RHEINISCHE
FRIEDRICH-WILHELMS-UNIVERSITÄT
BONN

Master Thesis

Detection and Tracking of Small Objects in Sparse 3D Laser Range Data

Author:
Jan Razlaw

Advisors:
David Droeschel
Jan Quenzel

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Dirk Schulz

Autonomous Intelligent Systems
Institute for Computer Science VI

September 10, 2018

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Abstract

Detection and tracking of dynamic objects is a key feature for autonomous behavior in a continuously changing environment. With the increasing popularity and capability of unmanned aerial vehicles (UAVs) efficient algorithms have to be utilized to enable multi object tracking on limited hardware and data provided by lightweight sensors. We present a novel segmentation approach based on a combination of median filters and an efficient pipeline for detection and tracking of small objects within sparse point clouds generated by a Velodyne VLP-16 sensor. We achieve real-time performance on a single core of our UAV's hardware by exploiting the inherent structure of the data. The approach is evaluated on simulated and real scans of in- and outdoor environments, obtaining results comparable to the state of the art.

Contents

List of Figures

Chapter 1

Introduction

As robotics are getting more and more popular autonomous robots are utilized in a growing variety of environments and situations. One basis for the safe deployment of autonomous machines is a robust perception and anticipation of continuous changes in the world. This problem is addressed by detection and tracking algorithms. Detection consists of identifying or perceiving objects of interest, while tracking is the task of monitoring the objects' states over time. Knowledge about their temporal history enables to anticipate future behavior.

1.1 Motivation

Recent developments in the field of lightweight light detection and ranging (LiDAR) sensors facilitate their use on unmanned aerial vehicles (UAVs). These UAVs are utilized in an increasing number of applications, like mapping [1], inventory [2] or even health care [3]. For those, collision avoidance and dynamic path planning ensure safety and enable the efficient usage of restricted resources with regards to energy consumption and flight time. Detection and tracking of dynamic objects is a key feature to solve these tasks and to interact with the environment in general. Additionally, mapping the static part of the world can be supported by providing knowledge about dynamic objects and their state within the map.

Another beneficiary of improvements in this field is autonomous driving. The cars are usually equipped with powerful computers and a variety of different sensors. UAVs, on the contrary, are constrained by their lifting capacity — hence, providing limited computational power and allowing lightweight sensors only.

The goal of this thesis is detecting and tracking multiple objects of a limited size — such as humans — in sparse point clouds. These point clouds are generated by a lightweight LiDAR sensor mounted on a UAV. Due to the UAV's hardware limitations, efficient algorithms have to be utilized for detection and tracking to achieve real-time performance.

1.2 Contributions

Additionally to the UAV's hardware limitations the lightweight LiDAR sensor provides further challenges, like sparse data and a restricted vertical field of view. Our contribution consists of a set of efficient algorithms exploiting as much of the sparse data as possible. We present a novel approach to segment point groups of a specified width range. A detection method utilizing these segments and the inherent structure of the data. And finally, a tracking algorithm, capable of tracking a multitude of objects simultaneously. This pipeline is able to run in real-time on a single CPU core of our UAV's hardware.

Furthermore, we provide a practical application to separate the static from the tracked dynamic part of the world in the data. This is not only useful for mapping, but also enables an easier qualitative evaluation of our method.

1.3 Outline

In the following chapter, we will start with a thorough problem definition consisting of the sensor setup and a description of the resulting advantages and disadvantages. We will continue with a review of the state of the art and classify which ideas can be applied to our problem. In the fourth chapter a detailed step-by-step description of our proposed method is provided. This is followed by an extensive evaluation, finished with our conclusion and prospects for future works in the last two chapters.

Chapter 2

Basis

In the following, we first want to build a knowledge base concerning the details of the chosen sensor and point out how its advantages and disadvantages contribute to solving the task described in detail afterwards.

2.1 Sensor Setup

The chosen sensor to detect small objects in a long range is the Velodyne VLP-16, which is mounted underneath an unmanned aerial vehicle (UAV) shown in Figure 2.1. The VLP-16 is a multi-beam light detection and ranging (LiDAR) sensor that measures the distance towards obstacles in its environment.

In general, this task is accomplished by emitting a laser beam and measuring the time it takes for it to get reflected back by a surface to finally hit a detector in the sensor. Knowing the duration (Δt) of the flight, the distance d of the sensor to the surface can be calculated by

$$d = \frac{c \cdot \Delta t}{2} \tag{2.1}$$

with c as the speed of light.

The Velodyne VLP-16 has 16 laser-detector pairs placed on a vertical axis and oriented with an angle of 2° to each other resulting in a 30° vertical field of view (Figure 2.1). This setup allows to get 16 measurements at a time. Spinning the laser-detector pairs around the sensor's vertical axis generates a 360° horizontal scan of the environment (Figure 2.2).

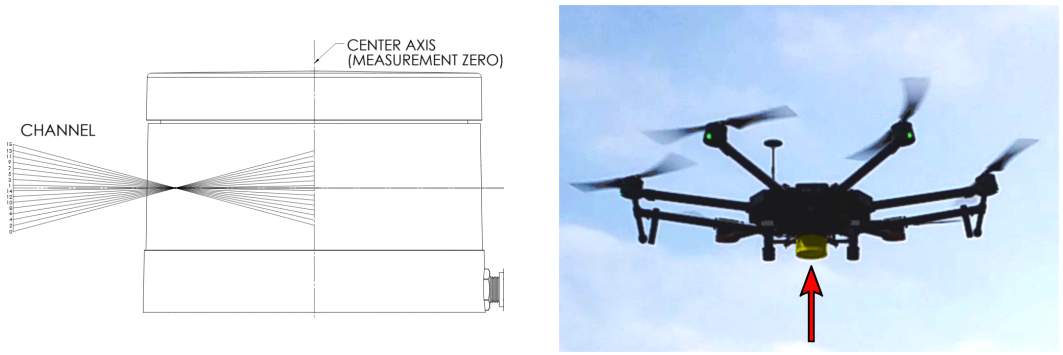

FIGURE 2.1: **Left**: A schematic drawing of the Velodyne VLP-16 visualizing the scan distribution — adapted from [4]. **Right**: The Matrice 600 with a VLP-16 mounted underneath and highlighted in yellow.

The mathematical model used to convert the distance measurements to 3D-points in the sensor coordinate frame is the same as the one used for its predecessor models given in [5], as:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d_o^i A_i \cos(\delta_i) - H_o^i B_i \\ d_o^i B_i \cos(\delta_i) + H_o^i A_i \\ d_o^i \sin(\delta_i) + V_o^i \end{pmatrix} \tag{2.2}$$

with

$$d_o^i = s^i R_i + D_o^i, \tag{2.3}$$

$$A_i = \sin(\epsilon)\cos(\beta_i) - \cos(\epsilon)\sin(\beta_i), \tag{2.4}$$

$$B_i = \cos(\epsilon)\cos(\beta_i) + \sin(\epsilon)\sin(\beta_i), \tag{2.5}$$

ϵ	as the encoder angle measurement,
s^i	as the distance scale factor,
R_i	as the raw distance measurement,
D_o^i	as the distance offset,
δ_i	as the vertical rotation correction,
β_i	as the horizontal rotation correction,
H_o^i	as the horizontal offset from the sensor frame origin and
V_o^i	as the vertical offset from the sensor frame origin

for laser i.

Additionally to the distance, the VLP-16 measures an intensity value for each point. This intensity value corresponds to the reflectivity of the measured surface.

Diffuse reflectors, like vegetation or humans, generate values from 0 to 100 for reflectivities from 0% to 100%. Retroreflectors, like road signs, are represented in the range from 101 to 255, where the latter represents an ideal reflection.

The main advantages of this sensor are the high measurement range of 100m, the low weight of about 830g and the low power consumption of 8W. Furthermore, it has a high accuracy of $\pm$ 3 cm, a good initial calibration and is robust to temperature changes [6].

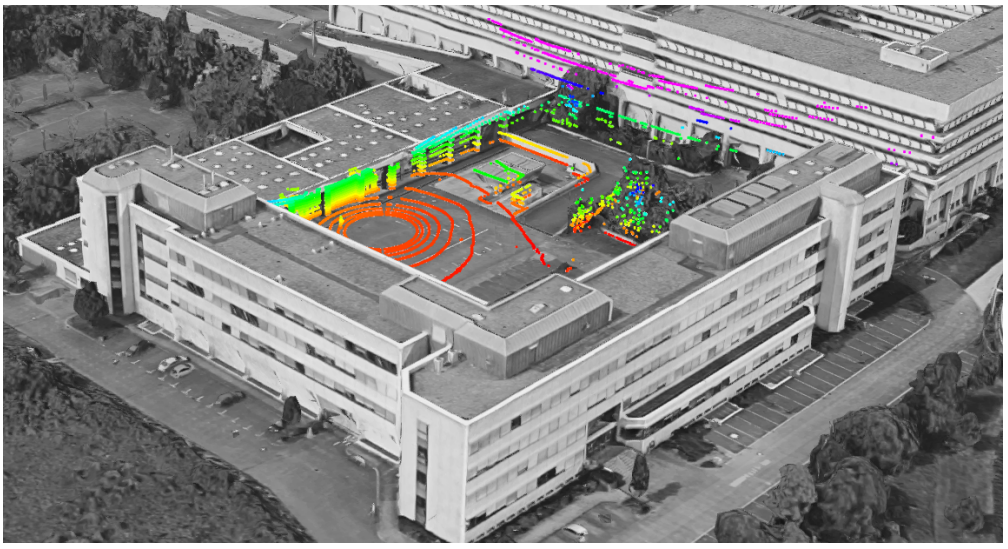

FIGURE 2.2: An exemplary point cloud generated from one scan of the courtyard of the Landesbehördenhaus in Bonn, projected into a Google satellite map [7]. Points are colored by height.

2.2 Problem Definition

Our use case consists of real-time tracking of multiple small objects in the data of a VLP-16 using the limited hardware on a UAV. Solving this kind of problem is often split up into detection and tracking. Detection in general is the task of finding target entities in the data. Tracking consists of monitoring the state of an entity over a period of time. Both of these tasks are far from trivial and highly dependent on the provided data — especially in cluttered real world scenarios.

Detection in itself is not feasible at every instance, due to occluded targets, faulty scans or objects disappearing between sparse measurements. Additionally, perceived objects may change their appearance in consequence of their orientation, posture or illumination. A detection algorithm has to account for this and provide a detection every time a target object is measured.

Tracking comes with a different set of challenges. Trackers are provided with a sequence of detections from individual time steps. Their task is to estimate the true state of an entity, even under the influence of noisy, erroneous and irregular detections. For simultaneous tracking of multiple objects, additionally the assignment problem — deciding which detection is corresponding to which tracked object — has to be solved. It is advantageous to handle cases of temporary occlusions, false detections or objects entering or leaving the sensor's field of view to prevent false assignments.

Different sensors provide differing advantages and disadvantages with regards to tracking. Cameras, for instance, scan the environment densely but usually only in a directed and limited perspective. Furthermore, the images suffer heavily from different lighting situations as can be seen in Figure 2.3. LiDAR sensors on the contrary are mostly independent from the lighting, providing a wide field of view but do not scan the environment densely.

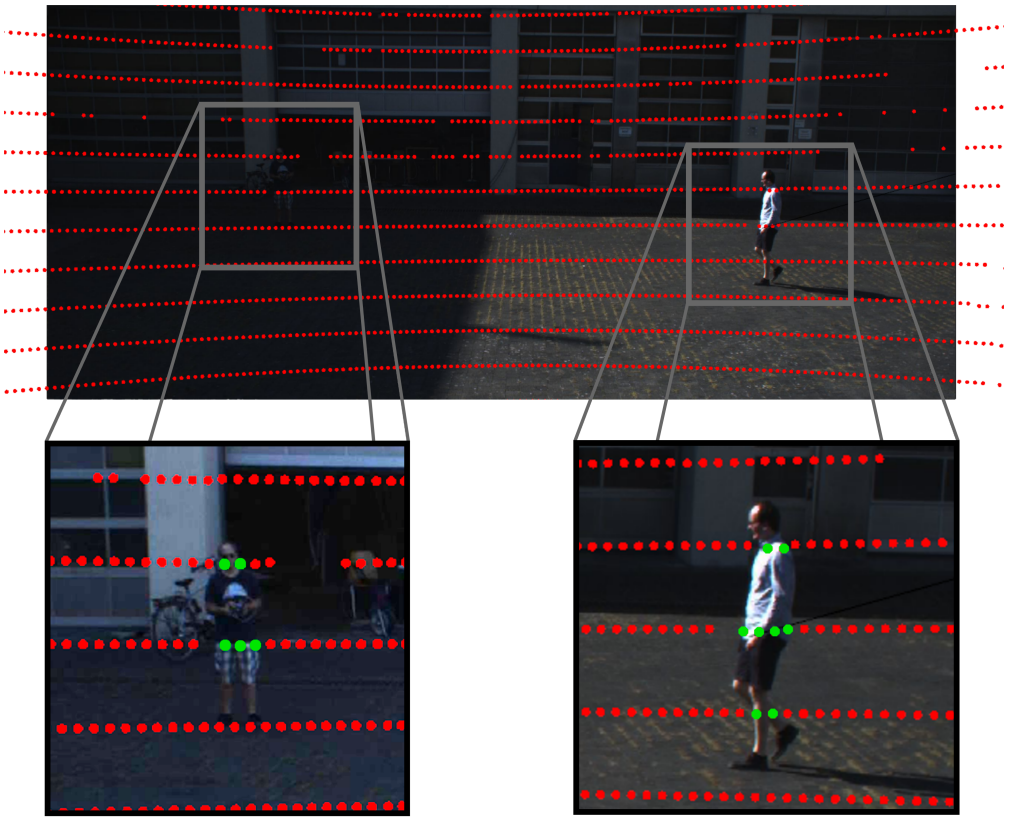

FIGURE 2.3: **Top**: A part of a point cloud generated using the Velodyne VLP-16 and projected onto a camera image recorded from the airborne Matrice 600 UAV. **Bottom**: Points corresponding to humans — with a distance to the sensor of 20m and 17m respectively — are colored in green while measurements on the background are colored in red.

We decided to use the Velodyne VLP-16 LiDAR sensor for detection and tracking, because we want to have an overview of as much of the environment as possible at every scan. As a result, we have to cope with smaller objects disappearing between scan rings or being represented by only very few measurements. Hence, our detector has to trigger every time an object is perceived by the sensor. Nevertheless, the tracker has to handle missing detections as they are an inherent problem to that kind of data at a certain distance.

Chapter 3

Related Work

The task we want to solve is the detection and tracking of small objects in sparse 3D laser range data. The following chapter defines both tasks and gives an overview of related works. Since only a few works attempt to utilize the sparse data of a Velodyne VLP-16, we also inspect approaches concentrating on denser data of similar sensors.

3.1 Overview

Tracking in itself is the task of monitoring the state of an entity. Usually one or several objects of interest are tracked in several consecutive measurements of a sensor. In our case we are especially interested in tracking the positions of dynamic objects. This task in the field of autonomous robotics has been investigated already for a long time [8], as successful tracking enables robots to interact with moving objects in their environment. Thus, it allows to use the knowledge of the object's positions for example during interactions with humans to anticipate their behavior or in traffic to avoid other participants optimizing the robot's trajectory to save time and energy.

A popular tracking paradigm is tracking-by-detection. Here, the raw measurements are preprocessed by a detector. The task of the detector is to find target objects in measurements at one instant of time — for example in one image or scan. This reduces the amount of data used for tracking drastically. The detections are passed to the tracking algorithm following them over several points in time.

There are numerous ways to design detectors. For instance, to find humans in a video, one possibility is to build a detector particularly for this one task [9]. It would search for characteristics solely or mostly present in humans — rather tall than wide, two arms and so on. The particular difficulty lies in providing a precise object description without restricting it too much in order to avoid special cases — lying people or missing arms. This subtype of tracking is called model-based tracking, as you need a precisely specified model of the target objects.

For other use cases, like mapping the environment, only static and fixed objects should be present in the map while all dynamic should be discarded [10]. Thus, it would be necessary to track all dynamic objects. It would be very difficult — if not infeasible — to specify all possible object models. One solution would be to provide a rather general specification of what a dynamic object could be and how it is distinct from the static background.

Another use case could be to track an object that was marked or just indicated by a user. Again it is hardly possible to have an exact model directly available. One would rather try to compute discriminative features in the marked area and find them again in the next measurement. Those are model-free approaches, as they do not rely on a highly detailed predefined object model.

In order to trace the dynamics of several objects simultaneously we need to restore which detection corresponds to which object at which time. This is the purpose of Multi Object Tracking (MOT). In other words it has to combine detections from different time steps to form *tracks*. This task can be subdivided in three parts. First each new detection has to be assigned to at most one already existing track representing the same object. Then false positive detections have to be filtered out. Lastly the tracker has to decide about the creation and deletion of tracks. The first case occurs when a tracked object enters the sensor's field of view and the latter accordingly after leaving it. It is helpful to maintain a motion model for each object, as it allows making predictions for the future states — even in the absence of detections — and facilitates the assignment task.

In the following, we will review related works and asses whether we can utilize them or their basic ideas. First we will concentrate on detectors working successfully on 3D point clouds. This is followed by a description of basic tracking approaches that serve as the foundation for a variety of state-of-the-art trackers. After that, a number of recent combinations of detectors and trackers is presented. The chapter is closed by tracking methods utilizing recurrent neural networks.

3.2 Detection

Our use case and setup create special demands on a detector. The VLP-16 LiDAR scanner has a full 360° horizontal field of view and maximum measurement range of 100m (Figure 3.1). However, the scan rings have a vertical angular resolution of 2° increasing the risk for smaller objects to lie between them, especially at larger distances. For this reason, we want to utilize as much of the sparse data as possible to never miss a sighting of an object. Every measurement corresponding to an object should raise a detection — even at the risk of false positives. The detector additionally has to be computationally efficient to run in real-time on the limited hardware of an unmanned aerial vehicle (UAV).

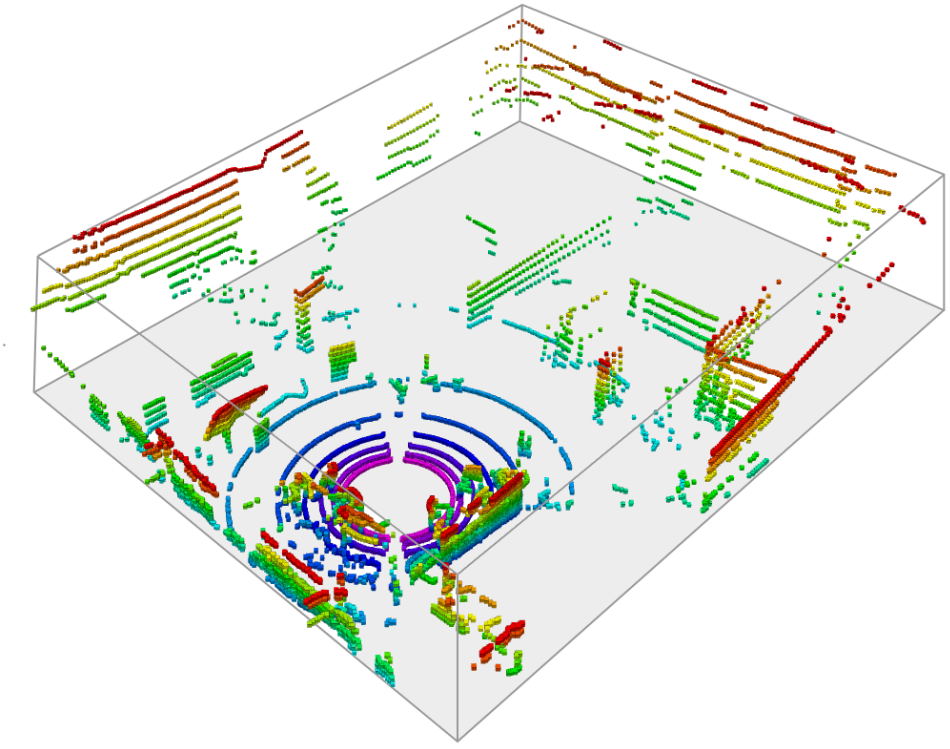

FIGURE 3.1: Exemplary scan from a Velodyne VLP-16 in a large hall of about 20m x 25m. Each of the sensor's 16 laser-detector pairs generates a scan ring. The rings are color coded. The hall is mostly visible to the sensor due to the full horizontal field of view.

Spinello et al. [11] propose a model-based approach use a Velodyne HDL-64E LiDAR scanner to detect people. The sensor scans the environment in several scan rings similar to our sensor with the difference of providing 64 scan rings with a higher vertical angular resolution. Each scan ring is segmented using Jump Distance Clustering, where a new segment is created when the distance difference between two consecutive points of a scan ring exceeds a threshold. For each

segment a set of geometrical and statistical features is computed. These features are utilized to create specialized AdaBoost classifiers [12] for different height levels. The classifiers vote into a continuous space and regions with a high vote density result in detections.

This approach has a high classification rate in a close distance of up to 10 meters. On the downside, the classification rate decreases rapidly with larger distances due to the sparsity of the data. Furthermore, it has a detection frequency of only 1Hz on a point cloud limited to a 10 meter maximum range. Hence, it is not suited for our purposes.

Maturana and Scherer [13] propose a different approach to robustly classify safe landing zones for a helicopter in vegetated terrain by detecting small and potentially obscured obstacles. First, registered scans are used to generate a volumetric occupancy map. In this map they select a 20m $\times$ 20m $\times$ 4m volume and process it by a trained convolutional neural network. This network returns the probabilities of a safe landing for each $1m^3$ subsection of the volume.

On the one hand, the detector is robust compared to approaches based on a Random Forest classifier [14] or Bag of Words [15]. On the other hand, this method cannot be used for our purposes either. By creating a map from registered scans the assumption of a static scene is made since moving objects would cause artifacts. Further, the generation of the volumetric occupancy map and the labeling of the patches takes up several seconds for the rather limited used volume.

Tuncer and Schulz [16] concentrate on providing a real-time segmentation method applicable to point clouds and robust to undersegmentation. The goal is achieved by using spatial and temporal information to discriminate close objects.

An occupancy grid is utilized to remove the ground points. Then a connected components algorithm extracts blobs in an 8 neighborhood. A motion field algorithm combined with a set of Kalman filters determines the temporal information of the scene. The preprocession is concluded by a smoothing on the dynamic grid cells to compensate association errors.

The extracted blobs are processed by a mean shift algorithm to find the number of modes in each blob's state space. Blobs containing more than one mode are considered undersegmented. Those serve as the input to the distance dependent Chinese Restaurant Process, which counteracts the undersegmentation. The algorithm is evaluated on the KITTY data set [17] providing better results than other versions of the approach.

3.3 Tracking

Just as there are special demands on the detector, there are also special requirements for the tracker. It also has to run in real-time on a UAV and be able to cope with the detector's output. Hence, it should be able to track objects robustly even if they are not visible for several measurements. Additionally, it is advantageous to filter out false positives. In the following we will present some basic approaches that became very popular and still serve as the foundation for a variety of state-of-the-art trackers.

3.3.1 Kalman Filter

Despite the fact that the Kalman Filter is less of a tracker than a state estimator, it is hardly possible to give an overview of different trackers without at least mentioning it. It is used in tracking already for decades [8, 18] and is used still [19]. In general, the Kalman filter utilizes the previous object state and a motion model to predict the current state. This prediction is corrected using the measurement at the current time step. The cycle of prediction and correction is repeated for the following measurements.

Since both the motion model and the measurements are subject to a certain amount of error or noise, these are modeled as well. For this purpose, probability distributions are utilized. In the prediction step, the Kalman filter not only estimates the next state but also the error of this state estimation.

Similarly, the error of the sensor is taken into account during the correction step. If the sensor has proven to be reliable, more weight is given to the measurement. If the measurements are erroneous, we favor the estimate of the motion model. Consequently, the new state is a weighted mean of estimate and measurement. As a final step the estimated error of the motion model is adjusted according to this weight and the next iteration is initialized. A more detailed description of the Kalman filter is provided in 4.5.2.

The original Kalman filter exhibits some limitations. For one, the error is assumed to be a zero mean Gaussian distribution which might be not expressive enough to model the real error. For another, the motion model is linear and hence has problems to cope with complex movements or abrupt changes in direction. Extensions of the Kalman filter address these shortcomings.

3.3.2 Particle Filter

A more general way to state estimation is provided by the particle filter [20]. It uses so-called particles as hypotheses of the target objects' states. These particles are distributed randomly in the entire search space, if no prior knowledge of the objects' positions is available. Each particle has a weight representing the probability of being close to an object. Detections are utilized to increase the weights of particles in their vicinity. After that, a new set of particles is created — the resampling. Higher weighted particles are sampled more often. This leads to accumulated particles in places where detections frequently occur. In the absence of detections, these accumulations are prone to persist before dissolving gradually.

If the robot or sensor is moving, it is necessary to model the particles' positions relative to it. If we want to deal with multiple dynamic objects, it is helpful to extend the filter. The first extension would be an assignment algorithm — for example the Hungarian method [21] — which would assign each particle to its corresponding object. After creating these links it is possible to use an object's motion model during the resampling.

The advantage of the particle filter is that it does not have the limitations of the simple Kalman filter. However, it needs more computational resources to maintain all particles.

3.4 Combinations of Detection and Tracking

Neither the Kalman filter nor the particle filter are providing detections or solve the assignment problem. On one hand, this assures flexibility by combining different detectors with various state estimators and assignment algorithms. On the other hand, specially tailored solutions for different sensors and object classes potentially increase development time and reduce reusability. In the following, we will present model-free and model-based methods for detection and tracking in 3D point clouds.

Moosmann and Stiller [19] propose an approach for self-localization, mapping and multi object tracking in 3D laser range data. This is a model-free approach creating range images from point clouds and segmenting them using the local convexity criterion [22] to detect object hypotheses. Tracking is split into three parts. First, the prediction step of a Kalman filter with a constant velocity model is utilized to update the state of each track. Then, the point clouds corresponding to tracked

object are registered to the current scan of the environment using Iterative Closest Points [23] to restore the movements of these objects. Finally, a classification method based on Support Vector Machines is utilized to associate and manage the tracks.

The method achieves good results, while including mapping inherently. On the downside, it needs dense data. If a small object would disappear repeatedly between the scan lines, the registration step would fail and tracking would be compromised.

Dewan et al. [10] attempt to track all dynamic objects in 3D LiDAR scans recorded from a moving autonomous vehicle. They use a model-free approach to detect groups of points that move in the same direction. This is accomplished by comparing two consecutive scans using Signature of Histograms of Orientations (SHOT) [24] descriptors. The largest group of points moving in the same direction is considered to be the background while the remaining groups are assumed to be the dynamic objects. Both are tracked utilizing a Bayesian approach.

The authors manage to surpass the results achieved by Moosmann and Stiller [19] and are able track moving vehicles even through short occlusions. One drawback though is that this approach has the assumption of rigid body motion. Hence, for example people can not be tracked robustly. With regards to our problem further drawbacks are that the objects have to move fast enough in order to be detected and need to exceed a certain size or measurement count.

Romero-González et al. [25] present a different approach for detection and tracking of people and a hand-labeled indoor data set the method is evaluated on. First, a not closer defined segmentation method is utilized for background extraction and outlier removal. This is followed by a projection of the remaining points to a ground plane and a euclidean clustering [26]. These clusters are processed using the global Ensemble of Shape Functions (ESF) descriptor [27] and classified by Random Forests into the classes *Person* and *Not person*. These Random Forests are trained on ground truth annotations.

For tracking, clusters classified as persons are matched to the closest existing hypotheses. Matches with a distance below 0.5m are considered valid and are used to update the circular velocity buffer of the assigned hypotheses. Invalid clusters generate new hypotheses while unassigned hypotheses are propagated using a weighted mean of the velocities in their buffer.

The approach is evaluated on the data set that was recorded, labeled and published especially for this purpose. It generates promising results, but lacks an evaluation of the run time. We utilize this data set to evaluate our method — a more detailed description of the data is provided in 5.2.1.1.

3.5 Learning to Track

All those methods utilize handpicked combinations of algorithms for MOT. However, they are designed for rather specific use cases and come with a variety of downsides for our task. The ability of Recurrent Neural Networks (RNNs) to tackle these problems are investigated in recent approaches. The idea is to create a network that continuously reads raw sensor data and outputs the tracks of all target objects.

Milan et al. [28], for instance, utilize a combination of an RNN and a Long Short-Term Memory (LSTM) network [29] to track multiple humans online in camera images. The RNN is responsible for three tasks. First, learning a complex dynamic model for target motion prediction even in the absence of measurements. Second, correction of the states, given target-to-measurement assignments. And third, identifying the birth and death of tracks based on the states, measurements and data associations. The LSTM network is utilized to generate the target-to-measurement assignments. For this purpose a pairwise-distance matrix between each detection and the predicted state of each target is computed and provided to the LSTM network. The output is a vector with assignment probabilities for each target. The networks are trained separately on synthetic data compiled by a simple generative trajectory model. The results are competitive against the state of the art but leave room for improvement, since the network was intended to be comparatively simple — hence general but also efficient.

Ondruska and Posner [30] investigated the capabilities of RNNs for tracking several objects in simulated 2D laser scans provided as occupancy grids. They used an Encoder-Recurrent-Decoder architecture to generate another occupancy grid representing a future state. The Encoder performs convolutional operations followed by a sigmoid nonlinearity to convert the raw input into a hidden state. The function of the Encoder is to analyze the data and serve as a detector. The Recurrent combines the Encoder's hidden state and its own previous hidden state. Thus, it serves as the tracker, combining the information from several time steps.

The objective of the Decoder is to map the hidden state of the Recurrent back to an occupancy grid.

The results of this network were motivating. So it was tested on real data which required to extend the network. For this purpose the authors experimented with different network structures [31]. They came to the conclusion that it is more advantageous to use Gated Recurrent Units (GRU) [32] rather than just increasing the size of their first network. Furthermore, they utilized several layers of GRUs and dilated convolution with different strides to enable the network to encode objects of different sizes in its memory.

Additionally, they extended their network by incorporating a Spatial Transformer module [34]. The module's purpose is shifting the hidden state with respect to the sensor's motion which the system receives from visual odometry. The results of the network were compared to one model-free approach [35]. It outperformed that approach especially in terms of occlusion handling and prediction performance while staying able to process the data in real-time. Still, the method is working on voxelized 2D data with a limited range. Hence, it is not directly applicable to tracking grounded objects using an airborne sensor.

Farazi and Behnke [36] attempt to track and identify several robots that are visually identical. Using only camera images of one robot and the heading information of all robots as the primary cue to distinguish the robots. The proposed method is split into model-based detection and tracking. Detections are generated using Histograms of Oriented Gradients (HOG) [37] features in combination with a cascade classifier. After non-maximum suppression, bounding boxes for all detections are computed and projected into egocentric world coordinates. A Support Vector Machine multi-class classifier is utilized for heading estimation. Input to this classifier are a feature vector based on a dense HOG descriptor and the normalized center position of the detection's bounding box.

For tracking, a deep LSTM network is used. The input is a vector consisting of the absolute headings broadcasted by the robots and the estimated headings, positions and probabilities of the detections. The output is an assignment of the detections to the targets or a false positive. The network was pre-trained on simulated data and then fine-tuned on a small real data set. It achieved good results compared to base line methods and was able to solve the problem of online visual tracking and identification of visually identical robots. On the downside, the number of tracked objects is limited and build into the network. Thus tracking a varying number of objects is not feasible.

Although generating promising results for their specific tasks, none of the presented methods is directly applicable to our data and problem. An additional drawback for methods relying on machine learning is the need for thoroughly labeled and versatile training data, that to our knowledge does not exist yet — details in 5.2. Thus, we propose a novel approach for multi object tracking in sparse 3D laser range data.

Chapter 4

Approach

In the following, we are going to provide a step-by-step description of the implemented Multi Object Tracking (MOT) pipeline (Figure 4.1). Starting with the point cloud that is generated from the sensor, we preprocess the data by segmenting it into foreground and background. This segmented cloud is used to create object detections that are provided to the multi object tracking algorithm. The estimated tracks are then returned to the detector to aid the detection in following measurements. We estimate the state of objects in a common world frame. For this purpose, a mapping algorithm is utilized to estimate the sensors position in the world.

The functionality is split up into ROS nodes [38]. This modular structure not only assures the option to easily replace parts of the pipeline if necessary, but also allows a convenient extension of the pipeline to fuse detections from several independent sensors.

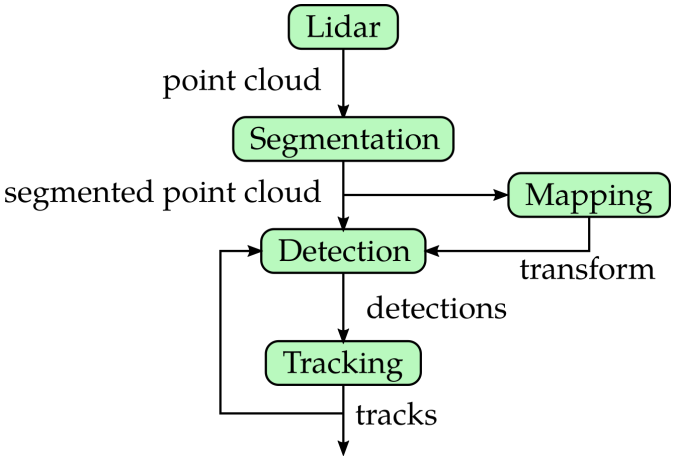

FIGURE 4.1: Overall concept of the detection and tracking method.

4.1 Point Cloud Generation

The sensor we use to generate 3D scans of the environment is the Velodyne VLP-16. A detailed description of the functionality can be found in chapter 2. In general, 16 scan rings — each consisting of about 1800 measurements — are generated. The Point Cloud Library [39] is used to store and pass the data in a unified format from node to node. Each measurement is converted into a 3D point and stored together with the intensity value and the ring number, corresponding to the ID of the laser-detector pair it was measured with. All points from one scan are stored in a point cloud, containing information about the number of measurements, their structure, time of measurement and coordinate frame.

As the segmentation method utilizes the measured distances, we adapted the sensor's driver to save those inherently existing values for each point instead of recomputing them in a later step.

Additionally, we extended the driver with an option to generate either unorganized point clouds containing only valid measurements — which is the original behavior — or organized point clouds that preserve the grid-like structure of the scan. Invalid measurements occur most frequently when the maximal measuring range of the sensor is exceeded — for example in the sky — or if the laser is directed to an absorbing surface. These measurements do not provide any valid distance or intensity readings and are filtered for unorganized clouds to save memory. Organized clouds on the contrary store the information that an invalid measurement occurred to retain the structure of a fixed number of scan rings, each containing the exact same number of measurements. This grid-like setup allows to utilize an additional set of algorithms frequently applied to structures like images.

4.2 Segmentation

The Velodyne VLP-16 generates 16 scan rings with a 360° horizontal field of view. These scan rings are deformed by objects in the environment, resulting in grouped measurements that are closer to the sensor than their neighboring measurements from the background. Our goal is to find all points belonging to foreground groups of a specified width $[sw_{min}, sw_{max}]$.

Despite the high horizontal resolution of 0.2 degrees, the sensors vertical resolution of 2° between neighboring scan rings is rather low. Thus, especially small or distant

objects raise the risk of laying in between those rings or corresponding to only very few measurements (Figure 4.3). Training of sophisticated object models under these circumstances is hard if not impossible. Hence, we segment objects according to their width, as this is the most distinct feature we can compute even for distant targets. For this, we rely on the assumption that the sensor is scanning the environment mostly horizontally. Due to the high vertical resolution, we process each scan ring individually.

Our proposed method relies on a combination of two median filters to segment objects of a specified width range. Each median filter processes a different number of neighboring measurements depending on that range. Thus, we need to convert the targeted object width from continuous meters into the discretized space of point indices. We illustrate this conversion first before going into the details of the method itself.

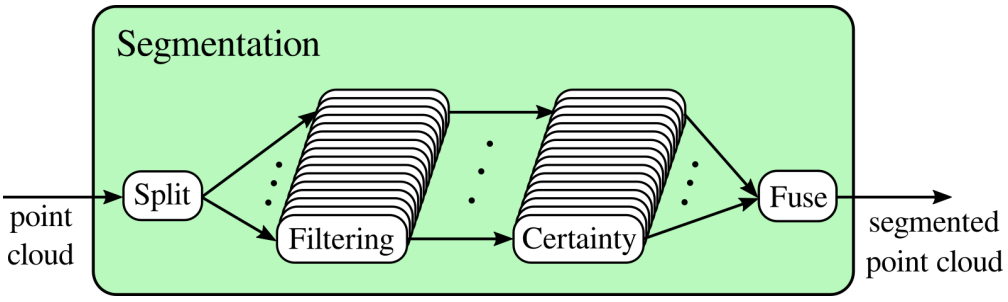

FIGURE 4.2: **Segmentation Overview**: The input cloud is split into the scan rings. These are segmented independently using a combination of two median filters and fused back to a cloud afterwards.

4.2.1 Conversion

Our goal is to segment objects of a specified width range within single scan rings. For this purpose, we need to convert a width in meters to the number of points approximately corresponding to an object of that size. This number does not only depend on the object's width w, but also on the angle α between neighboring measurements and the distance d to the object. While the width w and the angle α are constant, the distance to the object varies for different scans.

To assure computation in real-time, we apply a heuristic with simplified geometric assumptions. We neglect the orientation of the object and assume that the points

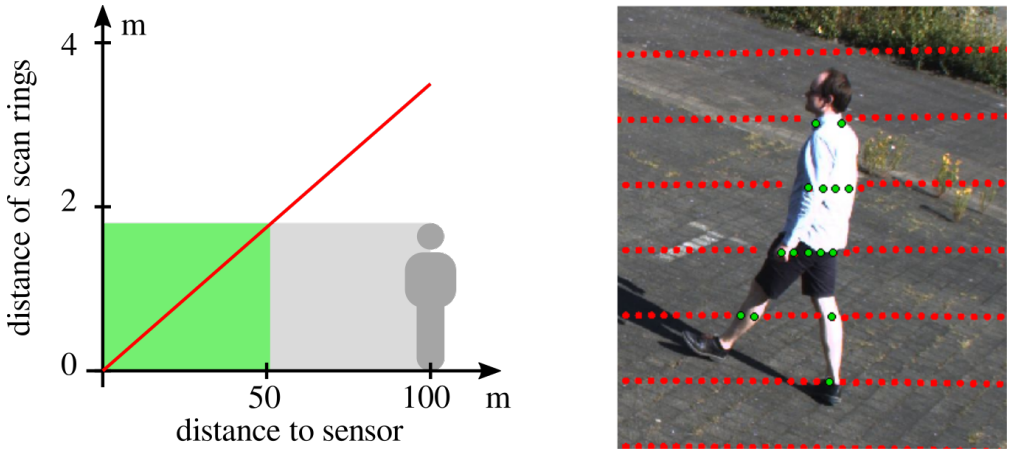

FIGURE 4.3: **Left**: Graph showing the distance of two neighboring scan rings depending on the distance to the sensor in red. A standing human of 1.8m is measured by at least one scan ring up to a distance of about 50m if scanned frontally — represented by green block. **Right**: A scan projected onto a camera image. Green points represent measurements on the human, while red points correspond to the background. Even at a distance of about 12m there are only a few measurements on the human.

form a line in the local neighborhood. Then, we compute the angle β between the center and the edge point of a hypothetical object at a distance of d by

$$\beta = \arctan\left(\frac{0.5w}{d}\right). \tag{4.1}$$

The resulting β is divided by the given angle α between the scan points (Figure 4.4b). As this corresponds to the approximate number of measurements on one half only, the double yields the approximate number of points corresponding to an object of the specified width.

4.2.2 Median Filter

In general, a median filter sorts a list of elements by a desired criterion and returns the middle element of the sorted list. In other words, the filter alternates between removing the minimum and maximum of the list until only one element is left. In the following, we will refer to the size of the input list as the kernel size k.

Our segmentation method consists of two median filters with different kernel sizes applied to the distance readings of one scan ring. The goal is to segment all point groups of a width ω within a specified width range $w_{min} < \omega \leq w_{max}$. A point group consists of neighboring measurements having a similar distance to the sensor. The *noise filter* is used with a smaller kernel size k_n to filter out noise and

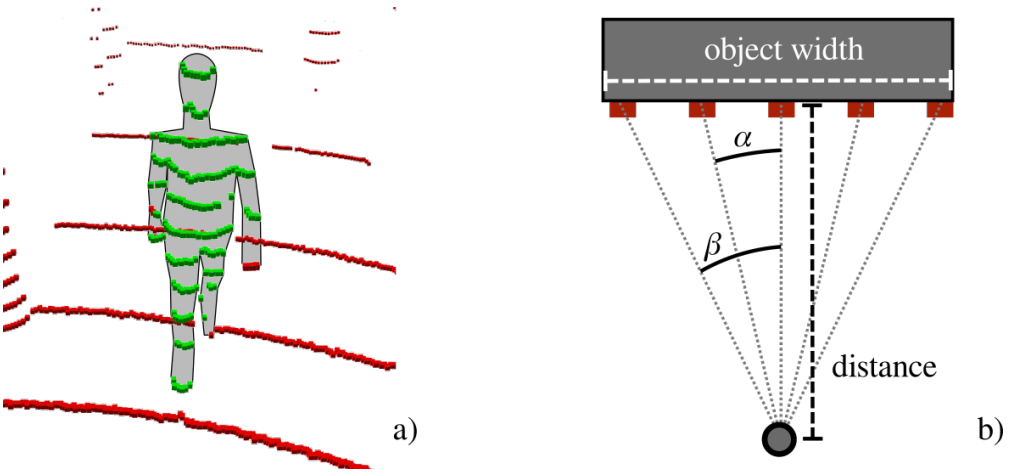

FIGURE 4.4: **a)** An exemplary segmentation of a target object. **b)** Conversion of object width to kernel size: The specified object width and the distance to the measurement are utilized to compute β. Using the fixed angle α between the measurements we approximate the number of points corresponding to an object of the specified width.

point groups with a width up to w_{min}. The *object filter* — having a kernel size of k_o with $k_o > k_n$ — is applied to the original distance measurements to additionally smooth away groups of the specified width range. Thus, the kernel sizes serve as bounds and the results only differ for point groups of the aspired width range.

To process one measurement with index i we provide a median filter with its distance d_i and the distances of the neighboring measurements with respect to the kernel size k.

$$m_i^k := median(d_{i-\lfloor \frac{k}{2} \rfloor}, \cdots, d_i, \cdots, d_{i+\lfloor \frac{k}{2} \rfloor}) \tag{4.2}$$

The difficulty here lies in finding the correct kernel sizes for each filter. If the kernel size of the noise filter is too large we might fail to segment some target objects. If it is too small the false positive rate is increased. The same applies in reverse to the object filter.

We set the kernel size k_n of the noise filter to $k_n = 2 * w_{min} + 1$. This ensures that the filter's input consists of less group points than background points, if we are processing a point that corresponds to a group we want to filter out. Thus, the filter's resulting distance will be of the background which implies that the point is filtered out (Figure 4.5a).

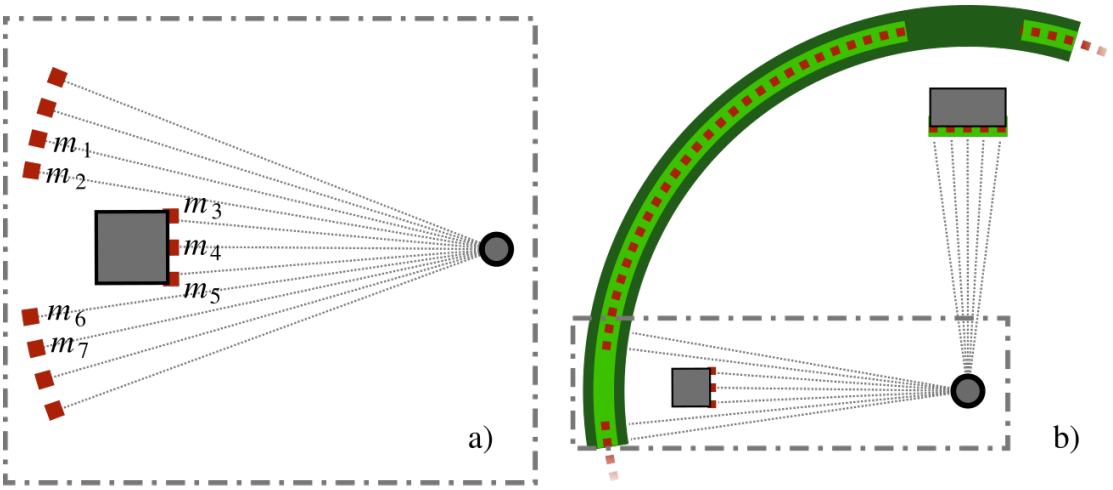

FIGURE 4.5: **a)**: Example of a median filter applied on the middle point m_4 using a noise kernel size $k_n = 2 * w_{min} + 1$ with $w_{min} = 3$. Three of the four background points m_1, m_2, m_6, m_7 cancel out with the closer object points m_3, m_4, m_5 resulting in a filtered distance that is equal to one of the background points for m_4. **b)**: Schematic example of the median filters' results on a scan ring. Measurements are depicted as red squares. The results of the object and noise filters are illustrated by the dark and bright green arcs respectively. The small object on the left is filtered out by both medians, while the target object is segmented as it is filtered out by the object filter only.

After that, the object filter is applied with kernel size $k_o = 2 * w_{max} + 1$ to the original measurements. The results are two filtered versions of one scan ring. For each point in each ring we compute the difference between those two filtered distances. They only differ in those points that correspond to objects with the desired width (Figure 4.5b). In the following we will refer to this difference of median filtered distances for a point with index i as the segmentation value Δ_i with

$$\Delta_i := m_i^{k_o} - m_i^{k_n}. \tag{4.3}$$

4.2.3 Certainty Computation

After computing the segmentation value $\Delta \in \mathbb{R}$ we need to convert it into a probability value p, with $0 \leq p \leq 1$, that specifies how certain we are of a measurement corresponding to a point group in the foreground and thus belonging to a segment. We first provide a conversion function for a general case and then present a second conversion to segment especially small objects on the ground using an elevated sensor.

4.2.3.1 General Case

We only consider positive results, since the segmentation value approximately states how many meters closer the point of a group is in comparison to the group's background. In addition, we avoid false positives by segmenting only those targets that clearly stand out. This is achieved by thresholding the segmentation value up to the fixed parameter δ_{min}. The more the point stands out from the background, the more certain we want to be. Thus, the probability is rising linearly depending on the segmentation value from δ_{min} to the second threshold δ_{low} up to a value of ρ_{max} (Figure 4.6). In the general case ρ_{max} is equal to one. The full conversion function results in

$$p(\Delta) = \begin{cases} 0 & : \text{if } \Delta < \delta_{min} \\ \frac{(\delta_{min} - \Delta) * \rho_{max}}{\delta_{min} - \delta_{low}} & : \text{if } \delta_{min} \leq \Delta \leq \delta_{low} \\ 1 & : \text{if } \delta_{low} < \Delta \end{cases} \tag{4.4}$$

with $0 \leq \delta_{min}$ and $\delta_{min} < \delta_{low}$.

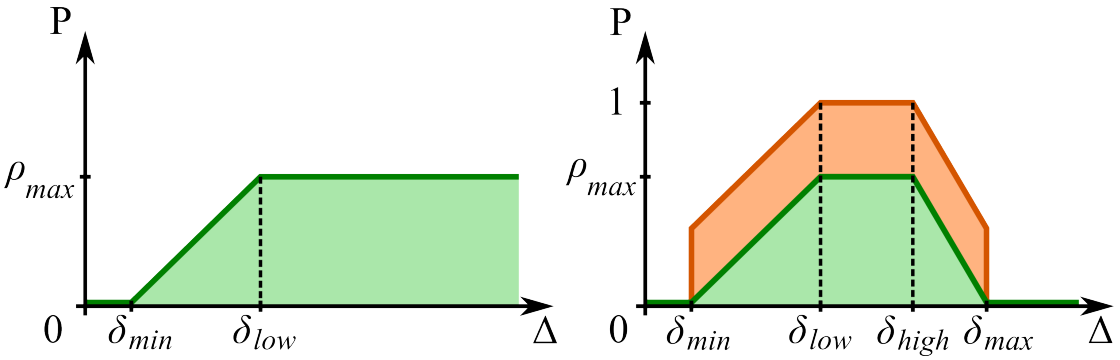

FIGURE 4.6: **Left**: General case of a function $p(\Delta)$ that converts a segmentation value Δ to a probability P defining the likelihood of a measurement corresponding to an object we want to segment (Eq. 4.4). **Right**: Special case of the same function to account for the properties of small objects measured from an elevated sensor depicted as the green graph (Eq. 4.5). The orange graph visualizes the effect of incorporating an exemplary intensity segmentation value (Eq. 4.6).

4.2.3.2 Special Case — Small Objects

The special case is an extended version of the function we have defined for the general case. It was implemented for the 2017 Mohamed Bin Zayed International

Robotics Challenge (MBZIRC), where one task consisted of retrieving small colored discs from an asphalt paved arena using UAVs. These discs have the special property of being slightly elevated but rather close to their background depending on the angle of measurement. We utilize this property to filter out segments that correspond to objects with a higher distance to their background. We do this by decreasing the probability if the segmentation value is above the threshold δ_{high} and restricting it completely if it exceeds δ_{max} (Figure 4.6). This results in the extended probability function

$$p_{MBZIRC}(\Delta) = \begin{cases} p(\Delta) & : \text{if } \Delta < \delta_{high} \\ \frac{(\delta_{max} - \Delta) * \rho_{max}}{\delta_{max} - \delta_{high}} & : \text{if } \delta_{high} \leq \Delta < \delta_{max} \\ 0 & : \text{otherwise} \end{cases} \tag{4.5}$$

with $\delta_{low} < \delta_{high}$ and $\delta_{high} < \delta_{max}$.

Another extension can be made if the sensor provides intensities for each measurement. These can be processed analogically to the distances providing another segmentation value. This segmentation value can be seen as the difference in intensities between objects and their background. Hence, it can be used as another cue to segment only those objects that have the desired width and stand out from the background not only by their position but also by their intensity value. We combine both segmentation values by summing their probabilities if the distance probability is above zero (Figure 4.6). This condition is necessary to avoid segmenting false positives on surfaces like paintings where just the intensity segmentation value generates a response. All in all the combined probability function results in

$$p_{Comb}(\Delta^{Dist}, \Delta^{Intens}) = \begin{cases} p(\Delta^{Dist}) + p(\Delta^{Intens}) & : \text{if } 0 < p(\Delta^{Dist}) \\ 0 & : \text{otherwise} \end{cases} \tag{4.6}$$

with Δ^{Dist} and Δ^{Intens} as the segmentation values using the distances and the intensities respectively and $\rho_{max}^{Dist} + \rho_{max}^{Intens} = 1$.

4.2.4 Invalid Measurements

The previous description of the segmentation method states how we treat valid measurements only. This is the case for unorganized point clouds that were filtered of non-valid measurements. When using the organized point clouds, invalid measurements are stored to keep the grid-like structure of the data intact (Section 4.1). As these measurements do not contain valid distance or intensity readings, we have to treat them differently during the segmentation.

For airborne sensors its most likely that the majority of invalid readings are a result of measurements into the direction of the sky or other objects that exceed the maximal measuring range of the sensor. In general, we do not want any of those points to be classified as a segment point. For this reason, we assign a fixed probability value of zero and replace the invalid distance by a value that exceeds the maximal measuring range. This way, these invalid points can still be used by neighboring valid measurements during the median computation and allow segmenting objects in the sky — for example other UAVs.

4.3 Mapping

For tracking objects in general, we need to know their positions in a fixed coordinate frame of the world. Thus, we need to know the pose of the sensor within this world, as all measurements are relative to the sensor's current pose. We use the Multi Resolution Surfel Mapping [40] to register the current scan to the previous to estimate the movement of the sensor between both scans. This mapping algorithm was explicitly developed to work in real-time on sparse laser range data. We declare the coordinate frame of the first scan as our world frame and consecutively register each following scan into this frame to generate a map. By doing so, we simultaneously retrieve the pose of the sensor within the world during each scan. Knowing these poses, we are able to compute the positions of the segments in the world at each time step. In the following, all positions are relative to the coordinate frame of the world.

4.4 Detection

After preprocessing the scan to find segments of the specified width, we need to group the segment points to connect those that correspond to the same object in the world. This process is called *clustering*. Depending on the structure of the data — grid-like or unorganized — one of two clustering algorithms is chosen. Afterwards, we need to filter out clusters not fitting the description of our target objects. For this, tracks from a tracking algorithm can be utilized to incorporate temporal information. In the following both steps will be described in detail.

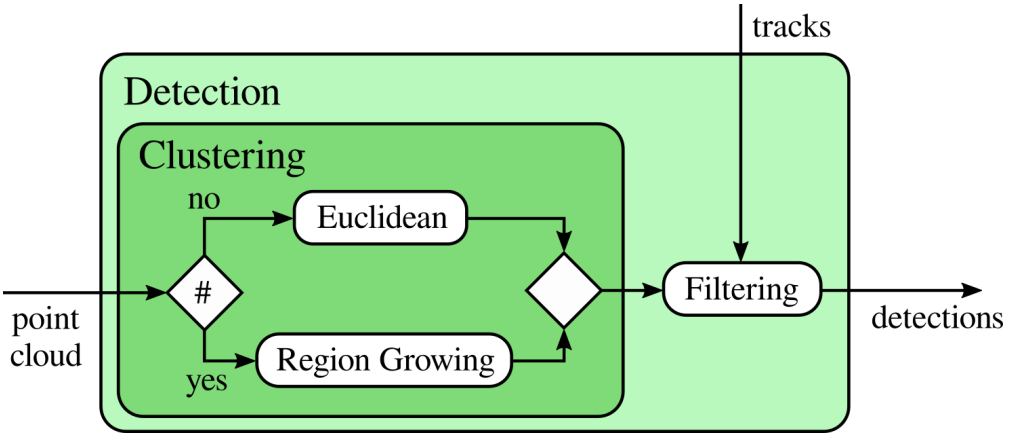

FIGURE 4.7: Detection Overview.

4.4.1 Clustering

Clustering in general consists of splitting up a set of data points such that the points in each cluster are similar to each other, while the points from different clusters are dissimilar. We will first describe the euclidean clustering [26], which is independent of the data's structure and can be applied to unorganized point clouds. Additionally, we will illustrate the region growing clustering, that exploits the internal structure of our data and is the clustering algorithm of our choice.

4.4.1.1 Euclidean Clustering

The euclidean clustering assumes points of the same cluster to be in the local vicinity of each other while being afar from points of other clusters. To generate this condition we first filter out measurements that do not belong to segments. We allow a threshold to specify how certain we want to be. All points with a

computed probability above this threshold serve as the input to the euclidean clustering (Algorithm 1). The general functionality to find all points in a cluster consists of an iterative search in a radius defined neighborhood (Figure 4.8).

Algorithm 1 Euclidean Clustering

```
1: function EuclideanClustering(Set of points P, Search radius r)
2:     C : empty list of clusters
3:     Q : empty queue of points to process
4:     c_min : minimal number of points within valid cluster
5:     for each unprocessed point p_i ∈ P do
6:         c : empty cluster
7:         add p_i to Q
8:         for each unprocessed point p_j in Q do
9:             mark p_j as processed
10:            add p_j to c
11:            find set P_i^n of unprocessed neighbors of p_j in a sphere of radius r
12:            add all points from P_i^n to Q
13:        if |c| ≥ c_min then
14:            add c to C
15:    return C
```

We would like to keep the search radius low to avoid a combination of points from different close objects to one cluster and to reduce the computational time. Still, the radius should be large enough to find measurements from the directly neighboring scan rings to successively find all points belonging to one object. Finding the best search radius is not trivial and highly dependent on the structure of the data and the scanned environment (Figure 4.8). Nevertheless, we have tested this approach and came to the conclusion that the disadvantages — high computation time for required search radius and spherical search space — outweigh the advantage of being applicable to unorganized data. Thus, we implemented a second clustering algorithm, that utilizes the organized structure of our data.

4.4.1.2 Region Growing Clustering

This second clustering method utilizes region growing on the organized structure of the point cloud. The region growing connects a seed point to its neighboring segment points and those successively to their neighboring segment points. The neighborhood is defined on the grid structure of the cloud (Figure 4.9). We classify each point with a segmentation probability above 0.8 as a segment point. Each segment point not belonging to an existing cluster induces the region growing (Algorithm 2).

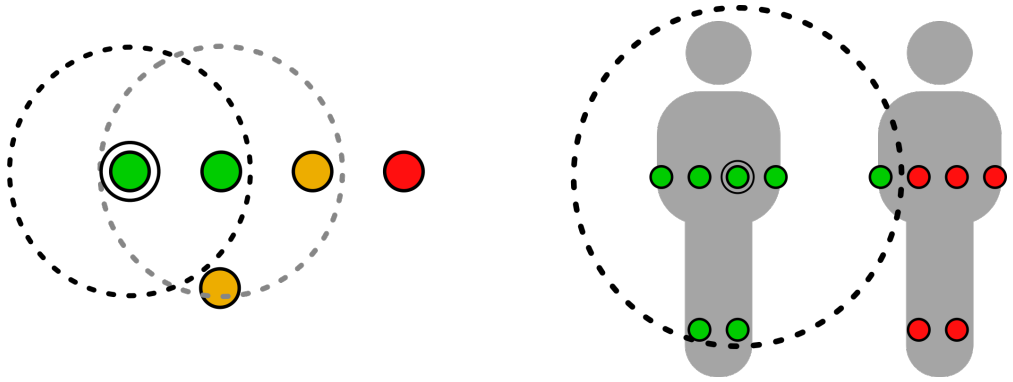

FIGURE 4.8: **Euclidean Clustering**: **Left**: Starting with the marked green point to the left, a radius search within the dotted ring is performed. All points within the radius are added to the cluster and initiate a search in their neighborhood in the next iteration. **Right**: The low vertical resolution of the sensor results in a wide search radius to find measurements from the neighboring scan rings. This raises the risk of underclustering, where several objects are falsely combined into one cluster.

Algorithm 2 Region Growing Clustering

```
1:  function RegionGrowingClustering(Organized grid of points P)
2:      C : empty list of clusters
3:      Q : empty queue of points to process
4:      c_min : minimal number of points within valid cluster
5:      for each not visited segment point p_i ∈ P do
6:          c : empty cluster
7:          add p_i to Q and c
8:          mark p_i as visited
9:          while Q not empty do
10:             dequeue p_j from Q
11:             for each direct neighbor p_n of p_j do
12:                 if p_n not visited and a segment point then
13:                     add p_n to Q and c
14:                     mark p_n as visited
15:         if |c| ≥ c_min then
16:             add c to C
17:     return C
```

We adapt this region growing clustering at two points to work more robustly on the special structure of our data. Due to the sparsity of the data, the neighborhood search radius in line 10 is extended to check more than just the direct neighbors (Figure 4.10a). Additionally, at line 11 the distance of the current point to its neighbors has to be taken into account to prevent the clustering of several distinct but partially occluding objects (Figure 4.10b). These extensions add two parameters to the clustering algorithm — a search radius r_{rg} and a distance threshold Δ_{rg}. The search radius defines the maximal Manhattan distance of the current

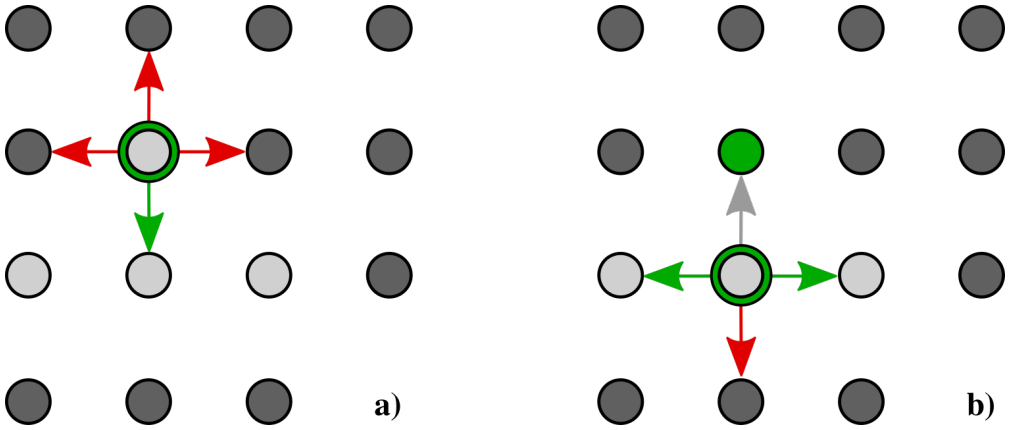

FIGURE 4.9: **Region Growing Example**: Light gray points represent segment points while dark gray points represent the remaining points not getting clustered. **a)** Initializing the cluster with the marked green point, we find all directly neighboring segment points — marked by green arrows. **b)** The next iteration adds those to the cluster and checks the neighbors of the neighboring segmented points. Points that are already in the cluster are not revisited — gray arrow.

point to a neighbor in the 2D index space of the organized point cloud. Thus, it scales well with the distance to the objects, as distant objects are represented by sparser data needing a larger euclidean search radius.

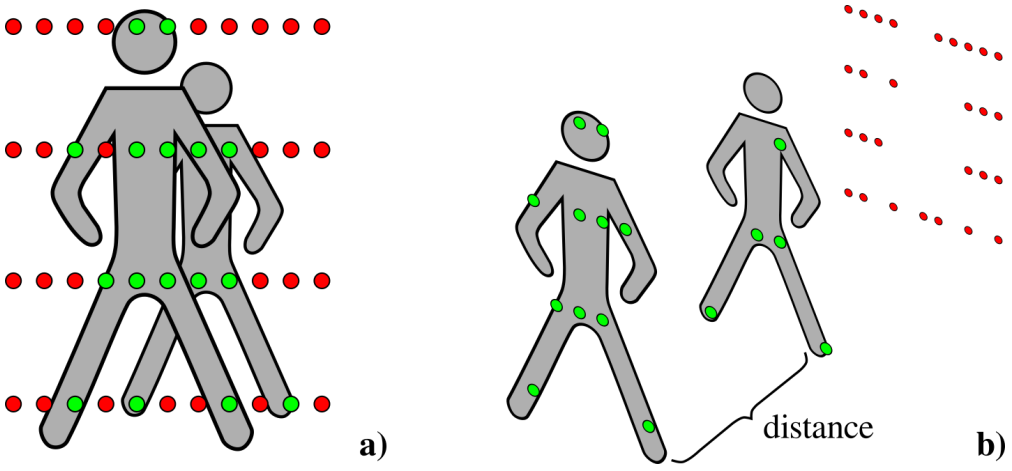

FIGURE 4.10: **Region Growing Parameters**: Exemplary part of a scan of two persons behind one another with green segment points and red background points. **a)** Sensor's Perspective: Simply connecting the direct neighbors during region growing would result in wrong clusters. **b)** Shifted Perspective: Taking the distance between neighboring points into account helps to distinguish segment points corresponding to different objects and thus prevents underclustering.

One drawback of our data for this clustering method is that the start and end of the scan might overlap. We need to handle this case explicitly by computing

the approximate overlap, finding the clusters laying within and fusing those that correspond to the same objects in the world. We estimate this overlap by computing the angle between the first and the last point of one scan ring in the sensor's coordinate system. If an overlap exists, we add 360° to the computed angle and divide by the number of points in one scan ring to get the mean angle between consecutive scan points within a ring. Dividing the overlap's angle by the mean angle results in the number of points within the overlap at each end of the scan ring. Knowing this number, we can check which clusters are within the overlap. We fuse those clusters whose bounding boxes intersect in the index space.

In the end we have decided to use the region growing instead of the euclidean clustering. The results are more precise over a wider range of distances and we are able to process the data in real-time — details in chapter 5. Furthermore, both parameters can be set intuitively and have just a minor effect on the already lower computation time, compared to the euclidean clustering.

4.4.2 Filtering

After clustering the segment points we need to make sure that the clusters' properties match our simple object model. This model consists of a height range $[h_{min}, h_{max}]$ and a maximal width w_{max}. We apply both to the axis aligned bounding box of each cluster. Due to the sensor's limited vertical field of view, objects might be scanned just partially. Consequently, we refrain from the test for a minimal height for those clusters containing at least one point from the first or last scan ring. Clusters fitting this description are considered *valid* detections.

Additionally, a second class of detections is provided. Tracked objects that increase their distance to the sensor are represented more and more sparsely in the data. This raises the risk of missing a detection of these objects. We exploit the tracker's temporal information to detect even those targets whose cluster does not fit the model. For this purpose, we utilize the objects' states provided by the tracker to loosen the thresholds for clusters in their vicinity — defined by the Euclidean distance below the threshold Λ. Detections solely generated using the tracker's knowledge are marked. As marked detections implicitly correspond to existing hypotheses, we prevent the tracker from creating new tracking hypotheses on their basis. The set of valid and marked clusters are the output of the detection method.

4.5 Multi Object Tracking

Multi Object Tracking (MOT) in general is the task of monitoring the states of several objects simultaneously. For this purpose, the algorithm maintains a set of object hypotheses. These are updated using the information from detections in the most current scan. Hence, we need to find those detections that correspond to currently maintained hypotheses. We utilize an assignment algorithm that matches detections to hypotheses with respect to a similarity measure. Each assigned hypothesis is updated using its own Kalman filter. Unassigned hypotheses might correspond to objects that left the sensors' field of view and thus must be tested for validity. Unassigned detections on the contrary might be correspondent to objects entering the field of view and thus are candidates for new hypotheses. As a final step, close hypotheses are merged. Before assigning the detections of the next time step, the current hypotheses' states are projected into this time step in the prediction. An outline of our MOT algorithm is depicted in Figure 4.11.

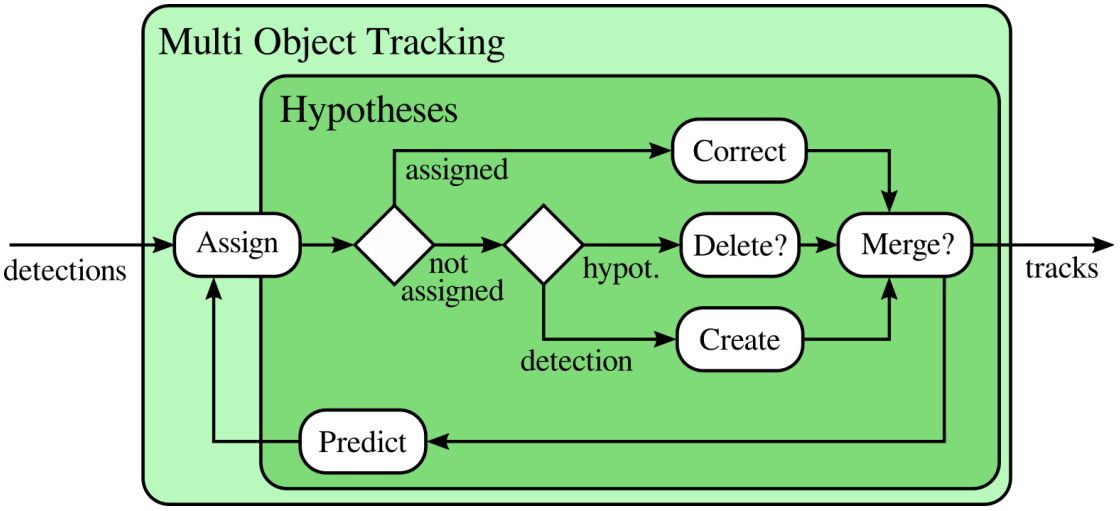

FIGURE 4.11: Multi Object Tracking Overview.

4.5.1 Object Representation

The object hypotheses are represented by an axis aligned bounding box and a state consisting of a 3D position and velocity. Additionally, the Kalman filter deployed for state estimation provides an error covariance indicating its' estimation accuracy. Detections are represented by their position and error covariance only. The latter is dependent on the noise model of the sensor. We are using the same hand tuned covariance for every detection.

4.5.2 State Estimation — Kalman Filter

The Kalman filter is a computationally efficient algorithm that uses a series of possibly noisy measurements from different time steps to estimate the state of an entity. We utilize one Kalman filter to estimate the state of one hypothesis. The underlying model assumes the state transition from the state x_{t-1} at time $t-1$ to the state x_t at time t to be of the form

$$x_t = F_t x_{t-1} + B_t u_t + w_t \tag{4.7}$$

with F_t as the state-transition model applied to the previous state x_{t-1}, B_t as the control-input model applied to the control vector u_t and w_t as the process noise, drawn from a zero mean multivariate Gaussian distribution with covariance Q_t.

Observations z_t at time t are assumed to be of the form

$$z_t = H_t x_t + v_t \tag{4.8}$$

with H_t as the observation model applied to the current state x_t under the influence of the measurement noise v_t, drawn from a zero mean multivariate Gaussian distribution with covariance R_t.

The Kalman filter alternates between the prediction of the current state and the correction of this state depending on the current measurement.

4.5.2.1 Prediction

We use the filter to estimate the state of an object hypothesis over time, correcting it using detections. Every time we detect an object in the current scan corresponding to an already existing hypothesis, we first need to project the hypothesis' state x_{t-1} into the time step t of the current scan. This is the purpose of the Kalman filter's prediction step (Figure 4.12). The state $\mathbf{x}_{t-1} = \begin{pmatrix} p_x & p_y & p_z & v_x & v_y & v_z \end{pmatrix}^T$ of a hypothesis at time $t-1$ consists of a three dimensional position p and velocity v. It is projected by

$$\hat{\mathbf{x}}_t = \mathbf{F}_{\Delta t} \mathbf{x}_{t-1} \tag{4.9}$$

to the estimated state $\hat{\mathbf{x}}_\mathbf{t}$ using a constant velocity state-transition model

$$\mathbf{F}_{\Delta t} = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.10}$$

with Δt as the time difference to the previous prediction. We omit the control-input model, since we have no control of the tracked objects.

Additionally, we make a prediction for the error covariance $\hat{\mathbf{P}}_t$ by

$$\hat{\mathbf{P}}_t = \mathbf{F}_{\Delta t}\mathbf{P}_{t-1}\mathbf{F}_{\Delta t}^T + \mathbf{Q}_{\Delta t} \tag{4.11}$$

with $\mathbf{Q}_{\Delta t}$ as a 6×6 identity matrix, element wise multiplied by Δt and a parameter q specifying the increase of the covariance per second during prediction. The error covariance is a measure for the estimated accuracy of the state estimate. The covariance increases during the prediction, representing the decreasing accuracy of the state estimate. We initialize $\mathbf{P}_0$ as a 6×6 identity matrix.

4.5.2.2 Correction

After predicting the current state and covariance, the Kalman filter corrects both using the detections' state and covariance (Figure 4.12). For this, we compute the Kalman gain $\mathbf{K}$ by

$$\mathbf{K} = \hat{\mathbf{P}}_t\mathbf{H}^T(\mathbf{H}\hat{\mathbf{P}}_t\mathbf{H}^T + \mathbf{R})^{-1} \tag{4.12}$$

with observation model $\mathbf{H}$ — in our case a 3×6 diagonal matrix with ones on the main diagonal — and measurement covariance matrix $\mathbf{R}$ representing the accuracy of the measurement. To indicate a high measurement accuracy, we set $\mathbf{R}$ to a diagonal matrix with variances of $\sigma^2 = 0.03^2$ on the main diagonal. The Kalman gain symbolizes the ratio of accuracies. It indicates how much we trust the measurements with respect to the state transition model during correction. We utilize it to weight the vector from the estimated position $\hat{\mathbf{p}}_t = \mathbf{H}\hat{\mathbf{x}}_t$ to the

measurement $\mathbf{z}_t$. This weighted vector is added to $\hat{x}_t$ computing the corrected state

$$\mathbf{x}_t = \hat{\mathbf{x}}_t + \mathbf{K}(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t). \tag{4.13}$$

If the weights in $\mathbf{K}$ are equal to zero, we distrust the measurement completely and do not correct the predicted state. If, on the contrary, all weights are one, we solely rely on the measurement during correction and neglect the predicted state.

We compute the corrected error covariance $\mathbf{P}_t$ similarly depending on the Kalman gain by

$$\mathbf{P}_t = \hat{\mathbf{P}}_t - \mathbf{K}\mathbf{H}\hat{\mathbf{P}}_t. \tag{4.14}$$

The error covariance decreases proportionally to the estimated accuracy of the measurement. This concludes one iteration of prediction and correction initialized by an assigned detection.

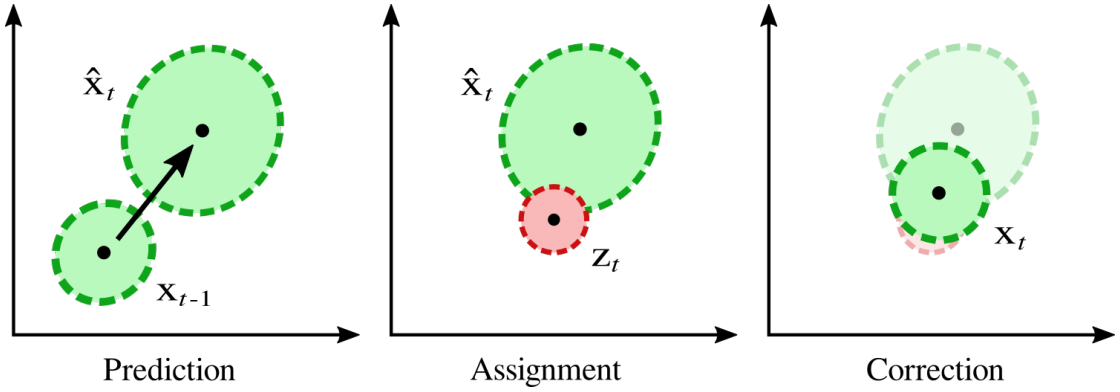

FIGURE 4.12: **Prediction**: The state $\mathbf{x}_{t-1}$ of the hypothesis — depicted as the lower green ellipsis — is projected into the next time step using the state-transition model. The accuracy of the state estimate $\hat{\mathbf{x}}_t$ decreases with every prediction, indicated by a larger ellipsis. **Assignment**: We find a detection $\mathbf{z}_t$ (red ellipsis) corresponding to the hypothesis — details in 4.5.3. **Correction**: We utilize this detection to correct the estimated state of the hypothesis. The estimated accuracy of the corrected state $\mathbf{x}_t$ is increased, due to the higher accuracy of the detection.

4.5.2.3 Velocity Limits

Achievable velocities of dynamic objects, like humans, are usually limited. False registration results in the mapping or wrong assignments can cause estimated

velocities beyond these limits. An additional cause for incorrect velocity estimates is inherent to the sparse data of the Velodyne VLP-16. Small pitch movements of the sensor can lead to significant differences in consecutive scans. A distant, static object for example measured at the bottom by a single scan ring can be measured at the top in the next scan. Although being static, its detections would have a high positional disparity inducing a high estimated velocity. To decrease this effect we restrict the estimated velocities to a maximum of $10\frac{km}{h}$. Velocities exceeding the bound are truncated.

Another effect induced by noisy data is the estimate of very small velocities for static objects. These are usually negligible, as the Kalman filter corrects slightly erroneous position estimates. In situations where another object starts to occlude the already tracked static object, this correction step is missing. In such cases, continuous predictions can cause a significant displacement of object hypotheses that correspond to static objects. We prevent this behavior by truncating velocity estimates of up to $1\frac{km}{h}$ to zero.

4.5.3 Assignment — Hungarian Method

For tracking multiple objects it is essential to know which detection corresponds to which object hypothesis. This is a classical assignment problem that we solve in polynomial time by utilizing the Hungarian method [41]. The algorithm finds a one-to-one assignment for a given cost matrix minimizing the total assignment costs. Hence, we model our problem as an $n \times n$ adjacency matrix

$$\boldsymbol{\mathcal{A}} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \tag{4.15}$$

with $a_{i,j}$ as the Bhattacharyya distance $B_D(\mathbf{d}_i, \mathbf{h}_j)$ between detection state $\mathbf{d}_i = \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ and hypothesis state $\mathbf{h}_j = \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ with position $\boldsymbol{\mu}$ and positional covariance $\boldsymbol{\Sigma}$. The Bhattacharyya distance is a measure of divergence between two probability distributions defined by

$$D_B(\mathbf{d}_i, \mathbf{h}_j) = \frac{1}{8}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \frac{1}{2} \ln \left(\frac{\det \boldsymbol{\Sigma}}{\sqrt{\det \boldsymbol{\Sigma}_i \, \det \boldsymbol{\Sigma}_j}} \right). \tag{4.16}$$

The Hungarian method finds the assignments minimizing the total distance between the detections and the hypotheses. We restrain from using the objects' appearances as recommended in [42].

After computing the adjacency matrix we provide it to the Hungarian method that performs the following steps to find an optimal assignment:

1. For each row: Subtract the smallest entry from all entries in the same row.
2. For each column: Subtract the smallest entry from all entries in the same column.
3. Use a minimal number of horizontal and vertical lines to cover all zero entries in the matrix.
4. *If* the number of lines is equal to the dimension n of the matrix, continue with step 6, *else* proceed to Step 5, as an optimal assignment of zeros is not yet possible.
5. Find the smallest entry not covered by any line. Subtract it from each uncovered row and add it to each covered column. Return to Step 3.
6. Repeat until every element assigned: Save all assignments represented by zero entries covered by the minimal number of lines, delete the corresponding lines and block the assigned elements for future assignments.

These are the steps of the Hungarian method for the simple case, where an equal number of detections and hypotheses are present and a full assignment is desired.

We enhanced the procedure by adding the possibility to forbid individual assignments between entries with a distance of $\Delta_{correspondance}$ or higher. Additionally, we need to handle the case where the number of detections and hypotheses is not equal. For this reason, we introduce a set of dummy detections and hypotheses — one dummy detection for each real hypothesis and one dummy hypothesis for each real detection. Every time a real-to-real assignment is impossible — for example if a detection is too distant to all hypotheses — a dummy is assigned. For this, we expand the possibly rectangular $m \times n$ adjacency matrix $\mathcal{A}$ for m detections and n hypotheses to a quadratic $(m+n) \times (m+n)$ matrix

$$\boldsymbol{\mathcal{A}}_{expand} = \left[\begin{array}{c|c} \boldsymbol{\mathcal{A}} & \boldsymbol{B} \\ \hline \boldsymbol{C} & \boldsymbol{D} \end{array}\right] \tag{4.17}$$

where $\boldsymbol{B}$ and $\boldsymbol{C}$ are blocks filled with the threshold Δ and $\boldsymbol{D}$ is a block of zeros. $\boldsymbol{B}$ enables assignments from the m detections to m dummy hypotheses, while $\boldsymbol{C}$ enables assignments from the n hypotheses to n dummy detections. $\boldsymbol{D}$ will contain dummy-to-dummy assignments for unused dummies. Entries in the block $\boldsymbol{\mathcal{A}}$ with a distance of Δ or higher are set to a constant value exceeding Δ. This ensures that the Hungarian method rather assigns dummy entries than making forbidden matches.

In the end, we generate an assignment where each real detection or hypothesis has a real match or is assigned to a dummy. Matched hypotheses are updated using the detections in the correction step of the Kalman filter. Detections and hypotheses assigned to dummies are treated in the following.

4.5.4 Hypotheses Generation and Deletion

We have defined how existing hypotheses are updated by assigned detections. Now we have to decide when and how new hypotheses are initiated and invalid hypotheses are deleted. Under the assumption that we track every object visible to the sensor, a new hypothesis is generated every time an object enters the sensor's Field of View (FoV). Since this object was not tracked before, its detection should be assigned to a dummy hypothesis. This triggers the creation of a new hypothesis. For this, we initialize the hypothesis' state using the detection's position and assume a velocity of zero. Additionally, a unique identification number is assigned.

Similar to objects entering the FoV, we have to handle the situation of tracked objects leaving the sensor's FoV. Here we have to differentiate between objects leaving for good and temporarily invisible objects. The latter case could happen due to a variety of reasons, like occlusions, movements at the border of the FoV or distant objects disappearing between measurements of a sparse scan. For those cases we would like to preserve the object hypothesis, project its position to the time of the current scan and recover the track once an object is visible again.

Unassigned hypotheses' covariances grow with each prediction step as their Kalman filter's accuracy of the position estimate decreases. Once a track is recovered, its covariance decreases. We exploit this behavior by deleting those hypotheses with a high covariance. For this we compute the eigenvalues of the covariance matrix and test if at least one exceeds the threshold cov_{max}.

4.5.5 Classification

We classify hypotheses into the classes *static* and *dynamic*. Every hypothesis is generated static. After each correction, we verify this state. A hypothesis is classified as dynamic once its current bounding box does not intersect with the initial bounding box at the hypothesis' generation. This way large objects have to move further to become dynamic.

Static objects being scanned from different perspectives by a dynamic sensor create the effect of moving bounding boxes. To prevent a false classification, we update the initial bounding box if it is enclosed by the current bounding box. The initial box is defined as enclosed if at least 95% of its volume are intersected by the current box. It is updated by the bounding box that encloses initial and current box.

4.5.6 Merging

Merging in multi object tracking can be applied in a number of situations. On the one hand, it is useful to model a hierarchy of objects, for example when a tracked human gets into a tracked car and becomes invisible to the sensor. On the other hand, it can be utilized to correct some of the pipeline's mistakes, like oversegmentation or falsely initiated hypotheses due to wrong assignments. In those cases, a new hypothesis is usually initialized in the vicinity of an already existing hypothesis. Hence, we merge object hypotheses if their euclidean distance to one another is below a threshold Δ_{merge}. Assuming that new hypotheses are false positives, we delete those that were initialized later.

Summing up, we have presented our pipeline to segment, detect and track multiple objects of a specified height and width in sparse, organized laser range data. We have concentrated on the usage of efficient algorithms to deploy this method in real-time on the limited hardware of an unmanned aerial vehicle.

Chapter 5

Evaluation

Now that we have described our Multi Object Tracking (MOT) approach we need to evaluate its efficacy and efficiency. The following chapter is dedicated to this purpose. We start by giving an overview of two metrics that are commonly used to evaluate MOT algorithms against ground truth annotations. We continue by presenting the data sets utilized to evaluate the pipeline and optimize its parameters. The chapter is closed by a discussion of the achieved results.

5.1 Evaluation Metrics

MOT algorithms can be used in a variety of different situations and use cases. Depending on these use cases, different behaviors are desired. For tracking people in sports, for instance, the algorithm not only has to retain its tracks but also should not mix up the identities of the players. For object avoidance however identities might not be as important as maintaining a precise motion model to predict the objects' future states.

To evaluate as many facets of our algorithm as possible we apply two different metrics each subdivided into a set of sub-metrics. Both are used by established benchmarks [17, 43] to evaluate the performance of MOT methods.

5.1.1 CLEAR MOT

The first metric we present is the CLEAR MOT metric [44]. It is based on the classification of events, activities and relationships (CLEAR) metric, but focusing on MOT algorithms.

For computation, we need a mapping M_t from objects o_t to object hypotheses h_t for every time step t. We initialize $M_0 = \{\cdot\}$ and use the procedure defined in Algorithm 3 to update the mapping sequentially for every t. The distance function $dist()$ and threshold T declared in line two and three, depend not only on the situation and use case but also on the representation of the objects and hypotheses. It is necessary to define up to which distance a ground truth object and a corresponding hypothesis can be counted as a valid match.

If both are represented by centroids, the euclidean distance would be an appropriate distance function. The authors of the CLEAR MOT metric suggest an intuitively chosen threshold of 0.5m for visual people tracking. If a bounding box representation is available, the intersection over union (IoU) could be utilized as the distance function. For three dimensional bounding boxes this is defined as the overlap's volume divided by the union's volume of two boxes. For this case, the suggested threshold would be an IoU of zero.

The mapping algorithm starts by initializing the mapping M_t for the current step t with the previous mapping M_{t-1}. Then, every previous match is checked for validity in the current step t in the loop at line six. To remain valid, matched object-hypothesis pairs must both be present at time t and have a distance below the threshold. All matches passing this test are kept for M_t.

All remaining objects and hypotheses are used to update M_t in lines 9 to 16. For this purpose, each remaining object has to be assigned to at most one hypothesis, while each hypothesis can be assigned to at most one object. Additionally, the assignment has to be of a form that the overall object-hypothesis distance is minimal. The stated assignment algorithm is the Munkres' algorithm [41], also known as the Hungarian method described in 4.5.3. Each assignment is tested for validity with respect to the distance. Whenever a valid assignment contradicts a match from a previous step, we replace the previous match in M_t. In other words, we match an object that corresponded to a specific hypothesis to another hypothesis or vice versa. We count this identity switch as a mismatch error. Assignments that do not contradict a previous match are added to M_t. The result is an updated valid mapping M_t from objects to hypotheses for time t.

Algorithm 3 CLEAR MOT Mapping

function ClearMotMapping(Objects o_t, Object hypotheses h_t at time t)
    $dist()$: distance function
    T : distance threshold
    M_{t-1} : mapping of objects to hypotheses at time $t-1$
    M_t : initialize with M_{t-1}
    for each mapping $(o_{t-1,i}, h_{t-1,j}) \in M_{t-1}$ **do**
        if $o_{t-1,i}$ and $h_{t-1,j}$ are present at time $t \wedge dist(o_{t,i}, h_{t,j}) < T$ **then**
            keep $(o_{t,i}, h_{t,j})$ for M_t
    for all $o_{t,i}$ and $h_{t,j} \notin M_t$ **do**
        assign using Hungarian method
        for each assignment $(o_{t,i}, h_{t,j})$ **do**
            if $dist(o_{t,i}, h_{t,j}) < T$ **then**
                if $(o_{t,i}, h_{t,j})$ contradicts $(o_{t-1,i}, h_{t-1,k})$ **then**
                    replace $(o_{t-1,i}, h_{t-1,k})$ in M_t
                else
                    add $(o_{t,i}, h_{t,j})$ to M_t
    return M_t

Based on these mappings for each time t, we are able to compute the evaluation metrics. We sum up the distances between all valid matches at a time t as d_t to calculate the Multi Object Tracking Precision (MOTP) by

$$MOTP = \frac{\sum_t d_t}{\sum_t c_t} \tag{5.1}$$

with c_t as the number of valid matches at time t. The MOTP denotes the average distance between valid object-hypothesis pairs. This metric is independent from the tracker's ability to provide consistent trajectories or distinguish valid from non-valid detections.

For this purpose, we compute the ratios of misses $\overline{m}$, false positives $\overline{fp}$ and mismatches $\overline{mme}$ as

$$\overline{m} = \frac{\sum_t m_t}{\sum_t g_t}, \tag{5.2}$$

$$\overline{fp} = \frac{\sum_t fp_t}{\sum_t g_t}, \tag{5.3}$$

$$\overline{mme} = \frac{\sum_t mme_t}{\sum_t g_t} \tag{5.4}$$

with m_t as the number of unmatched objects, fp_t as the number of unmatched hypotheses and g_t as the number of objects present at time t (Figure 5.1). The number of mismatch errors mme_t is determined in line 13 of Algorithm 3, as the total number of contradictions.

The sum of these ratios is the total error rate E_{tot}, while $1 - E_{tot}$ results in the Multi Object Tracking Accuracy (MOTA), equivalently defined as

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}. \tag{5.5}$$

Contrary to the MOTP, the MOTA is a measure for the consistency of the generated tracks. It accounts for identity switches, missed and falsely tracked objects.

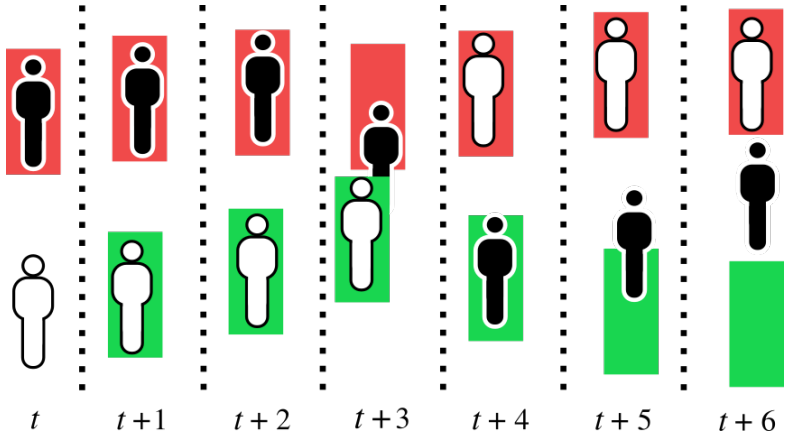

FIGURE 5.1: **CLEAR MOT Example**: Objects are depicted as manikins — hypotheses as boxes. The white object corresponds to the green hypothesis, while the black corresponds to the red at first. t : White object is not tracked, increasing the misses count by one. $t + 4$: Correspondences switch, resulting in 2 mismatches. $t + 6$: Green hypothesis looses track of the black object, resulting in another miss and a false positive.

5.1.2 Coverage

Another way to evaluate the performance of an MOT algorithm is to inspect how much of the objects tracks were covered by the hypotheses [45]. This metric is split up into three ratios — mostly tracked (MT), partially tracked (PT) and mostly lost (ML). An object's trajectory is mostly tracked if at least 80% of it is covered by hypotheses. It is mostly lost if less than 20% is covered and partially tracked for the remaining cases. We apply the same constraints as for the CLEAR MOT for an object to be classified as tracked.

This metric does not account for identity switches, false positives or precision. It can be seen as an addition to the CLEAR MOT metric, providing a more detailed insight to the ratio of misses.

5.2 Datasets

In the following, we are going to give an overview of the data sets we utilized for evaluation and parameter optimization. The first two data sets are used for a quantitative evaluation against ground truth data. The third consists of scans recorded during two flights of our UAV in a large courtyard. Due to missing ground truth data for the latter data set, we developed a method for a qualitative evaluation using unlabeled data.

5.2.1 Quantitative Evaluation

The following two data sets will be used for a quantitative evaluation and parameter optimization using the previously described metrics. For this purpose ground truth labels are required. These could be given as bounding boxes or three dimensional coordinates representing the objects' centers. As we did not manage to find a data set where all objects of a specified width and height are labeled, we tune the pipeline's simple object model to track humans and evaluate against the given labels.

5.2.1.1 InLiDa

The Indoor LiDAR Dataset (InLiDa) [25] consists of six hand labeled sequences captured using a Velodyne VLP-16 in an indoor environment. The sequences have a total duration of 501 seconds and contain 4823 scans. The sensor is placed in a fixed location in a corridor or a hall (Figure 5.2). Up to eight dynamic objects — seven humans and one robot — are simultaneously visible and labeled at point-level. The data set provides challenging situations with occlusions, groups of close objects moving in the same direction, rapid velocity changes and other dynamic objects, like doors.

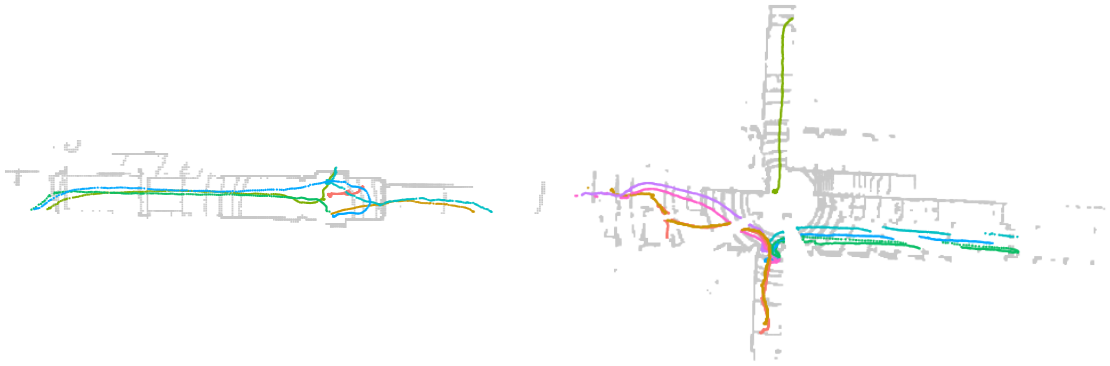

FIGURE 5.2: Top down views on two example sequences of the InLiDa [25] — corridor left, hall right. The dynamic objects' paths are colored. The robot's path is colored in red.

5.2.1.2 Simulated

Additionally, we simulated the Velodyne VLP-16 in a set of diverse outdoor settings to generate another data set (Figure 5.3). For simulation, we used Gazebo [46] and adapted a Velodyne VLP-16 simulator [47] to generate organized point clouds. The data set provides sequences with a varying number of dynamic persons within static environments with different amounts of clutter and distractions (Table 5.1). Our simulated persons avoid obstacles, change their velocities — $3.5\frac{km}{h}$ to $12.5\frac{km}{h}$ — and pause from time to time. The environments are limited to a distance of up to 140m to the static sensor. This implies that measurements of target objects do get very sparse or disappear completely due to occlusions or objects leaving the measurement range of the sensor. Ground truth labels are provided as axis aligned bounding boxes.

ID	Area	Targets	Duration	Scans	Environment
1	100m × 100m	6	98s	986	Empty field
2	100m × 100m	50	99s	995	Empty field
3	200m × 200m	6	97s	978	Empty field
4	100m × 100m	6	54s	547	Industrial
5	200m × 200m	6	92s	925	Industrial, Shops, Houses
6	100m × 150m	6	97s	974	Park

TABLE 5.1: Properties of simulated sequences.

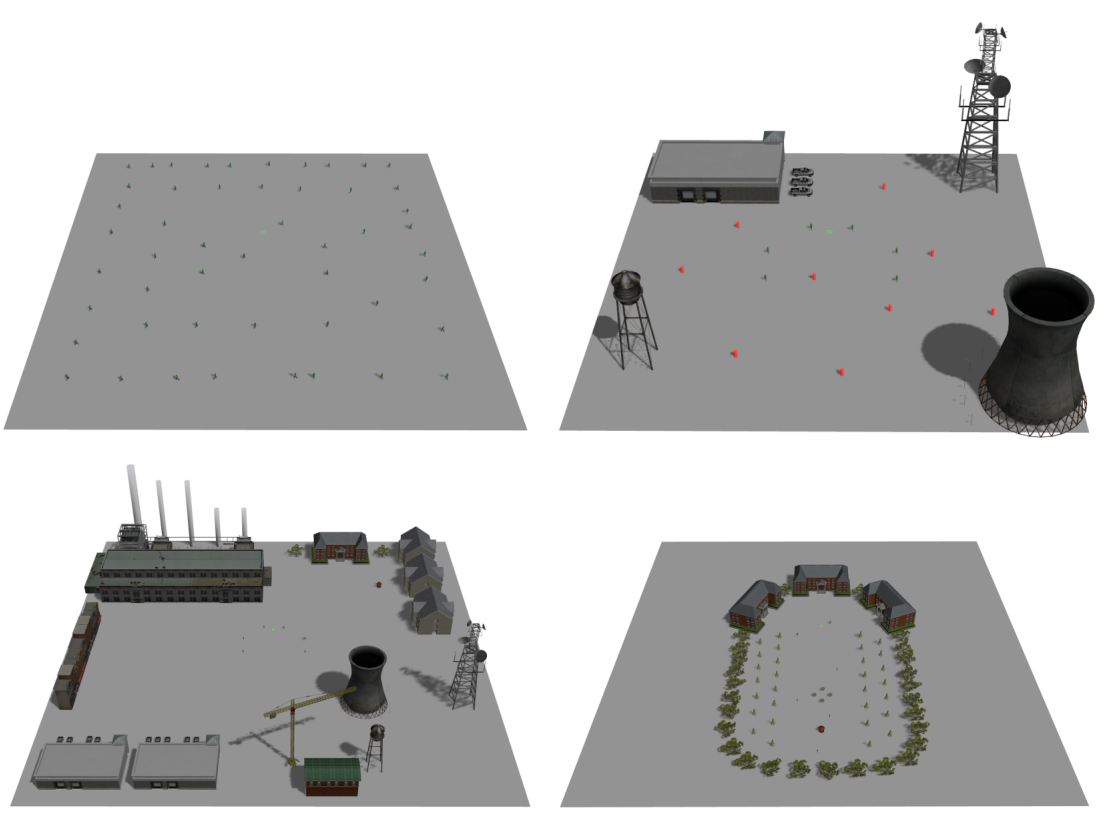

FIGURE 5.3: Exemplary views on environments provided in the sequences of the simulated data set. **Top Left**: Empty field sequences serve as a baseline providing easier circumstances for the pipeline. **Top Right**: Industrial setting with six dynamic persons and nine static distractors in red. **Bottom Left**: Mixed environment with apartments, industry and shops containing more natural distractors and occluders on a wider field. **Bottom Right**: A park with buildings, fountains and a variety of trees providing numerous possibilities for frequent short term occlusions.

5.2.2 Qualitative Evaluation

A quantitative evaluation of our pipeline on real world outdoor data sets was not possible. Publicly available data sets using the Velodyne VLP-16, like the L-CAS Multisensor People dataset [47] lack an exhaustive labeling, due to having a different purpose. Other data sets like the KITTY Object Tracking data set [17] or Spinello's 3D Point Cloud People data set [11] provide similar but denser scans from sensors such as the Velodyne HDL-64E. Again, both miss a complete labeling of the data. The first provides labels only for those objects that are visible in the camera images and additionally ignores objects at high distances, although clearly visible in the point clouds. The second restricts the sensors maximal measurement range of 120m and presents only measurements of a distance up to 20m.

Additionally, labels are provided only for those objects that are represented by at least 100 points and exceed 1m in height.

The last option of manually labeling our own data sets would have gone beyond the scope of this thesis. Hence, we circumvent this problem by recording a real world data set of several moving humans using an airborne sensor and evaluate the performance qualitatively. We filter out the dynamic humans and generate a map of the static part of the scene (Figure 5.4). This way it is easier to judge the pipeline's performance of tracking and filtering all dynamic objects fitting the model.

For each new point cloud we utilize the *Present Filter* discarding every point corresponding to an assigned detection. These filtered point clouds are saved in the *Cloud Log*. Once a hypothesis turns dynamic, all its previous bounding boxes — from the time it was static — are used in the *Past Filter* to remove corresponding points from logged clouds. If a lost dynamic track gets recovered, we provide the predicted bounding boxes from the steps in between similarly to the *Past Filter*.

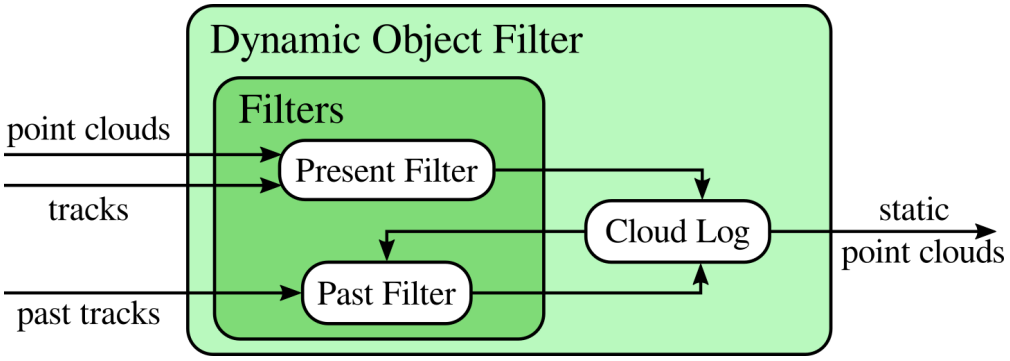

FIGURE 5.4: The Dynamic Object Filter separates measurements on tracked dynamic objects from points corresponding to the static part of the world. The latter are logged.

5.2.2.1 Real World Data

Our own data set was recorded during flights of a piloted UAV in the courtyard of the *Landesbehördenhaus* in Bonn (Figure 5.5). In the first sequence, only the pilot is visible to the sensor as a dynamic object. He moves in a slow pace within an area of about 7m × 7m.

The second sequence was recorded with four visible humans. They are walking and running, crossing each others' paths, changing speeds and occluding each other. The UAV and sensor are flying with velocities of up to $20\frac{km}{h}$.

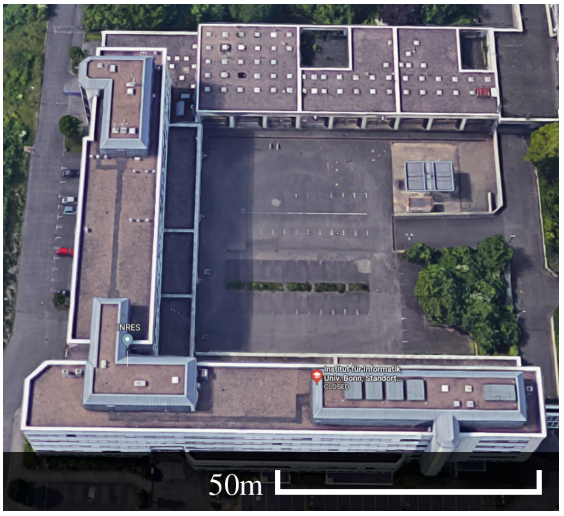

FIGURE 5.5: Google satellite map [7] of the Landesbehördenhaus in Bonn, Germany.

5.3 Optimization

For parameter optimization we used *hyperopt* [48]. Hyperopt is a distributed asynchronous hyperparameter optimization library. It utilizes the Tree of Parzen Estimators [49] to optimize parameters in a specified search space by minimizing a cost function depending on the given parameters.

The cost function we utilized is defined by

$$costs = 1 - MOTA. \tag{5.6}$$

The costs are minimal — equal to zero — for a perfect MOTA of 1.0 and rising as the MOTA decreases.

Finding the true optimal parametrization for our method is infeasible, due to the high dimensional parameter space and the unknown shape of the cost function. Hyperopt does not make that claim. Nevertheless, it is an approved tool to find a good parametrization in a short period of time.

Our method's parameters are:

Segmentation

sw_{min}	: minimal width of segment groups in meters (see 4.2)
sw_{max}	: maximal width of segment groups in meters (see 4.2)
δ_{min}	: parameter for conversion during segmentation (see 4.2.1)
δ_{low}	: parameter for conversion during segmentation (see 4.2.1)

Detection

c_{min}	: minimal number of points for a valid cluster (see 4.4.1.2)
r_{rg}	: search radius for region growing (see 4.4.1.2)
Δ_{rg}	: distance threshold for region growing (see 4.4.1.2)
h_{min}	: minimal height of a valid cluster in meters (see 4.4.2)
h_{max}	: maximal height of a valid cluster in meters (see 4.4.2)
w_{max}	: maximal width of a valid cluster in meters (see 4.4.2)
Λ	: maximal distance for filtering (see 4.4.2)

Tracking

Δ_{merge}	: distance threshold for hypotheses merging (see 4.5.6)
$\Delta_{correspondance}$	: distance threshold for assignment (see 4.5.3)
$cov_{increment}$	: covariance increase rate for uncorrected hypotheses (see 4.5.4)
cov_{max}	: maximally allowed covariance (see 4.5.4)

5.4 Results

In the following we are going to present and discuss the results of the evaluation. We start with the quantitative evaluation on the labeled data sets, continuing with the results generated using our real world data set and closing with a run time analysis of our pipeline. For all evaluations of our method, we use those hypotheses only that were classified as dynamic at least once.

5.4.1 Quantitative Results

For quantitative evaluations, we start by comparing the results of our approach to the results reported in the paper presenting the InLiDa [25]. Their multi object tracking method is denoted as *InLiDa Tracking* in the following. We discuss their evaluation approach and propose a different method usually applied to evaluate algorithms on small data sets. Using the latter, we inspect our approach on the InLiDa and a simulated outdoor data set we generated.

5.4.1.1 Comparison

InLiDa Tracking concentrates on multi person tracking. It utilizes global Ensemble of Shape Functions (ESF) descriptors [27] on extracted point clusters and classifies them using random forests into the classes *Person* and *Not person* to generate person detections. For tracking, existing hypotheses are matched to their closest detections within a search radius of 0.5m and propagated utilizing a circular velocity buffer.

The task for evaluation was to track humans only, distinguishing them from the dynamic robot present in four of six sequences. They evaluated their approach by training parameters on one InLiDa sequence and testing on the remaining. Each sequence was used for training once.

For a better comparison, we asked the authors to provide us with their evaluation script. The authors diverged from the official MOTA definition by replacing the Hungarian method with a naive assignment approach. It successively finds the closest hypothesis within a search radius of 0.5m around one object, assigns those if a match was possible and removes them for the assignments of the next objects within that time step. Unfortunately, there was a bug in their script, resulting in an assignment of an object to the last checked hypothesis within the search radius, instead of the closest. We contacted the authors and asked to provide us with a corrected version of the results for their method. The corrected results are presented and compared to our approach in table 5.2.

Additionally, we computed the total MOTA based on the provided MOTAs for each sequence. For this we reconstructed the number of errors, knowing the MOTA and the number of labels g_t at time t, on the basis of equation 5.5 by

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \tag{5.7}$$

$$\Leftrightarrow \quad \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} = 1 - MOTA \tag{5.8}$$

$$\Leftrightarrow \quad \sum_t (m_t + fp_t + mme_t) = (1 - MOTA) \cdot \sum_t g_t. \tag{5.9}$$

We compute the total MOTA by

Test Set	Sequence		1	2	3	4	5	6
Train 1	InLiDa	MOTA	0.966	-0.649	-0.180	0.490	0.703	0.484
		MOTP	162mm	137mm	158mm	110mm	159mm	110mm
	Ours	MOTA	0.874	0.134	0.420	0.108	-1.730	-0.289
		MOTP	191mm	213mm	199mm	156mm	224mm	159mm
Train 2	InLiDa	MOTA	0.881	0.920	0.761	0.503	0.534	0.591
		MOTP	156mm	136mm	146mm	121mm	135mm	94mm
	Ours	MOTA	0.558	0.813	0.466	0.296	-0.912	0.084
		MOTP	144mm	175mm	192mm	166mm	212mm	220mm
Train 3	InLiDa	MOTA	0.906	0.174	0.971	0.534	0.684	0.461
		MOTP	155mm	142mm	180mm	111mm	158mm	112mm
	Ours	MOTA	0.752	0.262	0.871	0.394	0.033	0.480
		MOTP	134mm	150mm	162mm	182mm	167mm	118mm
Train 4	InLiDa	MOTA	-0.797	-8.107	-1.175	0.932	-1.117	0.233
		MOTP	159mm	137mm	165mm	113mm	161mm	124mm
	Ours	MOTA	0.484	-0.007	-0.172	0.634	-2.408	-0.492
		MOTP	70mm	98mm	93mm	93mm	105mm	78mm
Train 5	InLiDa	MOTA	0.164	-3.499	-2.081	0.558	0.943	0.579
		MOTP	157mm	127mm	155mm	109mm	171mm	117mm
	Ours	MOTA	0.033	0.239	-0.144	0.462	0.723	0.345
		MOTP	92mm	104mm	117mm	105mm	137mm	105mm
Train 6	InLiDa	MOTA	-0.033	-2.833	-1.723	0.666	0.743	0.908
		MOTP	146mm	144mm	158mm	116mm	170mm	121mm
	Ours	MOTA	0.775	0.367	0.529	0.454	0.193	0.725
		MOTP	105mm	124mm	119mm	105mm	140mm	88mm

TABLE 5.2: Evaluation results on the InLiDa per sequence. MOTA and MOTP after training on one sequence and evaluating on the remaining. We compare the results of the InLiDa Tracking to our approach by highlighting our results in green, where they are better and in red where they are worse.

$$MOTA_{total} = 1 - \frac{\sum_s \sum_{t_s} (m_{t_s} + fp_{t_s} + mme_{t_s})}{\sum_s \sum_{t_s} g_{t_s}} \tag{5.10}$$

for all sequences s that were used for testing only.

In contrast to averaging the individual results, this way sequences containing more target objects have a higher weight — as intended for the MOTA. The InLiDa Tracking achieves a total MOTA of -0.213 — our approach a total MOTA of 0.071. The resulting MOTAs are rather low, considering that using no tracker

at all results in a MOTA of zero. One possible reason is the utilized evaluation procedure. Training on one sequence only increases the risk of overfitting the parameters. Additionally, the methods have difficulties to distinguish between the robot and humans, if there is not robot present in the training sequence. Applying the methods to other unseen sequences, with an attendant robot, yields bad results.

5.4.1.2 InLiDa

We evaluated our method a second time on the InLiDa. Due to the limited size of the data set, we perform a Leave-one-out cross-validation (LOOCV). For this, we split the data set containing n sequences into a training set of size $n-1$ for parameter optimization and a test set consisting of the left out sequence. The test set serves the purpose of evaluating the method's performance and ability to generalize on unseen data. This process is successively repeated n-times, each time leaving another sequence out. Table 5.3 presents the results of this evaluation. The total MOTA of 0.526 is computed as described in equation 5.10. The total MOTP of 0.108m is defined similarly based on equation 5.1 by

$$MOTP_{total} = \frac{\sum_s \sum_{t_s} d_{t_s}}{\sum_s \sum_{t_s} c_{t_s}}. \tag{5.11}$$

Test Set Sequence	1	2	3	4	5	6
↑ MOTA on Train Set	0.698	0.667	0.600	0.749	0.690	0.696
↑ MOTA	0.624	0.515	0.632	0.397	0.086	0.699
↓ Miss Ratio	0.133	0.121	0.094	0.596	0.224	0.252
↓ False Positive Ratio	0.240	0.360	0.272	0.005	0.684	0.044
↓ Mismatch Ratio	0.003	0.004	0.003	0.003	0.008	0.006
↑ MOTP IoU	0.608	0.749	0.688	0.531	0.507	0.501
↓ MOTP Eucl. in m	0.094	0.063	0.073	0.097	0.145	0.143
Mostly Tracked	0.800	0.400	0.667	0.167	0.800	0.250
Partially Tracked	0.200	0.600	0.333	0.333	0.200	0.750
Mostly Lost	0.000	0.000	0.000	0.500	0.000	0.000

TABLE 5.3: Evaluation results of the presented method on the **InLiDa** per sequence. Arrows indicate whether a higher value is better ↑ or a lower ↓. Best values per metric are colored in green — worst in red.

The worst result with respect to the MOTA is achieved when evaluating on sequence five. The low MOTA is mostly a result of misclassifying one static object that is present for whole sequence as dynamic. This induces a high false positive ratio which has a negative effect on the MOTA, despite the low miss ratio. The bad miss ratio in sequence four however is conditioned by a group of standing people occluding each other for a large part of the sequence.

We report the optimized parameters corresponding to the evaluation with the best MOTA on the test set (Table 5.4).

Segmentation		**Detection**		**Tracking**	
sw_{min}	: 0.013	c_{min}	: 7	Δ_{merge}	: 0.672
sw_{max}	: 0.590	r_{rg}	: 4	$\Delta_{correspondance}$	: 5.446
δ_{min}	: 0.070	Δ_{rg}	: 0.278	$cov_{increment}$	: 0.503
δ_{low}	: 0.872	h_{min}	: 0.731	cov_{max}	: 1.813
		h_{max}	: 1.531		
		w_{max}	: 1.339		
		Λ	: 0.110		

TABLE 5.4: Optimized parameters for InLiDa sequences one to five.

To get a better insight into the method's ability to generalize to unseen data, we plot the MOTAs $\{x_1, \ldots, x_n\}$ on the training set against the MOTAs $\{y_1, \ldots, y_n\}$ on the test set computed during n runs of the optimization process. For each sequence, we additionally report the Pearson correlation coefficient r defined by

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.12}$$

with $\bar{x}$ and $\bar{y}$ as the means of the MOTAs on the training and test set, respectively (Figure 5.7).

5.4.1.3 Simulated

For optimization using hyperopt, we need to set a range for each parameter to define which values are valid inputs. As it is not trivial to set these borders of the search space correctly, we optimized the parameters on the whole data set first. After that, we increased the search space at those points where the optimized parameters were close to its limits. We did not use any other knowledge from that initial optimization for all subsequent parameter optimizations.

Again we performed a LOOCV, this time on the simulated outdoor data set. The results are presented in table 5.5. Our method achieves a total MOTA of 0.677 and a total MOTP of 0.044m on the evaluation sets. Sequence 5 poses as a special challenge. In this sequence, a person collides with the sensor resulting in an impulse, causing the tracking to classify some static objects as dynamic. Hence, inducing a rise of the false positive ratio. Nevertheless, the method is able to track the target objects robustly, indicated by a low miss ratio of 0.05 and mismatch ratio of 0.0.

Test Set Sequence	1	2	3	4	5	6
↑ MOTA on Train Set	0.781	0.860	0.814	0.823	0.840	0.794
↑ MOTA	0.748	0.742	0.790	0.931	-0.379	0.791
↓ Miss Ratio	0.112	0.183	0.133	0.034	0.048	0.115
↓ False Positive Ratio	0.139	0.074	0.077	0.035	1.330	0.094
↓ Mismatch Ratio	0.000	0.002	0.001	0.000	0.000	0.001
↑ MOTP IoU	0.527	0.695	0.588	0.725	0.520	0.754
↓ MOTP Eucl. in m	0.097	0.046	0.013	0.067	0.010	0.038
Mostly Tracked	0.833	0.680	0.833	1.000	0.833	0.833
Partially Tracked	0.167	0.320	0.167	0.000	0.167	0.167
Mostly Lost	0.000	0.000	0.000	0.000	0.000	0.000

TABLE 5.5: Evaluation results of the presented method on the **simulated data set** per sequence. Arrows indicate whether a higher value is better ↑ or a lower ↓. Best values per metric are colored in green — worst in red.

We report the optimized parameters corresponding to the evaluation with the best MOTA on the test set (Table 5.6).

Segmentation		**Detection**		**Tracking**	
sw_{min}	: 0.027	c_{min}	: 1	Δ_{merge}	: 1.998
sw_{max}	: 0.944	r_{rg}	: 6	$\Delta_{correspondance}$	: 7.683
δ_{min}	: 0.299	Δ_{rg}	: 2.397	$cov_{increment}$	: 0.508
δ_{low}	: 0.663	h_{min}	: 0.603	cov_{max}	: 3.372
		h_{max}	: 2.523		
		w_{max}	: 1.412		
		Λ	: 1.963		

TABLE 5.6: Optimized parameters for simulated sequences, evaluated on sequence four.

To get a better insight into the method's ability to generalize across different environments and unseen data, we again plot the MOTAs achieved on the training set

against the MOTAs of the test set and compute the Pearson correlation coefficient r for each sequence (Figure 5.7).

5.4.2 Qualitative Results

For a qualitative evaluation, we filter the sequences recorded in the courtyard of the Landesbehördenhaus as described in 5.2.2. The mapped scans are presented in Figure 5.8. Measurements on dynamic objects and the UAV itself create artifacts. Our method is able to filter out most points corresponding to those objects, even for noisy mapping results present in the first sequence.

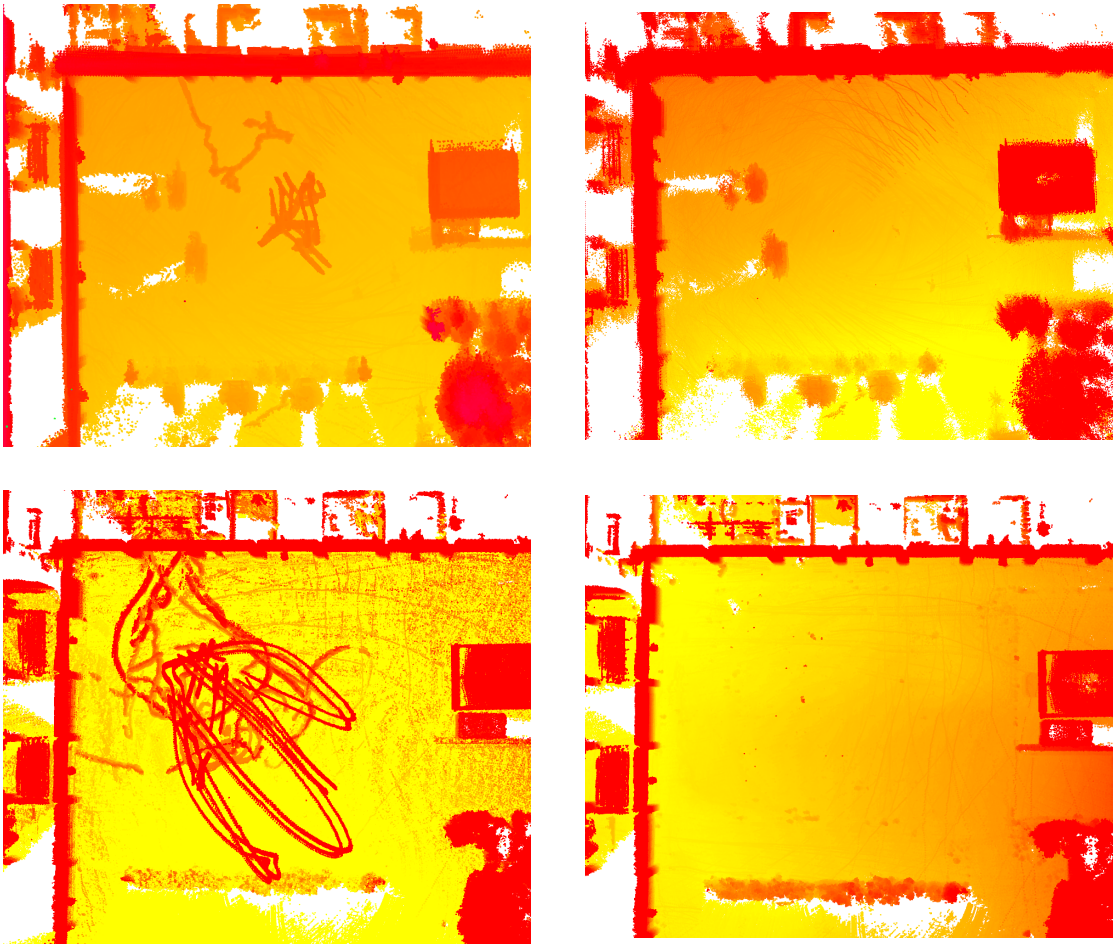

FIGURE 5.8: Top down views of two mapped sequences. Color encodes height — yellow low, red high. **Left**: Raw measurements mapped into one coordinate frame. Measurements on dynamic objects and UAV result in artifact visible as red or orange lines on the yellow ground. **Right**: Same scans and mapping with dynamic objects filtered out.

5.4.3 Run Time

Finally, we inspect the real-time capability of our method by plotting the run time per scan for three example sequences (Figure 5.9). We measured the time for each module — Segmentation, Detection and Tracking — to process incoming data using the *chrono* library provided in C++. Hence, we assume a sequential procession of the data. In practice, all modules are able to process the data of the next time step directly after processing the current data. The method was executed on the hardware on a UAV consisting of a *Intel Core i7-6770HQ* CPU and 32 GB of RAM.

For InLiDa and the simulated data set we chose the most demanding sequences — a sequence in the hall with a wall close to the sensor resulting in large kernel sizes during segmentation and the simulated sequence with 50 persons present. The final sequence was recorded during a flight in the courtyard of the Landesbehördenhaus. The run time of the detection module is increased for this sequence because of the data's transformation into the fixed world coordinate frame. This transformation is estimated in the mapping. Hence, we need to wait for it to process the data. We wait up to 500ms for this transformation — hence the spike in the plot. In all other cases, our method processes the data before the next scan is available after 100ms.

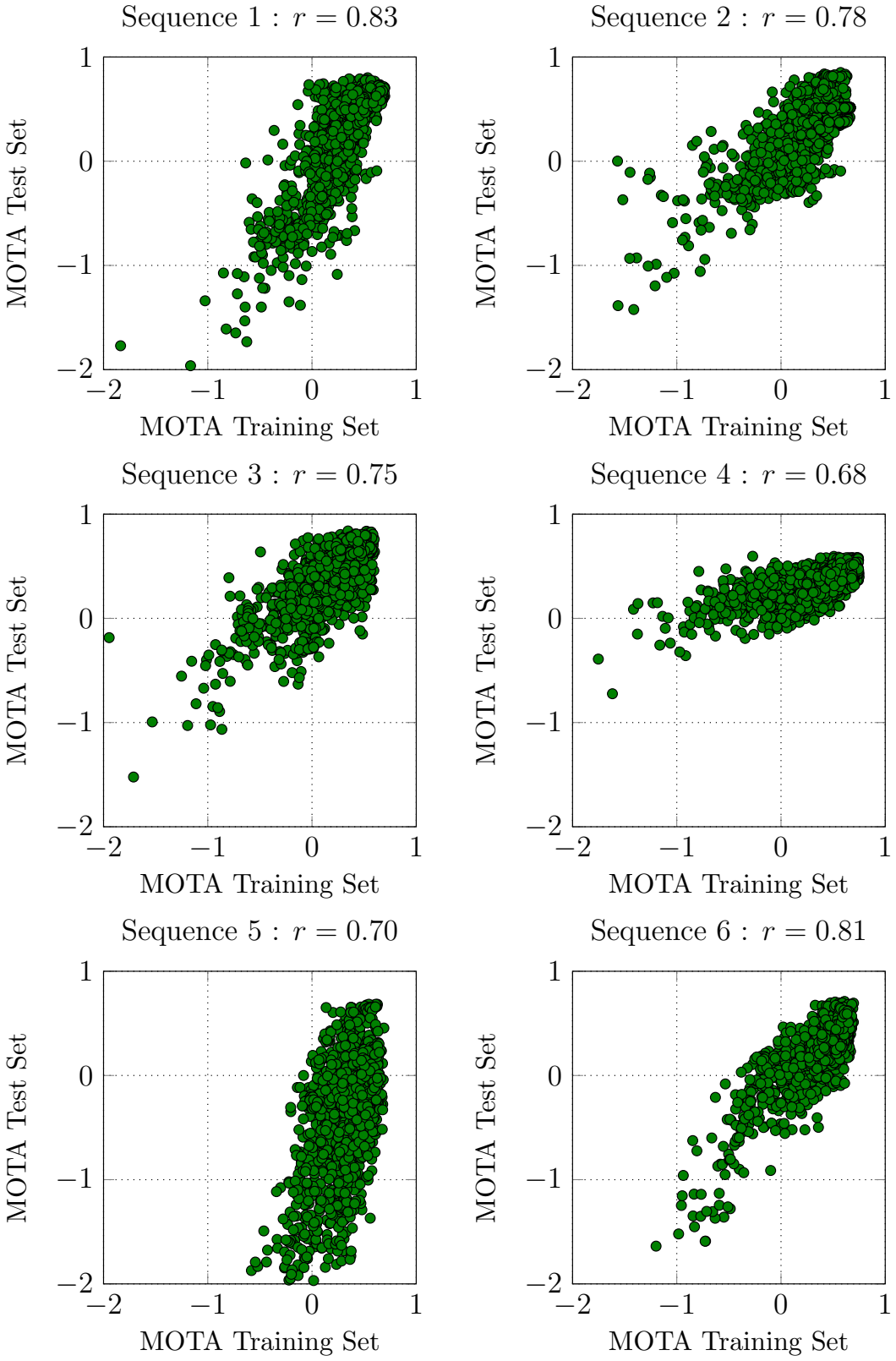

FIGURE 5.6: **InLiDa**: Plots visualizing the MOTA on the training set against the MOTA on the test set during parameter optimization. The title of the plot reports the sequence utilized as the test set. The Pearson correlation coefficient r indicates our method's ability to generalize to the unseen data.

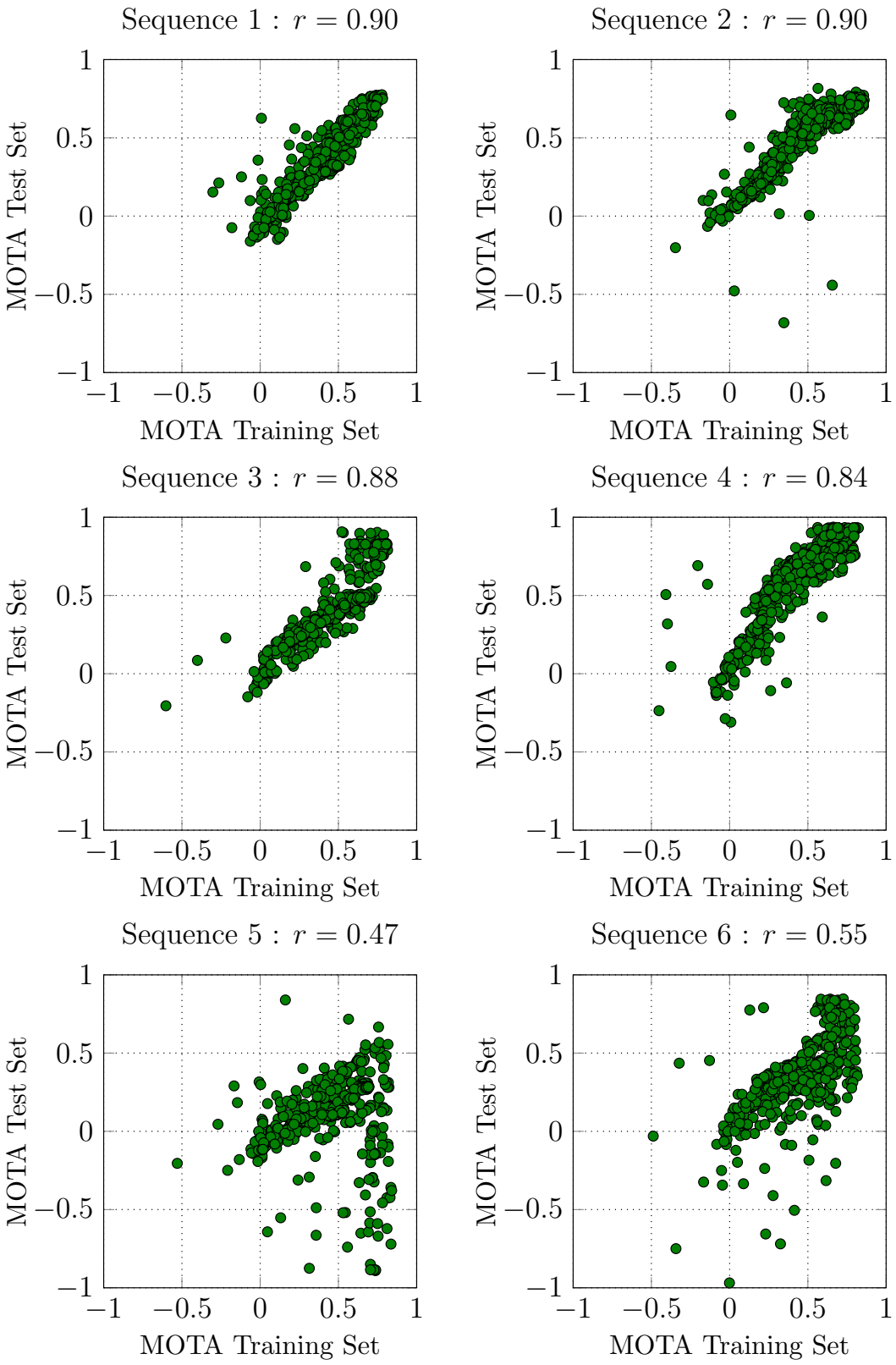

FIGURE 5.7: **Simulated Data**: Plots visualizing the MOTA on the training set against the MOTA on the test set during parameter optimization. The title of the plot reports the sequence utilized as the test set. The Pearson correlation coefficient r indicates our method's ability to generalize to the unseen data.

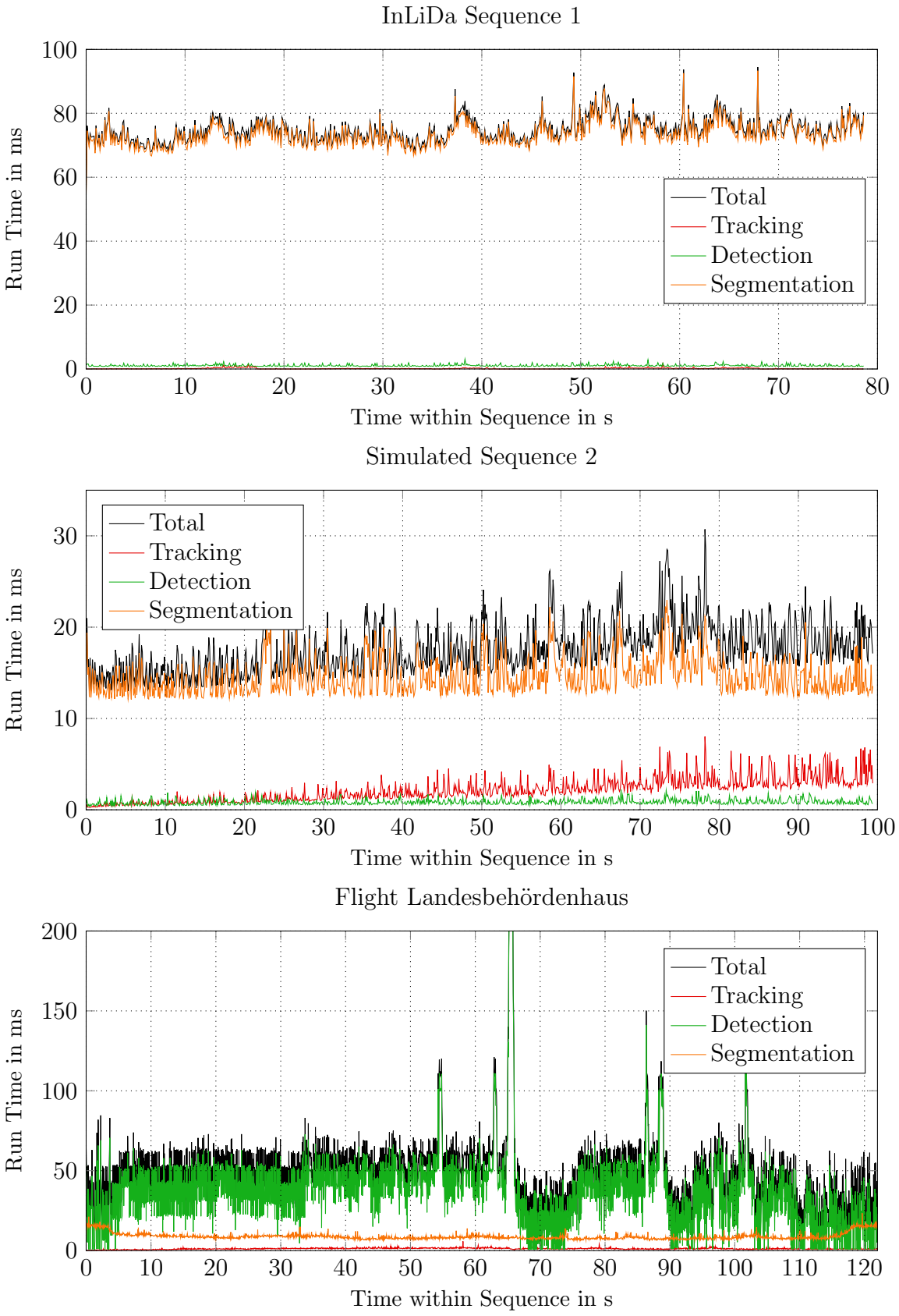

FIGURE 5.9: Run time in milliseconds of method and modules per scan on exemplary sequences.

Chapter 6

Conclusion

6.1 Summary

We implemented a method for real-time tracking of multiple small objects in the data of a Velodyne VLP-16 using the limited hardware on a UAV. For this, we utilized a novel segmentation approach to segment point groups of a specified width range in single scan rings. We deploy an adapted clustering method based on region growing on the segmented data. The resulting clusters are filtered according to simple geometric specifications and temporal information about already tracked objects to generate detections. These detections are provided to a multi object tracker using one Kalman filter for each tracked object.

We evaluated our approach against another state-of-the-art method on a publicly available indoor data set generating comparable results. Additionally, we generated a simulated outdoor data set for further quantitative evaluations. According to two metrics that are utilized in established benchmarks [17, 43], our approach is able to achieve even better results on the latter data set.

For qualitative evaluations, we implemented a filter to separate the static part of the environment in the data from the dynamic part. We applied the tracking and filtering to two real world data sets recorded by the sensor mounted on an airborne UAV. The approach successfully filters out most of the dynamic objects.

6.2 Prospects

For future work, the application of a more sophisticated but still efficient tracking approach should be investigated. Complex movement patterns of humans can only be tracked to a limited extent by the utilized Kalman filter. Furthermore, the information about occlusions could help to adjust the time an occluded hypothesis is retained.

The same information can be utilized to more robustly distinguish between static and dynamic objects. During partial occlusions, the bounding box of the occluded object changes its shape and size, as the object is only partially visible to the sensor. In some instances this is interpreted as a movement of the occluded object. Incorporating this information would counteract false classifications. Hence, enable the approach to filter dynamic objects more precisely.

Lastly, the proposed segmentation method is efficient only if the sensor has a certain distance to the environment. For close objects combined with a large specified target width, the run time of the object filter is increased drastically. Replacing the object filter by an approach that compares the measured distance of a point to the distances of two neighboring background points could generate a similar segmentation while being computationally more efficient.

Bibliography

[1] David Droeschel and Sven Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. 2018.

[2] Marius Beul, David Droeschel, Matthias Nieuwenhuisen, Jan Quenzel, Sebastian Houben, and Sven Behnke. Fast autonomous flight in warehouses for inventory applications. *IEEE Robotics and Automation Letters*, 3(4):3121–3128, 2018.

[3] Seon Jin Kim, Gino J. Lim, Jaeyoung Cho, and Murray J. Côté. Drone-aided healthcare services for patients with chronic diseases in rural areas. *Journal of Intelligent & Robotic Systems*, 88(1):163–180, 2017.

[4] Velodyne LiDAR. *VLP-16 User Manual*. Velodyne LiDAR, 63-9243 rev. d edition, 12 2017.

[5] Craig Glennie and Derek D. Lichti. Static calibration and analysis of the velodyne hdl-64e s2 for high accuracy mobile scanning. *Remote Sensing*, 2 (6):1610–1624, 2010.

[6] Craig Glennie, Arpan Kusari, and A Facchin. Calibration and stability analysis of the vlp-16 laser scanner. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3:W4, 2016.

[7] Google. Landesbehördenhaus, (n.d). URL `https://goo.gl/maps/3tNpQEdmqTy`.

[8] Greg Welch and Gary Bishop. Scaat: Incremental tracking with incomplete information. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 333–344. ACM Press/Addison-Wesley Publishing Co., 1997.

[9] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. People-tracking-by-detection and people-detection-by-tracking. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[10] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Motion-based detection and tracking in 3d lidar scans. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4508–4513. IEEE, 2016.

[11] Luciano Spinello, Kai Oliver Arras, Rudolph Triebel, and Roland Siegwart. A layered approach to people detection in 3d range data. In *AAAI*, volume 10, pages 1–1, 2010.

[12] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[13] Daniel Maturana and Sebastian Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3471–3478. IEEE, 2015.

[14] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[15] Adam Coates and Andrew Y. Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*, pages 561–580. Springer, 2012.

[16] Mehmet Ali Çagri Tuncer and Dirk Schulz. A hybrid method using temporal and spatial information for 3d lidar data segmentation. In *ICINCO (2)*, pages 162–171, 2017.

[17] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] Rudolf Frühwirth. Application of kalman filtering to track and vertex fitting. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 262(2-3):444–450, 1987.

[19] Frank Moosmann and Christoph Stiller. Joint self-localization and tracking of generic objects in 3d range data. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1146–1152. IEEE, 2013.

[20] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[21] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[22] Frank Moosmann, Oliver Pink, and Christoph Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *Intelligent Vehicles Symposium*, pages 215–220. IEEE, 2009.

[23] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2): 239–256, 1992.

[24] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *European conference on computer vision (ECCV)*, pages 356–369. Springer, 2010.

[25] Cristina Romero-González, Álvaro Villena, Daniel González-Medina, Jesus Martínez-Gómez, Luis Rodríguez-Ruiz, and Ismael García-Varea. InLiDa: a 3d lidar dataset for people detection and tracking in indoor environments. In *12th International Conference on Computer Vision Theory and Applications*, 2017.

[26] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[27] Walter Wohlkinger and Markus Vincze. Ensemble of shape functions for 3d object classification. In *Robotics and Biomimetics (ROBIO), IEEE International Conference on*, pages 2987–2992. IEEE, 2011.

[28] Anton Milan, Seyed Hamid Rezatofighi, Anthony R. Dick, Ian D. Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *AAAI*, pages 4225–4232, 2017.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. *arXiv preprint arXiv:1602.00991*, 2016.

[31] Julie Dequaire, Peter Ondrúška, Dushyant Rao, Dominic Wang, and Ingmar Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 2017.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[33] Julie Dequaire, Dushyant Rao, Peter Ondruska, Dominic Wang, and Ingmar Posner. Deep tracking on the move: Learning to track the world from a moving vehicle using recurrent neural networks. *arXiv preprint arXiv:1609.09365*, 2016.

[34] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[35] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. Model-free detection and tracking of dynamic objects with 2d lidar. *The International Journal of Robotics Research*, 34(7):1039–1063, 2015.

[36] Hafez Farazi and Sven Behnke. Online visual robot tracking and identification using deep lstm networks. 2017.

[37] Robert K. McConnell. Method of and apparatus for pattern recognition, January 28 1986. URL `http://www.google.co.uk/patents/US4567610`. US Patent 4,567,610.

[38] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[39] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[40] David Droeschel, Jörg Stückler, and Sven Behnke. Local multi-resolution surfel grids for mav motion estimation and 3d mapping. In *Proc. of the Int. Conference on Intelligent Autonomous Systems (IAS)*, 2014. URL `http://www.ais.uni-bonn.de/papers/IAS_2014_Droeschel_MAV-Mapping.pdf`.

[41] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[42] Peter Morton, Bertrand Douillard, and James Underwood. An evaluation of dynamic object tracking with 3d lidar. In *Proc. of the Australasian Conference on Robotics & Automation (ACRA)*, 2011.

[43] Laura Leal-Taixé, Anton Milan, Ian D. Reid, Stefan Roth, and Konrad Schindler. MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]*, April 2015. URL `http://arxiv.org/abs/1504.01942`. arXiv: 1504.01942.

[44] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008.

[45] Yuan Li, Chang Huang, and Ram Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. 2009.

[46] Nathan P. Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. Citeseer, 2004.

[47] Dataspeed Inc. Velodyne simulator. `https://bitbucket.org/DataspeedInc/velodyne_simulator`, 2018.

[48] James Bergstra, Dan Yamins, and David D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20. Citeseer, 2013.

[49] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.