Drone segmentation and tracking in grounded sensor scanned LiDAR datasets

Kristijan Mirčeta[1], Ciril Bohak[1], Byeong Hak Kim[2], Min Young Kim[2], Matija Marolt[1]

[1]*University of Ljubljana, Faculty of Computer and Information Science*
[2]*Kyungpook National University (Daegu, South Korea)*
E-mail: kristijan.mirceta@gmail.com, ciril.bohak@fri.uni-lj.si, durumy98@hanmail.net, minykim@knu.ac.kr, matija.marolt@fri.uni-lj.si

Segmentacija in sledenje dronom v podatkih prizemljenega senzorja LiDAR

V tem članku se spopadamo s problemom sledenja dronom v zunanjih scenah, ki jih skenira prizemljen senzor LiDAR. Problem razdelimo v 3 podprobleme: (1) segmentacija scene, (2) detekcija dronov, (3) sledenje dronom. Za segmentacijo scene uporabimo metodo radialno omejenih najbližjih sosedov. Za detekcijo dronov uporabljamo konvolucijske nevronske mreže. Za učenje mreže in za generiranje učne množice uporabljamo orodje Blensor, s katerim sintetizeramo sceno, ki želimo, da dobro simulira realnost. Na koncu, po detekciji dronov, le-tem tudi sledimo skozi več časovno zaporednih okvirjev s pomočjo Kalmanovega filtra, ki uporablja križno-korelacijski filter kot opazovalni model, skoraj konstantno hitrost kot model gibanja, ter vokseliziran deskriptor kot vizualni model.

Abstract

In this paper, we tackle the problem of tracking drones in outdoor scenes, scanned by a grounded LiDAR sensor. The problem is split into 3 subproblems: (1) segmentation of the scene, (2) drone detection and (3) drone tracking. For segmenting the scene, we use a Radially Bounded Nearest Neighbors method. For drone detection we use a Convolutional neural network (CNN). To train the CNN and produce a dataset of drones we use the Blensor framework, with which we synthesize drone examples, with the goal of them resembling realistic conditions. Finally after detecting the drones, we track them throughout multiple time frames using a Kalman filter, which uses a cross-correlation filter as the observation model, Nearly constant velocity as the motion model and a voxelized 3D descriptor for the visual model.

1 Introduction

In the cities of the future, Unmanned aerial vehicles (UAV) will have an important role in many functions of society, such as transport, delivery of goods, and surveillance systems [6]. But just as this technology could be very useful for society, it may also be used with malicious intent, for activities such as unauthorized access to off limits areas, spying on unsuspecting victims or disruption of other good-intented UAVs. Because of this, building a system for combating against malicious drones is an interesting and worthwhile problem. We want to detect and track drones to be able to avoid them (as other drones), destroy them (when they are in off limits areas) or inspect their activity to reveal patterns signifying malicious intent. In this paper, we are working with UAVs in LiDAR scenes, which are sparse 3D point cloud scenes. The sparsity of the LiDAR data is a big limitation, since it holds that the further away from the sensor an object is, the more sparsely it is sampled. This means that the extracted features will be less informative for detectors or trackers. In the following sections, we first present the data we are working on in Section 2, then we decompose the problem into 3 stages: segmentation in Section 3, detection in Section 4 and tracking in Section 5, and elaborate on each of them. In the final Section 6 we present conclusions and give pointers for future work.

2 Data

The driving force of today's AI algorithms is data. This puts large firms like Google, Amazon and Microsoft in a strongly advantageous position compared with their competition in the form of start-ups which may have great ideas, but no data to bring them to fruition. The recently emerging trend of synthetic data generation may be a way to democratize AI. The idea is that, instead of gathering massive amounts of data from the real world and labeling it by hand, we should create artificial datasets with which we may train our AI. Some researchers have already shown how synthetic data leverages the performance of their models when applied to real world cases. An example is [8], where the authors generated objects in the Unity game engine[1], and Blender[2] where they labeled key points on the objects in the virtual world and learned to predict the orientation of the object. This let them predict orientations of objects in the real world when filmed with a camera as well. There are also other packages like [17], which implements a wide range of sensors to scan these virtual worlds, and learn to predict various targets from this generated data in the hope of accurately simulating such data sensors in the real world. Because the generated data should be as close to real world data as possible, the Unreal Engine 4[3] is a great option due to

[1] https://unity3d.com/
[2] https://www.blender.org/
[3] https://www.unrealengine.com/

its Physically based rendering (PBR) capabilities which are unparalleled at the moment in similar software. Some packages like [18] work as a plugin to Unreal Engine 4, where both data is synthesized and simulation tests for self-driving cars can be made, and there are also packages higher in abstraction which provide capability for general computer vision tasks. An example is UnrealCV [15].

In this fashion, we are using Blender with the Blensor framework [5], to simulate virtual worlds and scan them using a LiDAR sensor. For our scenes, we are scanning with a simulated Velodyne HDL-64e grounded sensor.

We use a scene generator, which generates a scene with a small number of trees, primitive shapes (cubes, cylinders and prisms) and drones. These shapes are uniformly distributed around a 50×50 meter field, and the drones can fly on heights of up to 12.5 meters. Meanwhile we enforce the constraint that no objects can overlap with each other. We do not include the ground, as removing it is a very well researched problem and has been shown empirically to improve segmentation results by [2].

We generated 1000 such scenes, and scanned them using Blensor, where the scanner was located in the coordinate origin $(0, 0, 0)$, which resulted in our final input dataset - a point cloud with (x, y, z) values.

When testing out our methods, the scenes would be split as per standard practice in machine learning, where there would be a training set and a test set, roughly in the proportion of $4 : 1$.

Because the scenes are generated, we basically had the labels and bounding boxes for each item in the scene, and were therefore fully able to measure the accuracy of our methods using standard procedures. An example picture of a scanned scene can be seen in Fig. 1.

3 Segmentation

For the segmentation of the scene, we used a simple method called Radially Bounded Nearest Neighbours (RBNN) [7]. This method basically performs a clustering of the points, by declaring all points that are transitively less than some defined threshold (radius r) distant from one another to belong to the same object. Because there is no overlapping of objects in our scenes, and the objects are not very close to one another, this method was able to provide a sufficient segmentation for our needs. Most of the time the result of applying this method to our scene, was that each object was its own cluster. This provided a good basis for the detection of drones.

4 Detection

A bigger problem to tackle was the detection of the drones within the cluster. While we now know which points represent which objects, we need to find out which of the objects are in fact drones, and then track them.

The first thing to do was to find a descriptor for each cluster of points, then decide on whether it is a drone on the basis of the descriptor. Researchers have tried various different descriptors for various point cloud classification

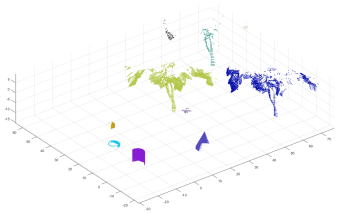

Figure 1: Plot of a segmented synthetic LiDAR scene. The gray object in the middle is the drone.

tasks, most prevalent among which fit into the following categories:

- **Multi-view descriptors**, where 2D projections (or picture) are taken from various angles of the point cloud . An example is in [19], where they classify office furniture via feeding multiview pictures into a CNN.

- **Voxelized descriptors**, where the point cloud is transformed into a regularized voxel grid. This can be done so, that if a point falls into a bin within the voxel grid, that voxel is set to 1. Then in some cases a Gaussian filter can be run through this grid, to give the voxels more volume. Such an approach is useful with methods that use the notion of similarity between descriptors to aid in their prediction. An example is in [12], which uses such a descriptor for registration. Even a descriptor without Gaussian filtering is useful, as it can be fed into a CNN, and the CNN will find its own filters which may prove to be even more useful. Examples of these are [16] and [11], where the latter was considered state-of-the-art in 2015. A plot of an example voxelized descriptor of a drone is show in Fig. 2.

- **Raw point clouds**, where the point cloud is not preprocessed, but used directly. Some deep neural nets have been researched, where this method produces very good results such as [14, 13]. A thing to watch out for here is permutational invariance, where it is ensured that the result of the prediction is independent of the ordering of the points in the point cloud.

- **Contextual**, where not only the point cloud segment itself is used, but its relations with the other parts of the scene are also accounted for. Such methods can be used as a leverage to other ones, and are currently producing state-of-the-art results as can be seen in [9].

Many previously listed methods employed CNNs to solve the problem of detection. We used a similar architecture as was used by VoxNet [11], where only the last layer was changed into a dense layer with a softmax activation, meant to classify only into two classes - *drone* and *not a drone*. The architecture defines a convolutional layer with 32 5×5×5 filters with $stride = 2$ for the first

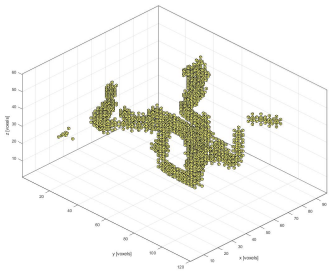

Figure 2: A visualization of the voxelized descriptor of a drone. The drone is dense because a Gaussian filter has been applied on it to give it more volume.

layer, another convolutional with 32 3×3×3 filters in the second layer, then a 2×2×2 max pooling layer, a 128 neuron fully connected layer which is used for the final prediction. A picture of the architecture can be seen in Fig.3.

The input is a voxelized descriptor, convolved with a Gaussian kernel with σ proportional to the distance from the sensor origin, because there is no way to inform the CNN about the variance in sampling density due to varying distance from the sensor origin. With applying the gaussian filter in such a way, we have encoded this information into the descriptor.

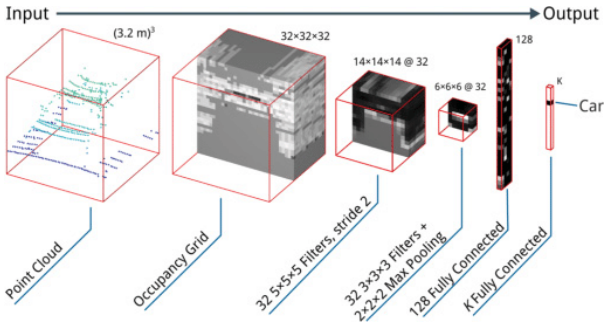

Figure 3: The original VoxNet CNN architecture. The point cloud is first mapped on to an occupancy grid (voxelized descriptor), flows through two convolutional layers, and finally through two fully connected layers to arrive at the prediction.

4.1 Detection Results

To train the CNN we used 2700 samples, of which there were roughly 300 drones, and the other objects were tree, cube, torus, cylinder, pyramid. The test set was 1000 samples, with roughly the same proportion (100 drones).

Through 10 random shufflings of the training and test set, the average AUC was above 0.99 when using a ROC curve, indicating that the CNN is a reliable classifier, while still being very fast. Though this result is very good, the reason is probably that the scenes we are predicting on are very simple. Nonetheless, it provides a good indicator that our direction of research is promising, and our next step would be to try this on more complex scenes.

5 Tracking

For the tracking, we were inspired by [12] where to track an object in a sequence of frames, an object was first segmented, then as stated before, a voxelized descriptor was used to match the objects in two subsequent frames using Bhattacharya similarity. If the similarity passed a certain threshold, then it was declared that the set of points in frame $t + 1$ was the same object as in frame t. The general idea is the same as the correlation filter [10].

In addition to this, we recognized a problem for the tracker, if the tracked object either came too close to another object or momentarily overlapped with it. For example if the drone passed through the edge of a tree canopy. In this case the tracker gets confused and loses track of the drone due to the similarity quickly dropping below a defined threshold.

To solve this, we used a Kalman filter [1] where the used observation model was the correlation filter, the visual model was the voxel grid, and the motion model was Nearly constant velocity (NCV). This provided our method with robustness to the stated problem. It is worthy to note here, that we assume the movement of the drone to be nearly linear, which is not a realistic assumption. Though research on how to relax this constraint has already been made with the extended Kalman filter [3] and unscented Kalman filter [20], which can track nonlinear movement as well.

5.1 Tracking Results

We tried tracking a drone following a linear path, but would pass through the edge of a tree canopy on its way. We tried applying only a correlation filter at first, where the tracker would get stuck in the tree canopy, unable to track the drone onwards. But as shown on Fig. 4, the Kalman filter was successful in overcoming its obstacle and tracking the drone all the way.

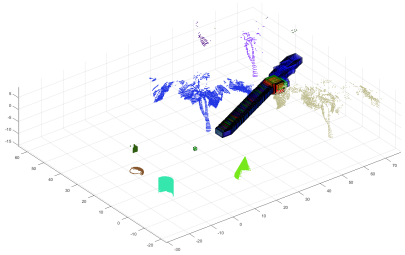

Figure 4: The picture shows multiple overlapped frames in a LiDAR scene, where the linear path of the drone is shown. The blue sequence of bounding boxes are the intermediate steps of the Kalman filter tracker. It can be seen that the Kalman filter was able to follow the linear path of the drone right through the tree canopy.

6 Conclusion

We have developed a pipeline of subproblems which defines a method for tracking drones in a LiDAR dataset. There are many opportunities to refine our method in more

domains. We could do the segmentation and detection steps in one single step instead of two, just as the current state-of-the-art method SPGraph [9]. We could also refine our neural network architecture to use the new information we are accounting for with relationships among objects and object parts. We believe that much more work should be done regarding the data generation, where we could use better simulations to extract RGB values from scenes as well, and perhaps use some modern techniques like Generative Adversarial Networks [4] to generate scenes that are more believable and realistic than our randomly crafted ones. By optimizing for realism in virtual scenes, we are also optimizing for performance in real-world scenes.

Lastly, regarding tracking, we could upgrade the method by using an unscented Kalman filter for tracking, to model nonlinear motion, which is more usable in reality. As tracking is also a bottleneck to real-time performance due to slow computation of cross-correlation, we could replace both the visual and observational models with faster ones, perhaps in RGB data, we would use color histograms instead of regularized voxel grids for similarity computation.

References

[1] Gary Bishop and Greg Welch. An Introduction to the Kalman Filter: SIGGRAPH 2001 Course 8. In *Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques*, pages 12–17, 2001.

[2] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the Segmentation of 3D LIDAR Point Clouds. In *2011 IEEE International Conference on Robotics and Automation*, pages 2798–2805, May 2011.

[3] Keisuke Fujii. Extended Kalman Filter. *Refernce Manual*, 2013.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender Sensor Simulation Toolbox. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, pages 199–208, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[6] Ismail Guvenc, Ozgur Ozdemir, Yavuz Yapici, Hani Mehrpouyan, and David W. Matolak. Detection, Localization, and Tracking of Anauthorized UAS and Jammers. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2017.

[7] Klaas Klasing, Dirk Wollherr, and Martin Buss. A Clustering Method for Efficient Segmentation of 3D Laser Data. *2008 IEEE International Conference on Robotics and Automation*, pages 4043–4048, 2008.

[8] Marcel Klomann, Michael Englert, Kai Weber, Paul Grimm, and Yvonne Jung. Improving Mobile MR Applications Using a Cloud-based Image Segmentation Approach with Synthetic Training Data. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology*, Web3D '18, pages 4:1–4:7, New York, NY, USA, 2018. ACM.

[9] Loïc Landrieu and Martin Simonovsky. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. *CoRR*, abs/1711.09869, 2017.

[10] Yang Li and Jianke Zhu. A Scale Adaptive Kernel Correlation Filter Tracker with Feature Integration. In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, pages 254–265, Cham, 2015. Springer International Publishing.

[11] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[12] Daniel D. Morris, Brian R. Colonna, and Paul H. Haley. LADAR-Based Mover Detection from Moving Vehicles. *CoRR*, abs/1709.08515, 2017.

[13] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv preprint arXiv:1706.02413*, 2017.

[14] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[15] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. In *Proceedings of European Conference on Computer Vision*, pages 909–916. Springer, 2016.

[16] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Classification of Point Cloud Scenes with Multiscale Voxel Deep Network. *CoRR*, abs/1804.03583, 2018.

[17] Raffaele Schiavullo. SYNCITY virtual hyper realistic simulator for machine learning to train ADAS systems. *Virtual Reality*, 4:06, 2018.

[18] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[19] Vladeta Stojanovic, Matthias Trapp, Rico Richter, and Jürgen Döllner. A Service-oriented Approach for Classifying 3D Points Clouds by Example of Office Furniture Classification. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology*, Web3D '18, pages 2:1–2:9, New York, NY, USA, 2018. ACM.

[20] Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. IEEE, 2000.