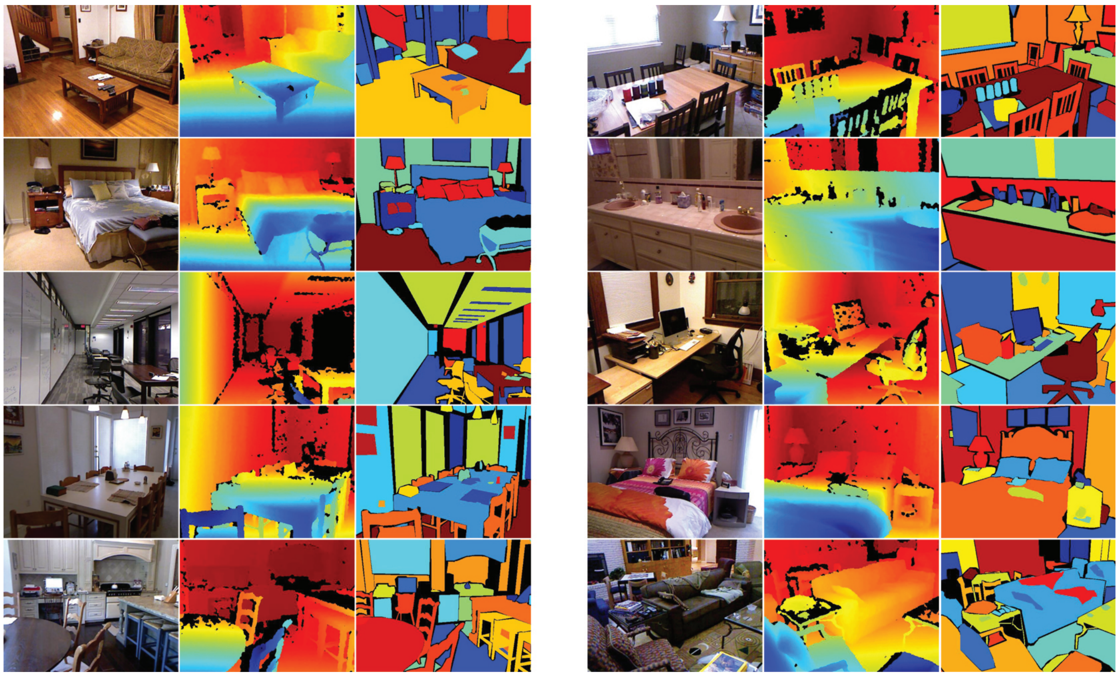


Figure 10: Samples of the RGB image, the raw depth image, and the class labels from the NYU Depth Dataset V2.

3.4.3 Data augmentation

In order to improve the robustness of the network, the training data is modified before each training iteration in order to prevent oferfitting and get more usage out of the same amount of data. This is done in the following ways:

- *Rotation*: Color and depth images are rotated at random to simulate different observation angles. The angles are chosen at random between $r \in [-5,5]$
- *Color Normalization*: The RGB images are normalized using mean substitution and division by the standard deviation.
- *Color Jitter*: The brightness, contrast and saturation of the RGB images are each scaled at random by $k_i \in [0.6,\ 1.4]$.
- *Flipping*: There is a 50% chance that the image is flipped horizontally.
- *Sample noise*: In order to help the network better handle small reprojection errors, a small amount of noise is added to the distance measurements.

In addition to these pre-processing steps that are used only for training, the images are also cropped at the center to make sure the input to the network is a consistent size.

3.4.4 Depth Sampling

The training dataset provides a series of RGB images and their matching depth images. The network uses an RGB image and a sparse depth image as input and evaluates them using the depth images as ground truth. The RGB images and depth images are provided with the dataset, however, the sparse depth images are not, and still need to be generated during training.

In order to provide the network with sparse depth samples during training, so called sparsifiers are used, converting the ground truth depth images into sparse subsets that can be used as input during training. The most common way to design this sparsifier is to randomly sample points from the depth image. However, this does not adequately represent the way 3D points are obtained in ORB SLAM.

To make sure the depth estimation network is optimally using the depth samples obtained by ORB SLAM, an ORB based sparsifier is developed. This allows the network to take full advantage of the information provided by ORB points, as ORB uses an edge detector, which provides samples mainly at the edges of objects. This way, edges of tables, for example, provide significantly more samples than the surfaces of tables. This information can be used by the network to improve the depth prediction.

A downside of the ORB edge detector is that textureless areas do not provide as many feature matches, making it harder to get depth estimates in these areas, often resulting in lower accuracy. This can either be mitigated by filtering out low texture areas when doing reconstruction, taking an area around each sparse sample and then creating a convex-hull surrounding these points.

Running an ORB detector on an image provides a large set of key points, this can be up to several thousand points per image. However, in ORB SLAM, not all these keypoints are translated into map points. Thus the number of depth samples during training should also be limited to best match the actual implementation. To accommodate for this, the sparsifier can be trained with a varying number of sparse samples, keeping it around a specific expectation. This way, the network is trained to be more robust to changes in the sparsity of the samples in different frames.

3.4.5 Intrinsic parameters of the neural network

A drawback of neural networks is that the intrinsic parameters, like focal length, of the camera that is used to record the training data, are embedded into the network weights. Not only this, but the network also crops the images to make sure the input size is consistent, changing their resolution. To correctly reproject the depth images into the world coordinate system, these differences need to be reflected in the focal length and depth images.

To correct the camera's intrinsic parameters of the sensor for the change in resolution resulting from cropping the image equation 2 is used. Where f is the focal length in the x and y directions of the frame, c is the optical center, and r is the ratio by which the scale is reduced from input image to output of the CNN.

$$A_{nn} = \begin{bmatrix} f_x r_x & 0 & c_x r_x \\ 0 & f_y r_y & c_y r_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

3.4.6 Scale calculation

As mentioned before, monocular SLAM uses an arbitrary scale. This limits the applications of monocular SLAM tremendously. The point-clouds produced by SLAM are therefore scaled using the information obtained using the trained CNN that provides metric depth information of the scene. This would otherwise require additional external sensors to correct, however, using the CNN this can all be done using only the camera.

In order to correct for this arbitrary scale, an initialization procedure is introduced, where the scale ratio between the SLAM pointcloud and CNN depth estimations is determined. This is done over a set amount of keyframes, using the triangulated map points obtained by SLAM and the matching depth estimates obtained by the CNN. By determining their ratio, the scale is obtained.

During this initialization procedure, the sparse depth map is not used as input for the CNN. Because of this, the results of these initial depth estimations are less reliable, which has to be taken into account when determining the scale.

As result of the CNN no longer having access the sparse depth measurements as input, a side effect is that it no longer provides depths at further distances. The maximum depth the CNN predicts is limited to roughly 4 meters. This is most likely due to the fact that at these larger distances, the CNN depth estimates, obtained only from RGB data, are relatively less accurate, and the resulting errors are larger then the errors at closer ranges. This could cause the CNN to rely more heavily on the more accurate sparse samples at these longer distances compared to the shorter rangers. Causing the CNN to ignore depth information contained in the RGB data at these ranges.

To improve the results of the scale measurements, a modified z-score (also called the standard score) is used. This is a numerical measure to describe a value's relation to the mean of its group. In this filter, the median of the group is used instead of mean values, as the median is more robust to outliers.

Using equation 3, a vector of ratios is determined, where D^{orb} and D^{nn} are the vectors of the depths of each ORB map point that has been observed up to now, and their corresponding pixel depth value determined by the CNN. Then the MAD (median absolute deviation) is determined4, where $\bar{S}$ is the median of the observed scales. Then equation 5 is used to determine the scale with respect to the MAD score for point i. Equation 6 is then used for filtering, where f is the threshold value used to determine if the scale at index i should be omitted. Larger values of f result in a broader range of scale indexes to be included. The scale s^{orb}_{map} is then determined by minimizing the squared error after filtering. In figure 11, the results of the median filtering can be observed for different values of f with the black x-axis representing the mean of the unfiltered data.

Using this filtering approach, a more accurate scale estimate can be obtained. However, in order to obtain as many usable scale measurements, it is recommended to mainly observe nearby objects.

$$S = \frac{D^{orb}}{D^{nn}} \tag{3}$$

$$MAD = median(|S - \bar{S}|) \tag{4}$$

$$M_i = \frac{S_i - \bar{S}}{MAD} \tag{5}$$

$$\delta_i = \begin{cases} 1, & M_i < f \\ 0, & otherwise \end{cases} \tag{6}$$

$$s_{orb}^{map} = \underset{s}{argmin} \sum_{i=0}^{n} \delta_i (s D_i^{orb} - D_i^{nn})^2 \tag{7}$$

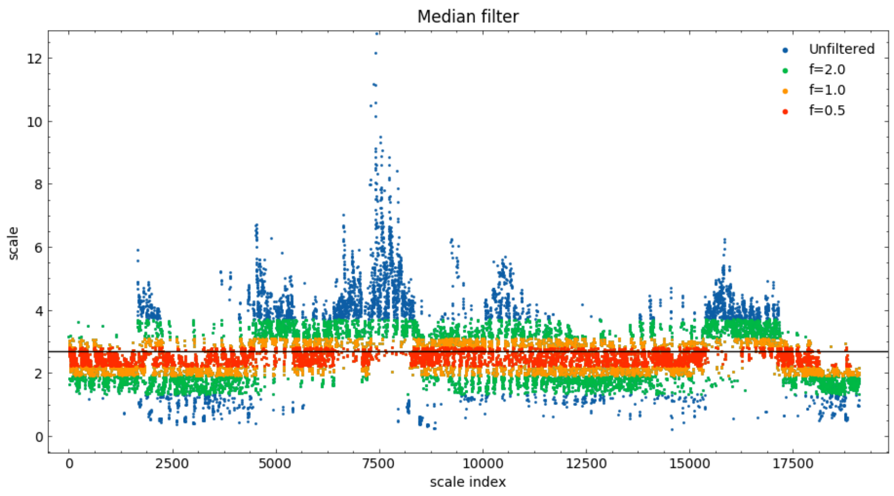


Figure 11: Median filter for different threshold values of f. The x axis represents the mean of the unfiltered data.

3.5 Depth image filtering

The depth estimations of neural networks can have sub-optimal solutions at distance jumps, for example, when looking over an object like a desk, with a wall in the far background. Here the network makes smooth edges, going from the desk to the wall. An example of such an issue is shown in figure 12. Here the camera is looking over a cubicle with a wall in the background. For the error metrics, these gaps are not detrimental, as they only occur in a small subset of the whole image. However, when used to construct an obstacle map, these floating pixels can have a negative effect, especially when fusing depth maps from different perspectives.

To improve the quality of the depth estimates, they are filtered using a *statistical outlier removal*. In this algorithm, all points in the point cloud are stored as a KD-tree, which is a data structure that allows the data to be efficiently searched for nearest neighbors. For each point, the mean distance from it to its n nearest neighbors is determined. Then all points whose mean distances are outside an interval defined by the mean and standard deviation of the entire set are considered outliers and removed from the image.

In the figure, the original estimate is shown in red. Here you can see that the jump from cubicle to the wall is filled with very sparse points. The filtered point cloud is shown in blue, removing these distance jumps and improving overall obstacle map quality.

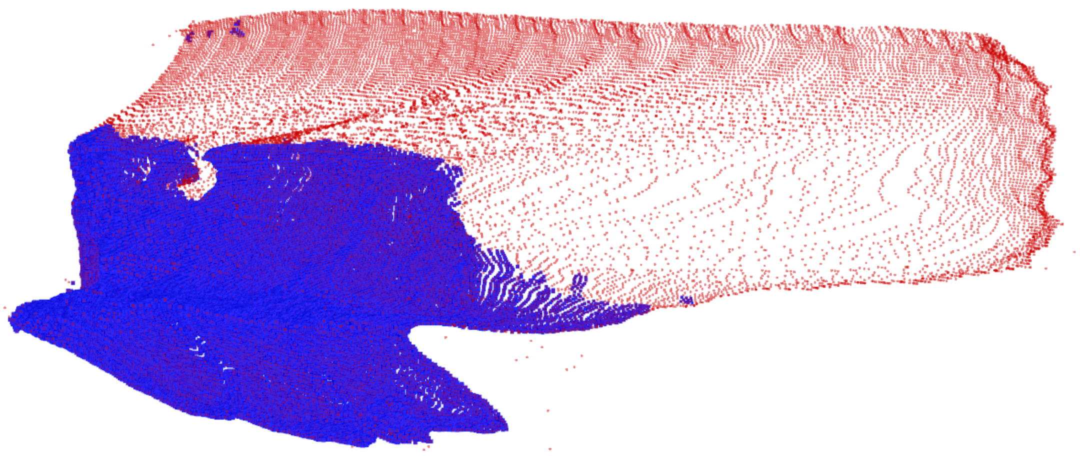


Figure 12: Example of depth estimation crossing distance gaps. Filtered image is shown in blue, raw data is shown in red.

Another improvement that can be made with filtering is in low texture areas. Due to ORB using an edge detection algorithm, points in low texture areas like walls are often missing, causing these areas to reduce the overall quality of the 3D reconstruction often. These areas can also be filtered out to improve the reconstruction. This is done by combining two filters.

First, a radius around each depth sample is selected, as the estimate close to triangulated map points are often more accurate. Then a convex hull (see figure 13) is used that encloses these points so that the depth information bounded within this area is also included. This way surfaces surrounded by sample points that don't include a lot of texture themselves, for example, a desk, where the corners of the desk have matched features, but the desk itself is bland, are still included.

3.6 Map construction

Ground-based robots often use 2D occupancy grids for path planning. This works since these robots only move in a 2D plane. UAVs, however, can move in 3D space, requiring a different type of map.

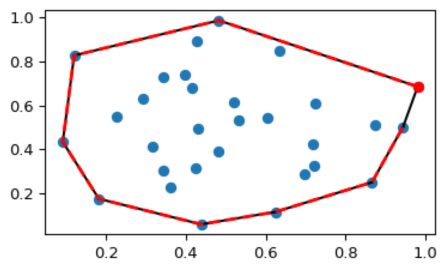


Figure 13: Example of a convex hull, a surface encapsulating all the points within, with no corners bent inwards.

In order to store and fuse the depth estimate from each frame, OctoMaps[56] are used. Octomaps provide a very memory efficient and relatively fast probabilistic 3D mapping procedure. This means that it can integrate probability into the occupancy, allowing them to take into account measurement noise and incorrect measurements that are being reobserved. In addition, octomap is able to differentiate between, free, occupied and unknown space (see figure 14).

Octomaps are also very memory efficient. They can transform point clouds into a much more efficient structure. A point cloud of, for example, 3 million points can be depicted as only a few hundred voxels (see figure 15). As in the case of obstacle avoidance, there is very little loss of valuable information, as long as the voxels are of an adequate resolution to be able to avoid the obstacles.

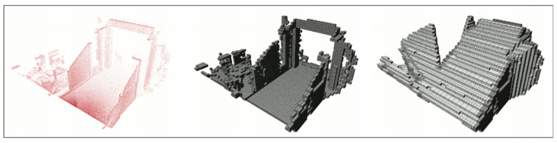


Figure 14: Left: Pointcloud data. Center: Octree structure generated from pointcloud data, showing occupied voxels. Right: Octree structure also showing free voxels.

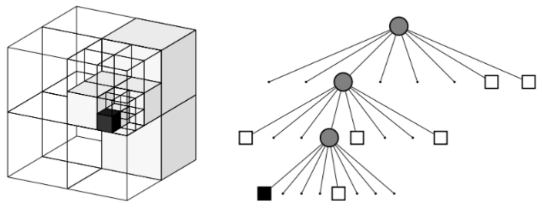


Figure 15: Left: Example of how octomap estimates occupancy using differed voxelsize. Right: representation of octree structure.

To be able to construct the octomap, the individual filtered depth images are converted into a 3D point cloud, using the camera parameters and the estimated pose to convert them into the world coordinate system. This point cloud is then integrated into the octomap. Octomap keeps track of the probability of each leaf of the node being occupied using probabilistic occupancy grid mapping. Using equation 8, the probability that a voxel is occupied is determined. In this equation, the probability that voxel n is occupied given the measurement z_t is denoted by the log probability term $P(n|z_t)$, and the result of the previous estimate is denoted as $P(n|z_{1:t-1})$. This allows octomap to take into account the probabilities and noise of the input system, in this case, the CNN.

A common assumption of a uniform prior probability $P(n) = 0.5$ is assumed. So a node is assumed to be occupied if the occupancy probability is $P(n) > 0.5$ and free otherwise.

To simplify this computation equation 8 can be rewritten using the log-odds notation shown in equation 9 and 10. This allows for a faster and less computationally intensive way to update the probability since you no longer have to do multiplications, only additions. Since the log-odds values can be converted into probabilities and vice versa, only the log values are stored in the nodes instead of the actual probability.

$$P(n|z_{1:t}) = \left[1 - \frac{1-P(n|z_t)}{P(n|z_t)} \frac{1-P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1-P(n)}\right]^{-1} \tag{8}$$

$$L(n|z1:t) = L(n|z_{1:t-1}) + L(n|z_t) \tag{9}$$

$$L(n) = log\left[\frac{P(n)}{1-P(n)}\right] \tag{10}$$

A clamping policy is used to set an upper and lower bound to the occupancy estimate, with a minimum and maximum. This results in equation 11. By using this clamping method, it limits the number of updates that are needed to change the state of a voxel. This way, a voxel that has been observed to be occupied for a long time won't need to be observed as free for a similar amount of times to be considered as a free area, allowing the octomap to adapt to change quickly.

$$L(n|z1:t) = max(min(L(n|z_{1:t-1}) + L(n|z_t), l_{max}), l_{min}) \tag{11}$$

When tuning the octomap parameters, the most important parameters are *hit/miss* and *min/max*. Where a hit is the probability $P(n|z_t)$ when a voxel is occupied, and a miss is this same probability when the voxel is registered as free. The *min/max* are used to set the limits in equation 11.

These are provided as probabilities and are converted in the log-odds space, so the probabilities are added (hit) or subtracted (miss) until they reach min or max in log-space. So increasing hit or miss probability values that are added/subtracted causes octomap to trust the sensor more, where decreasing them makes it trust the sensor less. Similarly, increasing the min and decreasing the max allows for fast updates at the cost of more noise in the depth map and visa versa.

4 Experiments & Results

In order to evaluate the system and its performance, several tests are conducted.

CNN encoder architecture and sampling density The CNN can be trained using different parameters, like the ResNet architecture and the amount of sparse samples that are used. With these experiments different training parameters are compared to find obtain the best performance.

CNN depth sampling methodology During training, different methodologies are developed to obtain sparse depth samples. In this experiment, these different sampling strategies are compared and evaluated to obtain the best performance during implementation.

ORB map point accuracy The network uses sparse depth samples obtained by ORB SLAM to improve the CNN depth estimations. In order to gain insight into the accuracy of these points several tests are performed.

Scale estimation The network uses an initialization procedure to correct for the arbitrary scale of monocular ORB SLAM. In these tests, the accuracy of this scale estimation is evaluated. Looking at how accuracy improves over the number of frames used for initialization

Tracking performance The tracking accuracy of SLAM is greatly increased by the scale estimation. In order to observe how accurate and reliable its positioning information is, several tests are conducted, comparing the designed system with comparable state of the art systems and multi camera solutions.

Depth estimation and filtering Several tests are conducted to evaluate the performance and accuracy of the monocular depth estimates. Comparing the results to the current state of the art. Additionally, in order to obtain the highest quality obstacle maps, additional filtering is applied. The performance of the system before and after filtering is also compared.

Reconstruction and integration The system should be implemented on an inexpensive UAV. In order to evaluate how the system performs on a UAV, the reconstruction of an indoor testing environment is evaluated by using a Tello UAV.

4.1 Experimental setup

In this section, the evaluation criteria are described and explained. Using a set of benchmark datasets the performance of the system is evaluated within different unique scenarios to gain a better understanding of its strengths and weaknesses. To gain insight into the performance of each dataset, a set commonly used metrics are used to evaluate the performance of the CNN. Then, these results are used to gain a better understanding of the influence this has on the performance of the SLAM system. These metrics were mainly selected to gain insight into the rate of error and different types of errors, but it was also important to be able to compare our system to other state of the art solutions. That is why it was important to use the same evaluation criteria and matching datasets that were used in similar research.

4.1.1 Evaluation dataset

In order to make sure tests can be conducted in a repeatable and comparable manner, a benchmarking dataset is required. For this purpose the TUM RGB-D benchmark dataset [65] was selected. It provides different scenarios with depth and image recordings made by a Kinect. A ground truth trajectory of the Kinect movement is also recorded using a motion capture system. The sequences contain both color and depth images in full resolution at 30Hz. Multiple types of recordings have been done, using ground robots, dynamic scenes where people are moving, and handheld static scenes. In this report, only the handheld recordings of static scenes are used. A subset of these recordings, consisting of 3 scenarios, are used for performance evaluation. Examples frames of these scenarios are shown in figure 16 to give an understanding of what these scenes look like.

The abbreviations that are used to indicate the benchmark rooms are shown in table 1.

Table 1: Benchmark scene notations

Notation	Dataset
TUM1	freiburg3_long_office_household
TUM2	freiburg2_desk
TUM3	freiburg3_structure_texture_far
TUM4	freiburg3_nostructure_texture_near_withloop

4.1.2 Depth Estimation Evaluation

To measure the performance of the network, several error metrics are used for evaluation. These evaluation metrics are shown in equation 12, 13, 14 and 15. In these equations y is a matrix of the target pixel depths in meters and $\hat{y}$ is the depth estimate in meters.

The MAE in equation 12 is used to determine the Mean Absolute Error (MAE) of each frame. This gives the most basic insight into what the magnitudes of the errors are. However, this metric can be a bit deceptive, as scenes with only objects in close proximity result in very low errors. In contrast, larger-scale scenes can have relatively small errors at large distances result in a large MAE. To gain insight into the relative error, the *absrel* metric is used, described in equation 14. This is the absolute relative error, giving an error measurement relative to the size of the ground truth. To gain insight into the spreading of the error, the root-mean-square error (RMSE) is used, described in equation 13. This determines the standard deviation of the prediction error, giving insight into how spread out the error is from the optimal solution.

The perc. depth (δ_{10}) is determined using equation 15 and is represents the percentage of pixels whose depth estimation error is less then 10% with respect to the ground truth.

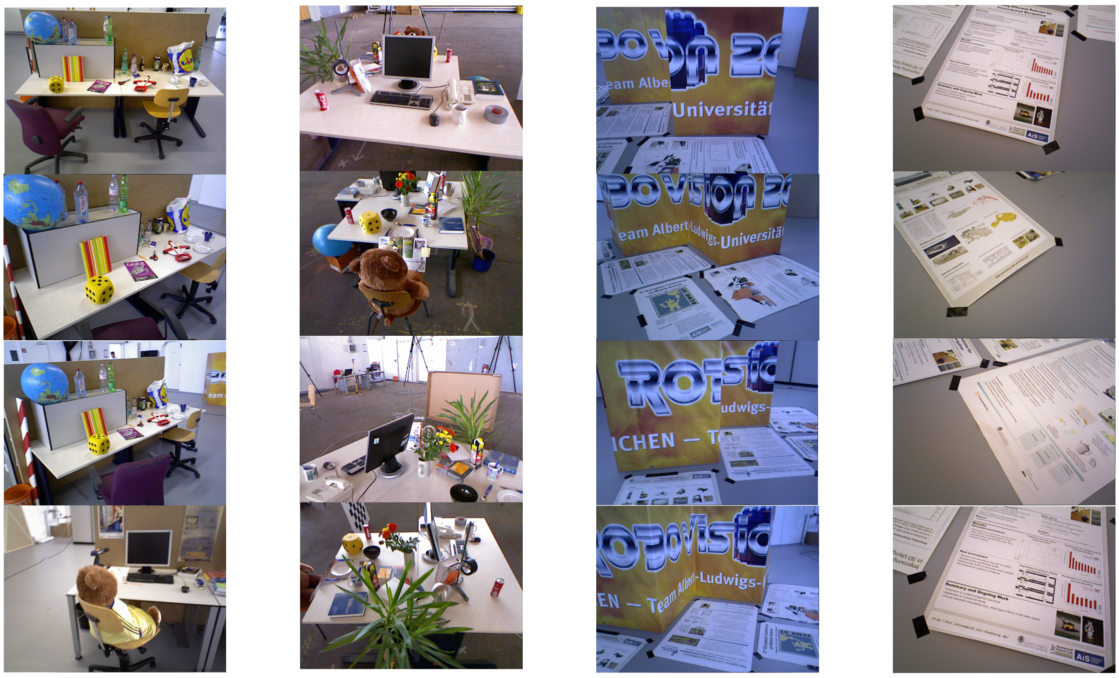

(a) TUM1 (b) TUM2 (c) TUM3 (d) TUM4

Figure 16: Four benchmark scenarios from TUM RGB-D dataset used for evaluation.

$$MAE = mean(|\hat{y} - y|) \tag{12}$$

$$RMSE = \sqrt{mean(|\hat{y} - y|^2)} \tag{13}$$

$$absrel = mean\left(\left|\frac{\hat{y} - y}{y}\right|\right) \tag{14}$$

$$\delta_{10} = \frac{max\{\frac{\hat{y}}{y}, \frac{y}{\hat{y}}\} < 1.10}{n(y)} \tag{15}$$

4.1.3 SLAM evaluation

To evaluate the performance and accuracy of the SLAM system, the approach described in [65] is taken as a guideline as it is commonly used in evaluating different SLAM systems. The research describes the datasets which are used for evaluation and also highlights different metrics used to evaluate the trajectory.

Relative Pose Error (RPE) This metric gives insight into the local accuracy of the trajectory over a fixed time interval. Therefore the RPE corresponds to the drift of the trajectory.

To determine the RPE equation 16 is used, where SE(3) represents the Special Euclidean Group in 3 dimensions, which is the group of simultaneous rotations and translations for a vector and $\triangle$ is the fixed interval between pose estimates.

For a sequence of n camera poses, the root mean squared error (RMSE) over all the time indices of the translational component is used. It should be noted that in some research, the mean error instead of the RMSE as these are less influenced by outliers. To determine the RMSE for the RPE, equation 17 is used, where $trans(E_i)$ refers to the translational components of the relative pose error E_i. As described in [65], if desired, additionally, the rotational error can also be evaluated. However, it was usually found that the comparison by translational errors was sufficient as rotational errors show up as translational errors when the camera moves. The choice for the fixed time interval between the estimates, $\triangle$ is left up to the reader. However, it is suggested for visual odometry to use $\triangle = 1 frame$, as it would directly relate to the drift per frame, making it a logical choice.

$$E_i = (P_{ref,i}^{-1} P_{ref,i+\triangle})^{-1} (P_{est,i}^{-1} P_{est,i+\triangle}) \in \mathrm{SE}(3) \tag{16}$$

$$\mathrm{RMSE}(E_{1:n}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \| trans(E_i) \|^2} \tag{17}$$

Absolute Trajectory Error (ATE) This metric uses corresponding poses which are directly compared between estimate and reference, given their pose relation. This is used to evaluate the global consistency of the trajectory. To determine the error, equation 18 is used, where R represents the rigid-body transformation corresponding to the least-squares solution that maps the estimated trajectory onto the ground truth trajectory. Similar to the relative pose error, shown in equation 19, the root mean squared error over all the time indices of the translational components are used.

$$F_i = P_{ref,i}^{-1} R P_{est,i} \tag{18}$$

$$\mathrm{RMSE}(F_{1:n}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \| trans(F_i) \|^2} \tag{19}$$

4.2 CNN encoder architecture and sampling density

When designing the depth estimating network, different design choices have to be made. Two very crucial parameters are the encoder architecture and the number of sample points. To gain insight into how these parameters influence the performance, the network is trained using two different encoder architectures, each with an increasing amount of sparse point samples.

As mentioned, there are two main implementations of the encoding layer used, ResNet18 and ResNet50. ResNet18 consists of 18 layers and ResNet50 of 50 layers, requiring more parameters, thus more GPU memory and computation time. Using the NYU depth dataset, consisting of both RGB Depth pairs, the network is trained using a different number of sparse point samples for input. The sparsifier that is used to obtain sparse depth samples from the training set is the *Uniform Random Sampler*, which takes samples with a small deviation around the set amount in an attempt to make the network more robust to variations in the number of samples. Both networks are evaluated using a different number of depth samples to determine what the improvements are on its performance. Both the ResNet18 and ResNet50 architectures are evaluated using 100, 200, 300, and 400 sparse depth samples. The results of these tests are shown in table 2.

4.2.1 Results

Looking at the results of ResNet18, it shows that including just 100 sparse samples already greatly improves the performance, reducing the RMSE by almost half. The more samples are used, the more accurate the system is. However, it can also be observed that using additional depth samples does experience some sort of diminishing return. Using 400 samples provides significantly less benefit than increasing the samples from 200 to 300, for example.

ResNet50 overall performs better then ResNet18, but the same diminishing return can also be seen here. When using 400 samples, the performance is quite similar. However, the GPU time (the amount of time it takes for the GPU to process a single image) and the amount of memory that ResNet50 requires is also considerably higher. Henceforth the ResNet18 architecture is used for the remainder of the experiments.

Table 2: Different sparse depth sample sizes compared to performance using both ResNet18 and ResNet50.

	ResNet18					**ResNet50**				
Points	MAE[m]	absrel	RMSE[m]	δ_{10}	time[s]	MAE[m]	absrel	RMSE[m]	δ_{10}	time[s]
0	0.439	0.165	0.592	0.435	0.012	0.413	0.157	0.562	0.463	0.020
100	0.191	0.072	0.309	0.793	0.007	0.165	0.058	0.279	0.852	0.020
200	0.137	0.049	0.240	0.879	0.010	0.146	0.051	0.252	0.878	0.046
300	0.134	0.047	0.231	0.887	0.011	0.137	0.047	0.232	0.899	0.045
400	0.129	0.046	0.218	0.897	0.012	0.119	0.042	0.212	0.907	0.039

4.3 CNN depth sampling methodology

The default method of training the network is done by sampling random points from the depth image. This does not accurately represent the way depth samples are obtained with ORB SLAM, as here, an edge detector is used to triangulate the 3D position of points. To account for this, the ORB sampler was implemented, as described in section 3.4.4.

To gain insight into how the sampling strategy influences the performance, the network is trained using both the default sampling strategy (UAR) and the new ORB sampling strategy. First, the training results of both networks are shown, then the integration results are evaluated to see how well the sparsifiers translate into real-world scenarios. The integration of both networks is tested on scenarios from the TUM dataset, described in section 4.1.1. The sparse depth samples are obtained using ORB SLAM running in RGB-D mode, so no scale correction is performed yet.

4.3.1 Results

Looking at table 3, it shows that the training results using UAR sampling performs significantly better than the network trained using ORB samples. This is to be expected since the uniform sampler is able to obtain diverse depth points on every point of the scene. In contrast, the ORB samples are mostly located on edges of objects and almost non-existent in textureless areas, giving it less information.

However, when looking at the integration tests, the results are reversed. It shows that uniform sampling does not perform as well anymore when the samples are no longer obtained uniformly. The network that was trained using ORB samples on the other hand, while not performing as well during training, is able to perform better than the uniformly sampled network on all of the TUM scenes based on all of the evaluation criteria.

What is observable is that there is a large difference between the performance of different scenes. Especially TUM2 and TUM1 perform worse then the other scenes. This can be accounted to the effect of them being relatively large rooms, as seen in figure 16, both scenes are recorded in a large machine hall, focusing on a desk in the center, but also containing objects in the far distance. This combination of far away, and nearby samples can cause the distance jumps, as shown in figure 12. This is also confirmed by the relatively high RMSE, compared to the MAE and absrel, indicating a larger spread in error.

Even though the uniform sampling performs better in training, when it is used in scenarios where the sparse measurements provided are no longer uniform, the performance drops significantly. The network that was trained using ORB sampling, however, does perform as well or better. Showing that the sparse structure obtained using edge detection does provide additional information about the scene that can boost depth estimation performance. These patterns are not recognized by the network when it is trained with random samples, whereas the edge detector allows the network to more clearly utilize this information and integrate into depth predictions.

4.4 ORB map point accuracy

The depth estimating CNN uses the sparse depth measurements obtained from the map points, triangulated by ORB SLAM to improve the depth predictions. Not only this, but during the scale initialization stage, these map points are compared directly to the matching depth pixels, making the map point accuracy crucial for obtaining a high-quality scale estimate.

Table 3: Training results and ORB SLAM integration results of UAR and ORB based sampling.

	Sampling Strategy							
	UAR 400				**ORB 400**			
Dataset	MAE[m]	absrel	RMSE[m]	δ_{10}	MAE[m]	absrel	RMSE[m]	δ_{10}
Training results	0.129	0.046	0.218	0.897	0.210	0.079	0.320	0.758
TUM1	0.517	0.242	0.975	0.402	0.372	0.176	0.766	0.599
TUM2	0.401	0.166	0.672	0.512	0.344	0.138	0.606	0.605
TUM3	0.297	0.099	0.752	0.721	0.188	0.081	0.392	0.777
TUM4	0.312	0.283	0.378	0.251	0.153	0.124	0.180	0.518

Therefore it is important to have an understanding of not only the quality of these map points but also on the number of points that are available in each frame and how much the number of depth samples varies. To gain insight into these metrics, several tests are conducted.

To determine the map point accuracy, ORB SLAM is tracked in monocular mode, and the depth of each map point in every keyframe is stored in combination with the matching depth measurement of the Kinect. The map point depths are then scaled using the ground truth to gain insight on the best possible performance.

The results of the map point evaluation can be seen in table 4. Where the *avg. points* is the average number of map points available over all the recorded keyframes and the *std. points* is the standard deviation of the number of available map points.

4.4.1 Results

Comparing the results of table 4 with the depth estimation accuracy from table 3, it shows that the results are comparable. The map points are more accurate than the neural network, however, not always by a large margin.

The average number of points does not give a lot of information about the quality of the map points. However, when training the network, it can provide a good estimate of the sample size that can be used during training, as well as the amount of depth noise that can be added during training.

Table 4: ORB Map point accuracy statistics.

	MAE[m]	absrel	RMSE[m]	keyframes	avg. points	std. points
TUM1	0.183	0.105	0.405	87	219	63
TUM2	0.218	0.082	0.604	99	251	54
TUM3	0.173	0.079	0.269	64	354	72
TUM4	0.091	0.067	0.117	63	320	32

4.5 Scale estimation

In order to correct for the arbitrary scale that monocular ORB SLAM is working in, the scale difference between the CNN depth estimates and the ORB map points must be determined over a sequence of frames. For these experiments, the map points of each keyframe, as well as the depth measurements from the Kinect, and the neural network are stored on disk. This is done for every scenario in section 4.1.1. The data is then used to determine the scale, evaluate the accuracy of the determined scale with respect to ground truth, and also to gain insight into the reliability and consistency of the scale estimation algorithm. During the initialization procedure, the triangulated map points are not yet used to improve the depth estimation, so no sparse depth samples are provided as input to the CNN.

The results of the scale estimates are shown in table 5, it shows the scale estimate on a per-keyframe basis. Where 3% Key Frames (KF) is the number of keyframes needed to reach 3% accuracy with respect to the ground truth scale, with a minimum of 5 frames. The ground truth scale is determined using the methodology described in [66].

4.5.1 Results

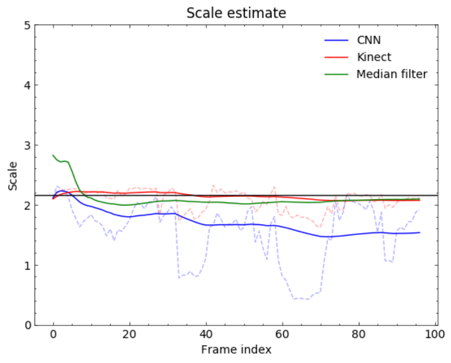
The scale estimation results for TUM2 are shown in figure 17a. This scenario was chosen as it clearly shows the limitations of the CNN depth estimation, deprived of sparse depth samples. In figure 17a it can be seen that the raw scale estimates of the CNN, shown in blue, are quite erratic. With the scale of each frame, represented as the dotted line, showing large outliers influencing the mean.

In figure 17b it can be seen why there are such large outliers. The CNN only provides depth estimates of up to roughly 3 meters, whereas the map points cover a much larger distance. To circumvent this problem, the median filtering algorithm described in section 3.4.6 is used. The result of this filtering can also be observed in figure 17b. In figure 17a the scale estimation result of the median filter is also shown, showing a more accurate result and faster convergence, with results comparable to those obtained using the Kinect.

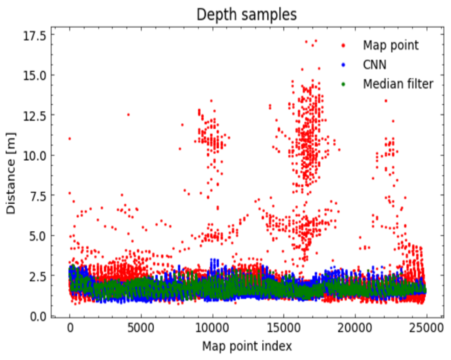
In table 5 it shows that the results of median filtering give accurate results, even outperforming the scale estimates obtained using the Kinect.

Table 5: scale estimates

	Scale estimate				
	Ground Truth	Kinect	CNN	Median filter	3% KF
TUM1	2.43	2.17	2.31	2.47	29
TUM2	2.16	2.01	1.53	2.09	11
TUM3	1.89	1.75	1.48	1.57	6
TUM4	1.29	1.21	1.35	1.31	8



(a) Different methods for scale estimation, the dashed lines represent the scale for each frame, the solid lines of the same color represent the mean of these scales. The axis line represents the ground truth scale.



(b) Scaled map point depths with depth estimates and the remaining map point depths after median filtering.

Figure 17: Scale estimation for TUM2

4.6 Tracking performance

The accuracy of the localization is not only of value for positioning of the UAV, but also for fusing the depth images. After the initialization procedure, the real-world scale is determined and used to improve the tracking. To gain insight into the trajectory tracking performance improvement that was gained by integrating the scale estimation, the trajectory errors are evaluated.

This is done using the scale obtained in table 5 and compared to the trajectory obtained using the tracking cameras provided by the datasets. The results of these tests are shown in table 6. The results are also compared to state of the art research shown in table 7.

4.6.1 Results

The results show significant improvement with respect to the unscaled trajectories, which was to be expected. As the neural network does not interact with the SLAM performance directly, the ATE and RPE are directly correlated with the scale estimation error. In figure 18, the tracked trajectory of TUM2 is shown in color, and the ground truth is shown in dark dotted lines. Here it can clearly be seen that the largest component of error is still the scale estimate. As most of the trajectory, the error is constant.

In table 7, the ATE of the system is compared to the results given in [2], the research to which their results are compared is also included. This is done for all the scenes from the TUM benchmark dataset that have been evaluated in [2], where LSD (BS) is LSD SLAM bootstrapped with the ground truth scale. It shows that the system is able to outperform or gives comparable solutions to most of the state of the art solutions. This can mainly be attributed to the accurate scale estimation procedure and the highly accurate tracking of ORB SLAM.

Table 6: Trajectory performance evaluation metrics for benchmark rooms.

	ATE scaled	ATE unscaled	RPE scaled	RPE unscaled
TUM1	0.039	1.207	0.008	0.105
TUM2	0.053	0.907	0.011	0.123
TUM3	0.303	0.829	0.074	0.196
TUM4	0.034	0.453	0.014	0.065

Table 7: Comparison in terms of ATE (Absolute Trajectory Error) [m] with state of the art.

Sequence	Ours	CNN-SLAM[2]	LSD (BS) [67]	LSD	ORB[22]	Laina[6]
TUM1	**0.0386**	0.542	1.717	1.826	1.206	0.809
TUM3	0.3033	0.214	**0.037**	0.937	0.733	0.724
TUM4	**0.0335**	0.243	0.106	0.436	0.495	1.337

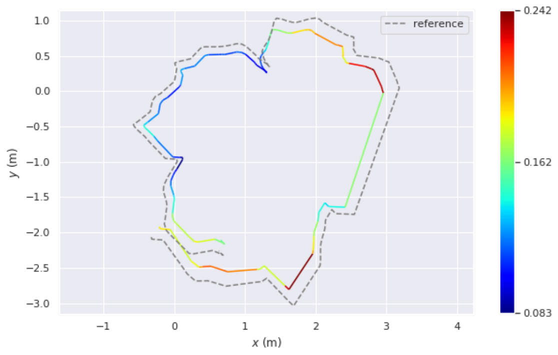


Figure 18: Estimated scaled trajectory vs ground truth, absolute trajectory error for scene TUM2.

4.7 Depth estimation and Filtering

In order to obtain high-quality OctoMaps, suitable for navigation, the input depth images are filtered as described in section 3.5. To summarise, the filters applied to the depth maps produced by the neural network:

KDtree filter This filter removes points that are too far away from their neighbors, in order to remove the low-density points that occur when large distances are bridged.

Maximum distance A maximum distance of 5 meters is used, ignoring far away map points which are more unreliable.

Convex hull A convex hull is placed around the sparse map points in order to filter out areas that don't contain sparse depth information.

These filtering tests are done in monocular mode, using the first 15 frames to obtain a scale estimate.

During the tests, the same scenario was evaluated with filtering turned off and then again with filtering turned on. The results of this are shown in table 8. In this table, the filtered column shows the percentage of pixels that have been removed. The filtering percentages also include pixels that were unable to be observed by the Kinect camera, as the IR and RGB sensors do not completely overlap. The missing Kinect measurements account for roughly 8-10% of the filtered pixels.

Additionally the depth estimation results are compared to the current state of the art, as obtained in [68].

4.7.1 Results

In table 8, it can be seen that the errors are drastically reduced. However, the amount of filtered pixels is also quite high. Filtering of the first 3 scenes is roughly between 35 to 50% (roughly 27-42% when ignoring the invalid Kinect pixels).

An outlier in this test is TUM4, where almost 60% is filtered, and the δ_{10} also decreases instead of increases. This intensive filtering can be explained by the way the KDtree filter works in for this particular scene. The KDtree filter removes points that are considered too far away from its neighbors with respect to the rest of the image. As TUM4 consists of only nearby observations of a flat surface, the mean distance between the depth pixels is very small and consistent, causing the filter to adhere to a very strict baseline. This could easily be solved by increasing the standard deviation multiplier of the filter. The reduced δ_{10} further confirms this, as it shows that a lot of correctly estimated depth pixels are filtered out due to the very strict filtering baseline.

When looking at table 9 it shows that this system, without filtering, is able to produce better then state of the art depth predictions. This can mainly be attributed to the fusion of sparse depth measurements obtained by ORB SLAM.

	Unfiltered				Filtered				
	MAE[m]	absrel	RMSE[m]	δ_{10}	MAE[m]	absrel	RMSE[m]	δ_{10}	filtered
TUM1	0.428	0.224	0.749	0.34	0.160	0.107	0.255	0.61	49.6%
TUM2	0.286	0.128	0.492	0.58	0.181	0.100	0.288	0.65	41.8%
TUM3	0.301	0.124	0.496	0.38	0.218	0.104	0.257	0.39	37.6%
TUM4	0.142	0.133	0.171	0.49	0.186	0.161	0.200	0.24	59.7%

Table 8: Comparison of performance before and after filtering

Table 9: Perc. correct depth comparison with state of the art.

sequence	Ours	DeepFusion[68]	CNN-SLAM[2]	LSD(BS)[67]	ORB[22]	Laina[6]	REMODE[48]
TUM1	**34.22**	8.069	12.477	3.797	0.031	12.982	9.548
TUM3	**37.68**	27.2	27.396	6.449	0.027	9.45	6.739
TUM4	**49.28**	14.774	24.077	3.966	0.059	15.412	12.651

4.8 Reconstruction and integration

The goal of the system is to be implemented on a low cost UAV, therefore additional tests are conducted using a Tello UAV. As there are no matching depth images collected during this test, the performance is evaluated using the map reconstruction.

As the datasets used for training the network were obtained using a Kinect, these tests also reflect how the CNN handles different input focal lengths. The first 10 keyframes are used to obtain a scale estimate. These frames are collected by observing nearby frames up to 4 meters, and avoiding far away frames, in order to obtain as many useful depth measurements.

For these tests, two rooms were selected with simple geometries, sufficient lighting and recorded using a handheld Tello. The reconstruction is done offline, separate from the obstacle map and the map is constructed using TSDF fusion [59]. Then a top down 2D map is generated from the constructed map and compared to the floor plan of the respective room.

4.8.1 Results

In figure 19 the 3D RGB reconstruction of the first room is shown, this is a simple office environment with a desk, window and a large collection of pictures on the wall. The blank areas in the center of the room are caused by gaps in observations, as this is where the Tello was located during most of the recording. In figure 20 an overlay of a cross section of the pointcloud and the floorplan of the room is shown. Giving an indication of the reconstruction quality and scale.

The Tello has quite a large focal length compared to the Kinect. This is common on a lot of consumer UAVs, as the large focal length stimulates more cautious flight behaviour due to objects seeming closer on the video feed. However, when mapping a small room this is less ideal, as the large focal length limits translational movements and forces a lot of rotational movements, which is less ideal for monocular SLAM tracking. The scale of the room shows to be relatively accurate, however, is on the larger side. From the 3D reconstruction it shows that the walls are relatively structured and flat. However, the corners are more rounded, this can be attributed to the network outputting overly smooth surfaces, and avoiding sharp edges. The window at the end of the room also shows an outward deformation, this could be due to the network looking further into the distance trough the glass.

In figure 21 the top-down floor plan comparison of the second room is shown. This recording was done in a laboratory environment. The scale of the room, again, seems relatively accurate. However, the boundaries of the room are not as straight as with the first room. This can be partially accounted to the walls containing a lot of lab equipment. On the right it shows a large outward deformation at the position of the door, as this door was not closed during the recording.

In both recordings the scale of the room is relatively accurate, though still on the larger side. This could be caused due to the difference in focal length with the training set. During the initialization procedure, due to the focal length of the Tello being larger then that of the Kinect, objects seem closer-by. This would result in a larger scale. This, however, would require more in depth research to confirm.

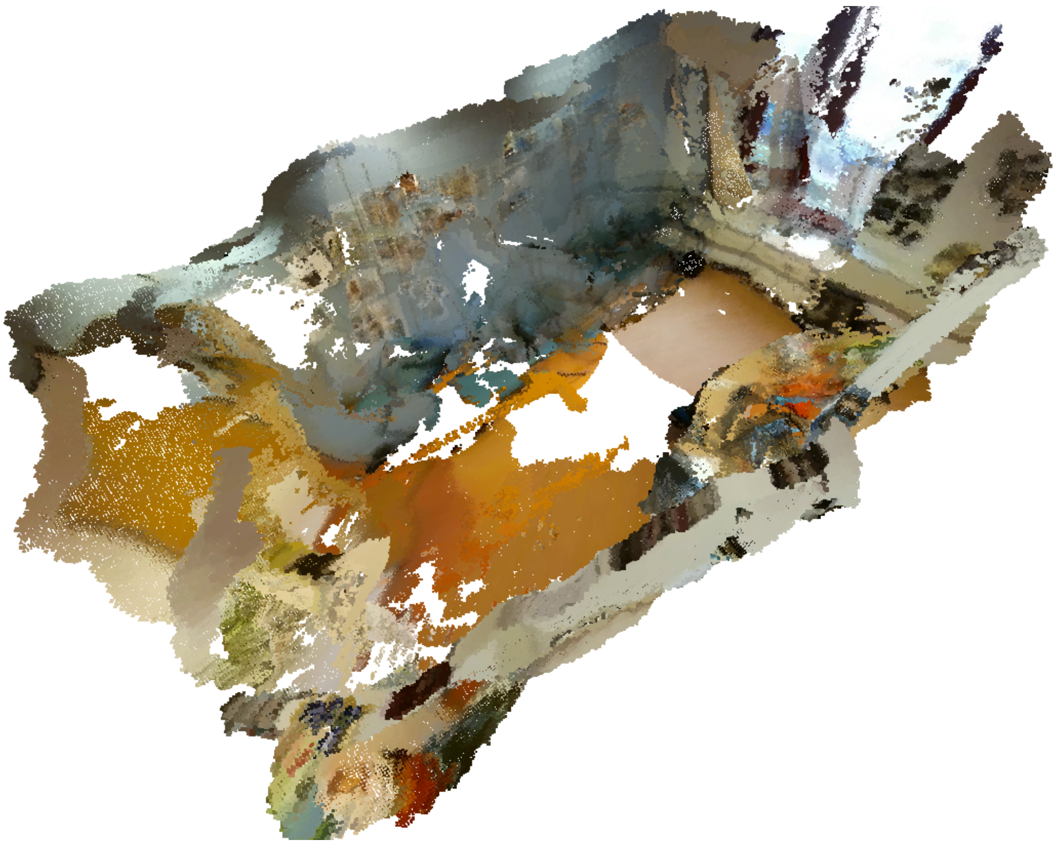


Figure 19: 3D RGB reconstruction.

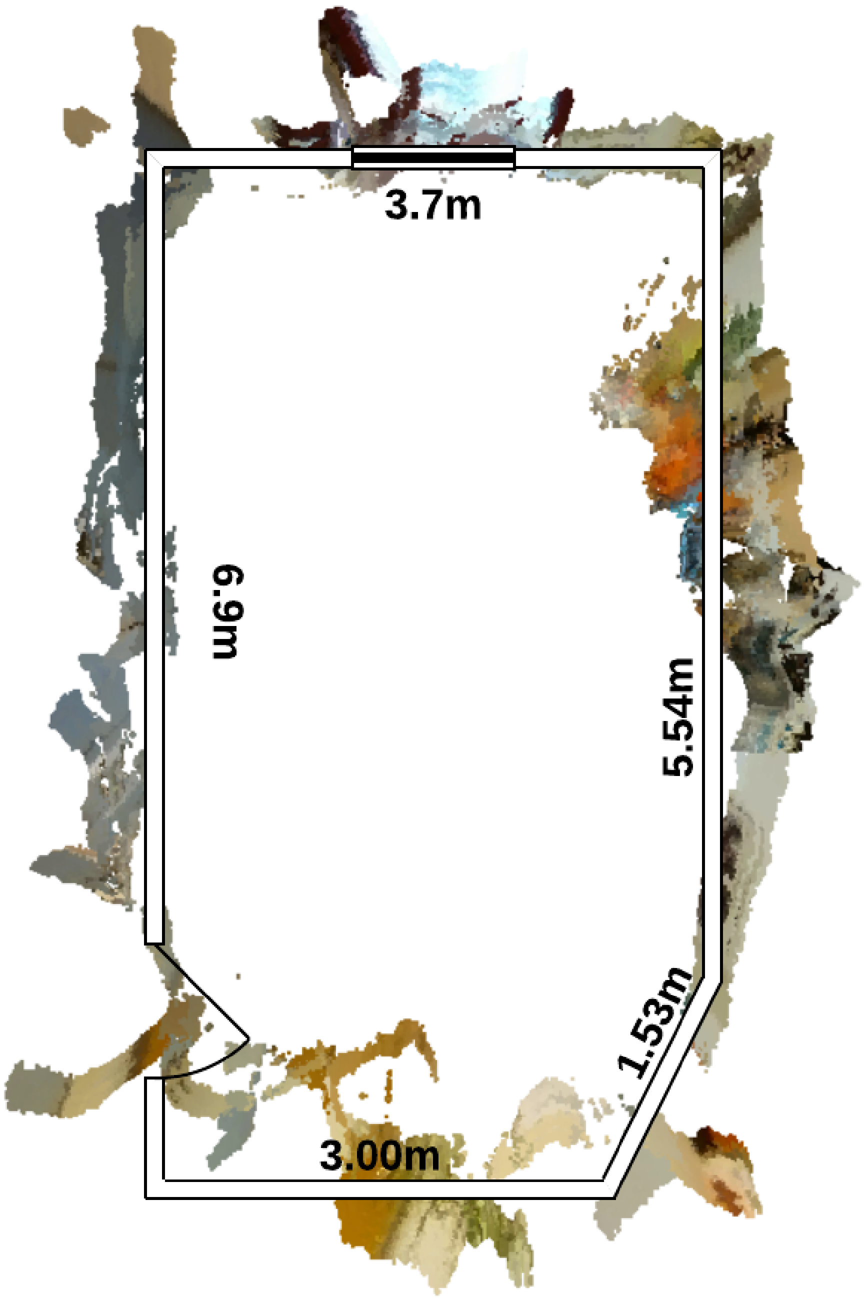


Figure 20: Floor plan overlay, office room.

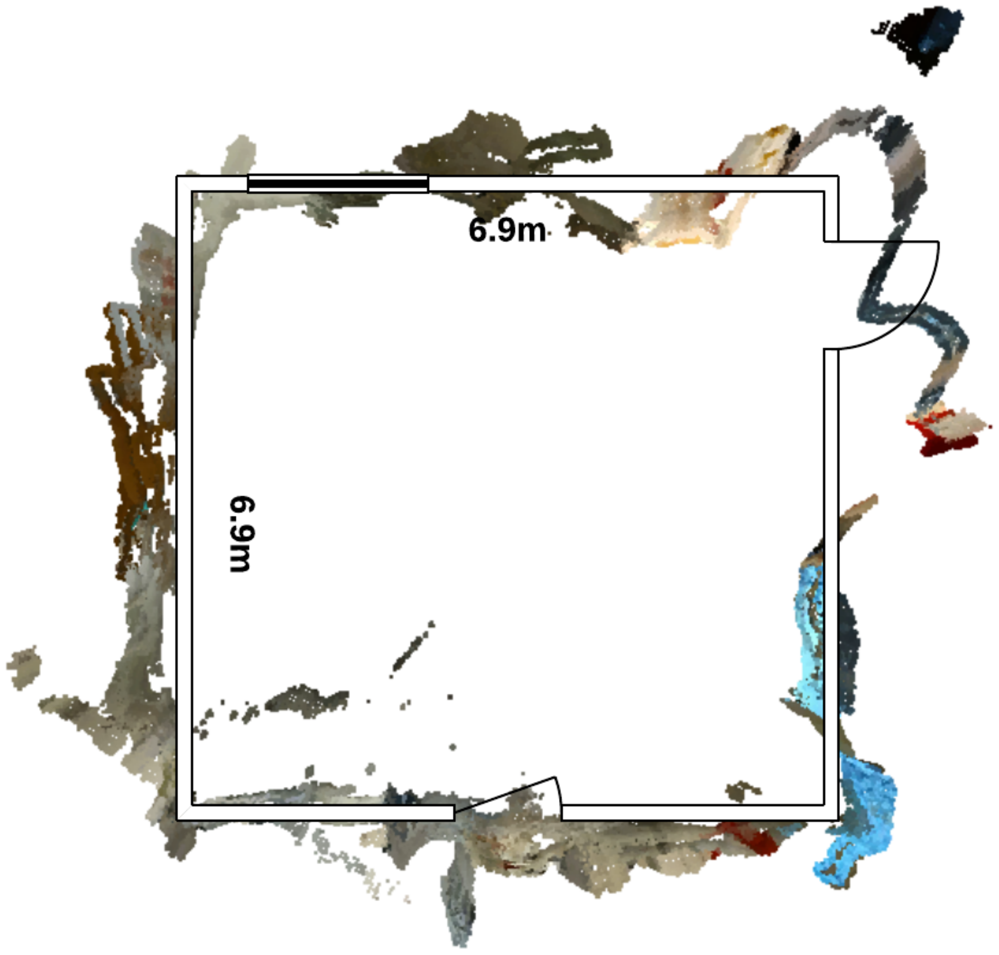


Figure 21: Floor plan overlay, laboratory room.

4.9 Discussion

The tests and experiments show good results, comparable or better then state of the art in monocular tracking and depth estimation. This is mainly due to the fusion of sparse depth estimates with the CNN based depth estimations, which other research has not yet combined in this way. However, this is not to say that this approach does not come with its own limitations. The network is very reliant on the sparse depth measurements. As seen during the initialization procedure for scale estimation, where without the sparse measurements the CNN only estimates depths up to roughly 4 meters. Showing that for further distances, the network completely relies on sparse measurements for depth prediction. Additionally noise and errors in the triangulated feature points negatively impact the depth estimations. Because of this the CNN also inherits the weaknesses of feature matching, like triangulating far away points with only translational movements. An issue that is very apparent when traversing, for example, long hallways. This is actually a case where ORB SLAM 2 improved upon its previous version, using depth cameras to make a distinction between close and far away feature points to make feature matching more reliable. Additionally, the Kinect itself can be a limiting factor during training. As the accuracy of the sensor is greatly reduced after about 5 meters.

The scale estimation of the system also shows relatively accurate results. This is mainly due to the filtering that is applied. This filtering not only helps removing outliers, but is also necessary due to the range limitations of the CNN without sparse samples. Therefore, in order to obtain a initialization scale, it helps to avoid far away scenes during the initialization. The scale estimates, using filtering from the CNN even outperform the scale estimated using the Kinect (without filtering). This however, does not indicate that the CNN scale estimate is more accurate then RGB-D SLAM, where bundle adjustment and RANSAC are also used to reduce errors and noise. Additionally, the current method still suffers from scale drift over time, as it is not being updated after the initialization procedure. This could however easily be improved upon by, for example, using intermediate frames to measure for any possible scale drift and correct for it.

The tests done with the Tello show that the system is also able to handle different camera focal lengths. In the work of CNN SLAM [2] it was suggested to correct the output of the CNN by multiplying the depth with the ratio between the focal length used for training and used for testing. This was tested in early phases, however it showed that this negatively impacts the performance. This is because the system is being corrected for the focal length difference by the sparse depth points from SLAM that are not effected by the focal length used for training. One possible way to gain additional improvements is to make sure the input images are augmented to have a similar ground sample distance (GSD) to the training set, or to train the network using different focal lengths.

5 Conclusion and recommendations

5.1 Conclusion

The goal of this research was to design a system that could be used by a UAV to generate a navigable 3D map of an indoor environment using only a monocular camera. Using only this monocular camera would keep the sensor-load, and thus the weight and size of the UAV to a minimum.

The use of conventional monocular SLAM came with several limitations. The generated maps were unsuitable for navigation and obstacle avoidance, due to them being to sparse. Additionally, monocular SLAM uses an arbitrary scale. In order to overcome these limitations, a novel system was designed that utilizes a depth estimating CNN, fused with a feature based monocular SLAM algorithm. The system is able to construct navigable maps that can be used for navigation and obstacle avoidance.

In order to obtain the scale of the scene, an initialization procedure was introduced, using the scale difference between the CNN and the sparse feature map to determine the real-world scale. The scale estimates are accurate, with estimates being correct of up to 1.48% with respect to the ground truth in some benchmark scenarios.

After the initialization procedure, a depth estimate is obtained for each keyframe. In order to obtain the best performance, the input images are paired with the sparse feature based depth estimates obtained by the SLAM algorithm. This results significantly more accurate depth estimates. By training the network to specifically use ORB feature points as input, improved performance of the CNN even more, showing that the network is able to make use of the additional information that edge detection provides about the scenes. The system still benefits from the robustness of feature based SLAM, being more robust to lighting changes and achieving better performance on low quality cameras, which is ideal for low cost UAVs. However, it does still suffer from some of the limitations of feature based SLAM. When confronted with scenes that contain little to no texture, tracking performance is reduced. This could be solved by further integrating IMUs.

In order to construct a navigable map, the individual depth estimates for each keyframe are fused into a global map using OctoMap. This has been a very effective, and efficient method for storing the maps. It does have several limitations though. The resulting maps are static, making it difficult to take loop closures into account, for example. Additionally, the maps are very susceptible to noise and outliers in the depth map inputs. Because of this, a large amount of filtering is required to obtain usable maps. By improving the way these individual depth maps are fused into a global map, a lot of benefit could still be gained.

The designed system has also been tested on a small, off the shelve, commercial UAV. Showing its ability to generate a map of a previously unknown area. This did expose the difficulty of the system to account for different focal lengths. Where the focal length of the camera used to obtain the training data is embedded into the weights of the network.

5.2 Future work

The system in its current state is able to construct obstacle maps in real-world scale. With this functionality, a basis is constructed for further work. In the following section several possible improvements are suggested that would be worth additional research when developing the system further.

Navigation and exploration

The system is able to generate navigable maps, so the next logical step would be implementing navigation and exploration components. Test have already been conducted in a simulation environment. Several solutions are available for 3D navigation and path planning, all supporting OctoMaps, however, these could still be improved to be made more reliable and suitable for use with visual SLAM. The navigation algorithm of the UAV could take into account how to best observe scenes. Currently, the path planning algorithms only consider the most efficient trajectory. However, as ORB SLAM requires texture to localize and is negatively impacted by high jerk movement, which causes camera blur, the path planning should take these requirements into account as to not result in localization losses. Another possible solution to this would be to utilize the IMU of the UAV to bridge the gap between losses of visual tracks due to illumination loss, motion blur or texture-less area traversal.

The problem of exploration however, is more complex and would require more research and development. Currently the most common way exploration is performed, is using frontiers. Where frontiers are the boundary regions between cells (or voxels in 3D maps) that are known to be free and the cells that are still unexplored. By moving from frontier to frontier, and observing these areas, an area can be incrementally explored until no frontiers are left within a set boundary. Several implementations are already available for 2D frontier exploration, however this same task in 3D becomes a lot more complex. In the work of [69], a lot of work on frontier based 3D exploration and path planning has been done, and their code is also available, however, the codebase is very much research oriented, and not ready to be directly implemented in other projects.

Visual-Inertial SLAM

As mentioned, the integration of IMU measurements would also make the system a lot more robust. In [31], an implementation of IMU measurements into ORB SLAM is described, however there are also other feature based SLAM systems like VINS-Mono [30] that provide this functionality out of the box. They also implement fusion of GPS data, for seamless transition between indoors and outdoors.

Depth map fusion

Furthermore, additional research could be done into the merging of individual depth maps. With more advanced depth fusion algorithms, the accuracy and usability of the resulting maps could be greatly improved. For example, with OctoMap, the same point observed from two different poses, with a slight error in measurement, would be considered as two different points. Further degrading the quality of the obstacle map. More recent depth fusion algorithms like Kinect fusion [59] and TSDF fusion [32] are able to fuse together multiple depth images, compensating for depth noise and trajectory errors. These mainly focus on fusing relatively high-quality RGB-D images, and less research has been done on how to handle less geometrically consistent depth estimations provided by CNNs. An example integration of these TSDF structures is presented in VoxBlox[70], which could serve as an alternative to OctoMap.

Alternatively, being able to generate a confidence map of the observed data could aid in the generation of more reliable obstacle maps. An example of such a confidence map is presented in [2]. During testing, an implementation of their confidence map generation algorithm, using only keyframes, was evaluated. However, the improvements were minimal, especially compared to the reconstruction quality when using TSDF.

Cooperative multi camera integration

Another new field of focus for improving the system would be to incorporate the use of multiple UAVs, working together to explore the same area. Being able to use multiple UAVs simultaneously for exploration would greatly reduce the time needed to map an area. There are already several implementations of ORB SLAM [71] that implement some sort of collaborative localization structure, as feature based SLAM well suited to this. The implementations for ORB SLAM are both centralized (all the work is done on a single processing machine) and distributed, where clients are able to work individually but are able to synchronize their individual maps. This would also require a different type of path planning, where frontier exploration and trajectory generation takes into account and utilizes the multiple UAVs in an efficient manner.

Loop closure and scale drift

Further research could also be done into how to handle loop closures (a previous point is re-observed and drift is corrected) and scale drift. The system currently does not correct the scale after the initialization procedure, even though the scale that ORB SLAM is using might drift and be corrected during a loop closure, which the system currently does not support.

Additionally, OctoMap is not deformable. Meaning that when a loop close occurs and the poses of the keyframes are adjusted, the complete Octomap has to be rebuild, making this a very computationally intensive and inefficient process. However, in [72], a novel method is suggested where the octomap is constructed of local sub-octomaps. Each submap being linked to a keyframe. These sub-octomaps are then fused into a full octomap. When a loop closure occurs, only a specific sub-octomap need to be restructured, making the system a lot more efficient.

Testing

Another important aspect for future improvements is to select more suitable benchmarking datasets that better reflect the targeted use cases and would allow for more representative tests. If, for example, the system would be to be deployed in different indoor datasets, the training dataset should also reflect these types of rooms. At the moment, the network is trained on the NYU indoor dataset, which consists of common indoor scenarios, living rooms, kitchens, bedrooms. However, if the UAV was to be deployed in large office environments, performance would significantly decrease. For this same reason, the system in its current state is not suitable for outdoor use.

Similarly, the datasets that are used at the moment consist mainly of 3D reconstructions and are less suitable to evaluate the performance of exploration and obstacle avoidance. For example, one issue why the testing datasets are less ideal for generating the structure of the obstacle map is that a lot of the movement is sideways. Where in real scenarios, the UAV would face the direction in which it is flying to make sure there are no obstacles. In the testing scenarios this is almost never the case. Often tracking an object in the center of the room and moving sideways around it. A great addition to the datasets would be scenarios where the UAV would be traveling from room to room, traversing doors, and maybe even windows. Such datasets would give a lot more insight into the versatility of the system and would also aid in shaping the design direction of the system. The scenarios are also handheld, making it more difficult to gain insight and tune for implementations using UAVs.

Semantic segmentation

Another topic of research that would greatly improve the applications and use fullness of the system is the fusion of semantic labels into the generated map. Being able to detect and recognize different objects in the environment would add a lot of value and possible insight to these maps. Opening up a whole new set of features and possibilities. For example, being able to distinguish between wall, floor, and ceiling could aid in automatically determining the size and shape of rooms, calculating the volume or surface, or even generate high-quality floor plans. These floor plans could even include objects like desks, beds, stairs, doors, and windows, for example. The detection of objects like doors, stairs, and windows could also help with path planning and control. For example, at the moment, clear glass windows are not detected, but by using semantic segmentation, it would be possible to restrict traversing through windows. Other useful detection possibilities, depending on the area of application, could include the detection of injured humans in disaster areas or detect damage to the building like cracks in walls or debris on the ground.

References

[1] E. Piazza, A. Romanoni, and M. Matteucci, “Real-time CPU-based large-scale 3D mesh reconstruction,” tech. rep., 2018.

[2] K. Tateno, F. Tombari, I. Laina, and N. Navab, “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction,” tech. rep., 2017.

[3] J. M. Levine, “Implementation of Monocular Visual SLAM Algorithms for Safe Autonomous Navigation of Miniature Aerial Vehicles (MAVs) in Unknown Environments,” 2016.

[4] Wikipedia, the free encyclopedia, “Image pyramid.” https://commons.wikimedia.org/wiki/File:Image_pyramid.svg, 2015. [Online; accessed April 8, 2020].

[5] F. Ma and S. Karaman, “Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image,” tech. rep.

[6] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, “Deeper Depth Prediction with Fully Convolutional Residual Networks,” tech. rep., 2016.

[7] “MarketsAndMarkets Unmanned Aerial Vehicles market analysis by Class..” http://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html. [Online; accessed April 8, 2020].

[8] S. Nuske, S. Choudhury, S. Jain, A. Chambers, L. Yoder, S. Scherer, L. Chamberlain, H. Cover, and S. Singh, “Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers,” *Journal of Field Robotics*, vol. 32, pp. 1141–1162, 12 2015.

[9] L. Yoder and S. Scherer, “Autonomous exploration for infrastructure modeling with a micro aerial vehicle,” in *Springer Tracts in Advanced Robotics*, vol. 113, pp. 427–440, Springer Verlag, 2016.

[10] Shaojie Shen, Nathan Michael, and Vijay Kumar, *Autonomous Indoor 3D exploration with a Micro-Aerial Veihcle.* 2012 IEEE International Conference on Robotics and Automation, 2012.

[11] Sungsik Huh, David Hyunchul Shim, and Jonghyuk Kim, *Integrated Navigation System using Camera and Gimbaled Laser Scanner for Indoor and Outdoor Autonomous Flight of UAVs.* IEEE, 2013.

[12] Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys, *Efficient Visual Exploration and Coverage with a Micro Aerial Vehicle in Unknown Environments.* Robotics and Automation (ICRA), 2015.

[13] G. Caroti, A. Piemonte, and I. Martínez-Espejo Zaragoza, “Indoor Photogrammetry using UAVs with protective structures: issues and precision tests,” 2018.

[14] L. Dowling, T. Poblete, I. Hook, H. Tang, Y. Tan, W. Glenn, and R. R. Unnithan, “Accurate indoor mapping using an autonomous unmanned aerial vehicle (UAV),” tech. rep., 2018.

[15] A. Bachrach, S. Prentice, and N. Roy, “RANGE -Robust Autonomous Navigation in GPS-denied Environments,” *Published in the Journal of Field Robotics*, 2011.

[16] S. Lange, N. Sünderhauf, P. Neubert, S. Drews, and P. Protzel, “Autonomous Corridor Flight of a UAV Using a Low-Cost and Light-Weight RGB-D Camera,” tech. rep., 2012.

[17] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, “End-to-End Learning of Geometry and Context for Deep Stereo Regression,” tech. rep., 2017.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J. Leonard, and Leonard}, “Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age,” *IEEE Transactions on Robotics*, 2016.

[19] J. C. Alejo Concha, “DPPTAM: Dense Piecewise Planar Tracking and Mapping from a Monocular Sequence,” 2015.

[20] J. Engel, V. Koltun, and D. Cremers, “Direct Sparse Odometry,” tech. rep., 2016.

[21] M. Risqi, U. Saputra, A. Markham, and N. Trigoni, “37 Visual SLAM and Structure from Motion in Dynamic Environments: A Survey,” *ACM Reference format*, vol. 51, 2018.

[22] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM: a Versatile and Accurate Monocular SLAM System,” 2015.

[23] D. M. Georg Klein, “Parallel Tracking and Mapping for Small AR Workspaces,” 2007.

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: an efficient alternative to SIFT or SURF,” tech. rep., 2011.

[25] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras,” 2017.

[26] R. Mur-Artal and J. D. Tardós, “Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM,” tech. rep., 2015.

[27] P. Thesis and J.-J. Engel, *Large-Scale Direct SLAM and 3D Reconstruction in Real-Time*. PhD thesis, 2016.

[28] H. Gaoussou and P. Dewei, “Evaluation of the Visual Odometry Methods for Semi-Dense Real-Time,” *Advanced Computing: An International Journal*, vol. 9, pp. 01–14, 3 2018.

[29] A. Nuchter, *3D Robotic Mapping - The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*, vol. 52. 01 2009.

[30] T. Qin, P. Li, and S. Shen, “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator,” *IEEE Transactions on Robotics*, vol. 34, pp. 1004–1020, 8 2018.

[31] R. Mur-Artal and J. D. Tardós, "Visual-Inertial Monocular SLAM with Map Reuse," 2017.

[32] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions," tech. rep., 2017.

[33] Z. Zhang, R. Zhao, E. Liu, K. Yan, and Y. Ma, "Scale Estimation and Correction of the Monocular Simultaneous Localization and Mapping (SLAM) Based on Fusion of 1D Laser Range Finder and Vision Data," 2018.

[34] Y. Li, C. Xie, H. Lu, X. Chen, J. Xiao, and H. Zhang, "Scale-aware Monocular SLAM Based on Convolutional Neural Network," pp. 51–56, Institute of Electrical and Electronics Engineers (IEEE), 8 2019.

[35] J. T. Zijlmans, "Improving Monocular SLAM using Depth Estimating CNN," 2018.

[36] W. N. Greene and N. Roy, "FLaME: Fast Lightweight Mesh Estimation using Variational Smoothing on Delaunay Graphs," tech. rep., 2017.

[37] A. Rosinol, "Densifying Sparse VIO: A mesh-based approach using structural regularities," 2018.

[38] N. Krombach, D. Droeschel, and S. Behnke, "Combining Feature-based and Direct Methods for Semi-dense Real-time Stereo Visual Odometry," tech. rep., 2017.

[39] O. Esrafilian and H. D. Taghirad, "Autonomous flight and obstacle avoidance of a quadrotor by monocular SLAM," in *4th RSI International Conference on Robotics and Mechatronics, ICRoM 2016*, pp. 240–245, Institute of Electrical and Electronics Engineers Inc., 3 2017.

[40] T. K. Larry Matthies, Richard Szeliski, "Incremental Estimation of Dense Depth Maps from Image Sequences," 1988.

[41] S. Bing Kang Richard Szeliski Jinxiang Chai, "Handling Occlusions in Dense Multi-view Stereo," tech. rep., 2001.

[42] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," tech. rep., 2006.

[43] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362–1376, 2010.

[44] R. A. Newcombe and A. J. Davison, "Live Dense Reconstruction with a Single Moving Camera," tech. rep., 2011.

[45] J. Stühmer, S. Gumhold, and D. Cremers, "Real-Time Dense Geometry from a Handheld Camera," tech. rep., 2010.

[46] A. J. D. Richard A. Newcombe, Steven J. Lovegrove, “DTAM: Dense Tracking and Mapping in Real-Time,” 2011.

[47] Z. Yang, F. Gao, and S. Shen, “Real-time monocular dense mapping on aerial robots using visual-inertial fusion,” in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4552–4559, Institute of Electrical and Electronics Engineers Inc., 7 2017.

[48] M. Pizzoli, C. Forster, and D. Scaramuzza, “REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time,” tech. rep., 2014.

[49] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan, “Cascade Residual Learning: A Two-stage Convolutional Neural Network for Stereo Matching,” tech. rep., 2017.

[50] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep Ordinal Regression Network for Monocular Depth Estimation,” tech. rep., 2018.

[51] C. Godard, O. Mac, A. Gabriel, and J. Brostow, “Unsupervised Monocular Depth Estimation with Left-Right Consistency,” tech. rep., 2016.

[52] A. Pilzer, D. Xu, M. Marian Puscas, E. Ricci, and N. Sebe, “Unsupervised Adversarial Depth Estimation using Cycled Generative Networks,” tech. rep., 2018.

[53] Y. Kuznietsov, J. Stückler, and B. Leibe, “Semi-Supervised Deep Learning for Monocular Depth Map Prediction,” tech. rep., 2017.

[54] T. Mukasa, J. Xu, and B. Stenger, “3D Scene Mesh From CNN Depth Predictions And Sparse Monocular SLAM,” tech. rep., 2017.

[55] K. Wang, S. Shen, and H. Kong, “MVDepthNet: Real-time Multiview Depth Estimation Neural Network,” tech. rep., 2017.

[56] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees,” vol. 34, pp. 189–206, 2013.

[57] L. Von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers, “From Monocular SLAM to Autonomous Drone Exploration,” 2018.

[58] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, “Autonomous aerial navigation using monocular visual-inertial fusion,” *Journal of Field Robotics*, vol. 35, pp. 23–51, 1 2018.

[59] R. A. Newcombe, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, “KinectFusion: Real-Time Dense Surface Mapping and Tracking,” tech. rep., 2016.

[60] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary robust independent elementary features,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6314 LNCS, pp. 778–792, 2010.

[61] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” tech. rep., 2015.

[62] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” tech. rep., 2015.

[63] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images,” tech. rep., 2012.

[64] K. Khoshelham and S. O. Elberink, “Accuracy and resolution of kinect depth data for indoor mapping applications,” *Sensors*, vol. 12, pp. 1437–1454, 2 2012.

[65] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A Benchmark for the Evaluation of RGB-D SLAM Systems,” tech. rep., 2012.

[66] Shinji Umeyama, “Least-Squares Estimation of Transformation Parameters Between Two Point Patterns,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, 1991.

[67] J. Engel, “Semi-Dense Visual Odometry for a Monocular Camera,” tech. rep., 2013.

[68] T. Laidlow, J. Czarnowski, and S. Leutenegger, “DeepFusion: Real-Time Dense 3D Reconstruction for Monocular SLAM using Single-View Depth and Gradient Predictions,” tech. rep., 2017.

[69] M. Faria, I. Maza, and A. Viguria, “Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS,” *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 93, pp. 113–133, 2 2019.

[70] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, “Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning,” tech. rep., 2016.

[71] S. Houben, J. Quenzel, N. Krombach, and S. Behnke, “Efficient multi-camera visual-inertial SLAM for micro aerial vehicles,” in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-November, pp. 1616–1622, Institute of Electrical and Electronics Engineers Inc., 11 2016.

[72] D. Yang, S. Bi, W. Wang, C. Yuan, W. Wang, X. Qi, and Y. Cai, “DRE-SLAM: Dynamic RGB-D Encoder SLAM for a Differential-Drive Robot,” *Remote Sensing*, vol. 11, 2 2019.