CNN-based Human Detection Using a 3D LiDAR onboard a UAV

Jack N.C. Hayton[1] Tiago Barros[2] Cristiano Premebida[2] Matthew J. Coombes[1] Urbano J. Nunes[2]

***Abstract*— This paper addresses the problem of detecting humans in a point cloud taken with a 3D-LiDAR onboard a UAV. The potential use cases of this technology are numerous, examples include security and surveillance, disaster relief and search and rescue operations. In this paper, a CNN-based approach is proposed which is able to analyse point clouds returned by a 3D LiDAR sensor in such a way that it can detect humans. The algorithm described here consists of 3 main components: data pre-processing, post-processing, and human classification. In this paper objects were assigned to one of two classes: human and non-human. The classification was performed by projecting the 3D point cloud onto a series of 2D planes using occupancy grid mapping. This creates a set of silhouettes of the object corresponding to the top, front and side views. Classification is achieved by supervised CNNs using single-view and multi-view (3 channels) images patches.**

I. Introduction and Related Work

The rise of robotics and autonomous driving over the past decade has brought LiDAR sensors into the mainstream, previously being too expensive and bulky to integrate into most robotics applications. LiDAR provides very accurate spatial awareness however, it is typically limited to significantly lower resolution than other common sensors such as cameras. In the context of robotics its primary function is to provide real-time spatial awareness that is used to control/navigate the robot. When dealing with the LiDAR data, also known as point cloud, one of the biggest challenges is the accurate detection of objects of interest within 3D set of data points.

Object classification and tracking using LiDAR has been thoroughly studied and its use has seen a sharp increase over the last decade due to advancements in robotics [1] and autonomous vehicles [2]. Most of the research carried out on LiDAR was from the perspective of unmanned (or autonomous) ground vehicles (UGVs) [3], however, over the past few years we have seen increased use on unmanned aerial vehicles (UAVs) [4] [5], primarily due to the reduction in size and weight of LiDAR systems.

Over the last decade, advancements in engineering and battery technology have brought consumer UAVs (or drones) into the mainstream opening the door to a host of potential uses of an airborne LiDAR [6] [7]. In particular, combining airborne LiDAR technology with object detection algorithms [8] offers many new possibilities as existing use cases are mainly restricted to static aerial mapping. The potential use cases are plentiful, including defense or law enforcement applications where a person must be tracked or followed in terrain that does not allow for ground vehicles. Additionally, such systems can be used in disaster relief or search and rescues operations where a UAV scans a given area, locates people and transmits their location.

Fig. 1. The octocopter drone (DJI S1000) with the onboard LiDAR (Velodyne VLP-16) used to collect point cloud data for human detection. The sensor coordinate reference system (F_{velo}) are designated by the *xyz* axes.

Most of the existing research work on small airborne LiDAR is related to terrain mapping, geo-surveying, agricultural robotics and digital elevation mapping. Tulldahl et al. [6] review application and capabilities of LiDAR from small UAV but, makes no mention of any object detection or tracking use-case. Commercial solutions exists, for examples the Phoenix LiDAR Systems which aims to provide a complete integrated solution for UAV mounted LiDAR used in mapping and geo-surveying applications [9]. Recently, Velodyne partnered with YellowScan to integrate its VLP-16 Puck LiDAR, the same model used in this project, into YellowScans surveyor for demanding UAV applications [10]. UAV mounted LiDAR systems are now being used for a more diverse set of applications; having previously been primarily used for mapping. To our knowledge, no study has hitherto been carried out on the validity of using an off-the-shelf LiDAR (such as the Velodyne Puck) mounted on a UAV to identify and track an object.

The scarcity of existing research on airborne LiDAR was an encouragement to collect real-world LiDAR data from a drone [11], as shown in Fig. 1. This work aims to develop an object detection algorithm based on CNN and by using 3D LiDAR mounted onboard a UAV. The key contributions are the following: (i) a dataset providing UAV mounted 3D LiDAR data for human detection; (ii) complete pipeline for point cloud pre&post-processing - including ground detection, clustering and occupancy-grid 2D representation; (iii)

[1] Authors are with the Department of Aeronautical and Automotive Engineering, Loughborough University, UK. Email: m.j.coombes@lboro.ac.uk

[2] Authors are with the Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra, Coimbra, Portugal. Emails: {tiagobarros, cpremebida, urbano}@isr.uc.pt

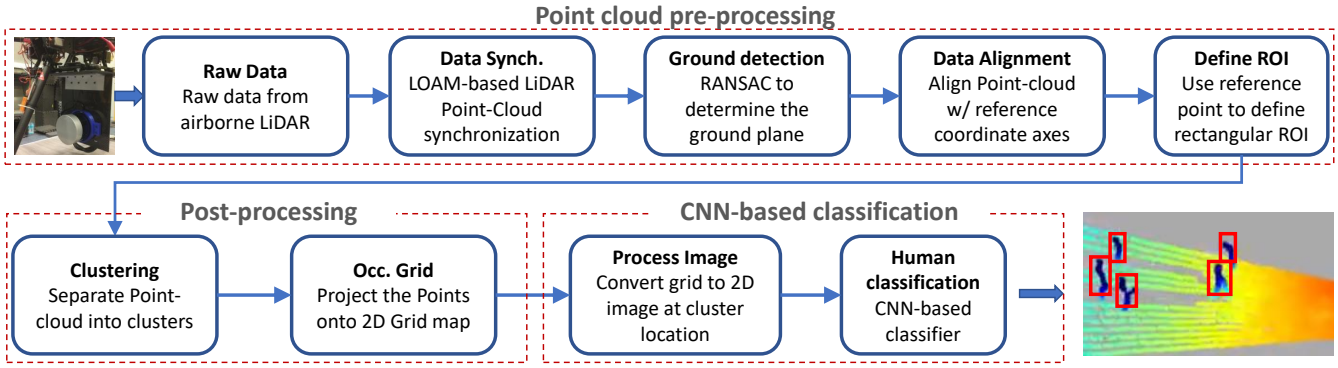

Fig. 2. Proposed pipeline of the 3D LiDAR-based human detection algorithm. Main modules are: pre-processing, post-processing, and object classification.

CNN-based human classification.

The remainder of this paper is structured as follows: Sect. II describes the methodology and the approach developed to process the 3D point clouds and detect humans. Section III details the dataset, the experiments and the results including augmented and non-augmented CNN models. Finally, Section IV concludes the study and present suggestions for future work.

II. Methodology and Proposed Method

This section details the developed algorithm that takes a point cloud delivered by a LiDAR onboard a drone, as an input, and outputs the positions and classes (or categories) of the objects within the field of view. A block diagram outlining the pipeline of the algorithm is given in Fig. 2, where the detection algorithm is broken down into 3 main modules: data pre-processing, post-processing, object classification.

A. Sensor Data Pre-processing

Data pre-processing refers to the process of extracting and "cleaning up" a subset of data from a much larger set of raw data. This is required as real-world data is often incomplete, inconsistent, unordered and noisy. In the context of this paper the input data is a point cloud: which is a list of unordered points described by rectangular coordinates ($N \times 3$ array, where N is the number of data points). The pre-processing required includes synchronization, ground detection, data alignment and ROI formation. Data synchronization ensures that data points between frames share the same global coordinate system, allowing for the coordinates of objects in different frames to be compared. Data clustering is required to isolate the objects in the scene from the environment (*i.e.*, object hypothesis generation) such that they can be classified.

B. Synchronization, Ground Detection and Alignment

The alignment of the point cloud data across multiple frames can be achieved by using a pre-existing algorithm that synchronises the frames based on the LiDAR odometry. The point cloud must be aligned with the global coordinate system to allow for manipulation of the data. The rotation of each frame can be achieved by identifying a specific 'reference-landmark' in the point cloud that can be used as a reference point. All the frames can then be rotated/translated in such a way that they align in a global coordinate system. In this work two distinct alignment processes are required, however both requiring the two-step process described below:

1) The identification and extraction of a common landmark *i.e.*, the ground.
2) Rotating/translating the dataset based on the common landmark.

The motion of the UAV causes the raw point cloud data output by the LiDAR to not be synchronised. The frame synchronisation is done by running the *loam_velodyne* SLAM package[1] [12], however this only uses the odometry data *i.e.*, it does not create a map. SLAM algorithms are generally used to generate maps by recursively adding to a single dataset but, in our case, only the localization aspect of the algorithm is used to synchronise the frames. Only once the data has been synchronised may it be fed into the main algorithm, which begins with ground detection. The data alignment process consists of determining a fixed reference, which can then be used to rotate the point cloud. Since all the objects are located on the ground, the ground plane would be an appropriate reference point in this case. Aligning the ground plane with a Cartesian frame $\{B\}$ (see Fig.5) allows for an easy top-down, front-back or side-on projection of the point cloud onto a plane, simplifying the occupancy grid mapping step. The rotation of a 3D dataset was achieved by multiplying it by a rotation matrix $R_{zyx}(\phi, \theta, \psi)$, where ψ, θ and ϕ correspond to the rotation angles yaw, pitch and roll, respectively.

The type of common reference-landmark determines the method by which to extract it, here the ground plane is the landmark and a 3D-plane is the primitive to be extracted. Therefore, we require an algorithm which can identify a set of points in the point cloud which lie in a common plane in 3D space. A plane in 3D space can be parametrized by two parameters, describing the spatial angle of the normal vector to the plane. RANSAC is the method implemented to determine a plane of best fit for a 3D dataset. So, the implementation of RANSAC returns the ground plane and its

[1] http://wiki.ros.org/loam_velodyne

corresponding normal. Using the normal and the coordinate x-axis, a rotation matrix is calculated. This rotation is then applied to all the remaining frames. This initial rotation ensures that the ground plane is parallel with the xy-plane, however, does not ensure the ground is set at a height of 0. This is done by translating the entire point cloud by the current mean offset to the ground. Note that this procedure is applicable as there is only a single ground plane (flat ground) in the field of view (FOV). The data collection was setup such that this was always the case. In cases where the ground may be uneven or have several levels this method of alignment would be inaccurate.

To finish the alignment the longitudinal axis of the point cloud must also be aligned with a coordinate axis (x-axis). This requires an additional rotation in the xy-plane, parallel to the ground plane. As before, this requires identifying a specific feature that is used as a reference to determine the longitudinal direction. A point cloud produces a symmetrical pattern on the ground (provided the UAV is not banked, pitch and roll angles are equal to 0). The axis of symmetry can be used to generate a line of best fit. Given the symmetrical nature of the ground plane, the symmetry axis can be calculated as the "line of best fit" of a linear data fit. This line of best fit is used to determine the rotational offset between the point cloud and the x-axis.

RANSAC [13] works very well for ground detection, succeeding in identifying the ground plane in all of the frames in the test dataset. Slight errors arise when there are relatively significant undulations in the ground, effectively splitting the ground into several linear elements. Occasionally this can lead to the detected ground plane not being exactly parallel with the true ground. However, these errors are insignificant in comparison to the unevenness of the ground and had no real impact on the result of the algorithm. However, the second rotation was more prone to error due to slight inconsistencies in the ground detection, creating an unsymmetrical shape which returned an "incorrect" line of best fit. As a result, the process for the second rotation was carried out twice, resulting in an average error of 0.45 degrees compared to the 4.6 degree error with only a single rotation.

C. Object hypothesis generation - clustering

The first step to classifying any objects that may be in the LiDAR's FOV is to identify the objects themselves. This process is known as clustering and consists of determining how many objects there are, the data points that belong to each object and the centroids of each object. Clustering is critical to the success of the algorithm as the ability to accurately cluster the point cloud will directly affect the accuracy of the classifier.

Prior to clustering, the point cloud must first be reduced by creating a region of interest (ROI). The computational cost of clustering scales with the number of data points to cluster, as such any reduction in the size of the point cloud that does not impact the clustering is beneficial. The generation of a ROI is dependent on the environment in which the UAV will operate. The data collection was designed to simplify the process of creating a ROI; all objects (human and non-human) are located on the ground plane and within a 10-meter radius of the UAV (constant altitude). In the data alignment process, RANSAC was used to determine the ground plane, and the data points belonging to it. The first step in creating the ROI is to remove the ground plane from the point cloud. This ensures that no data points associated with the ground are clustered as part of an object. The second step consists of trimming the remaining point cloud, removing unnecessary data points. This was done by creating a "box" around the UAV and removing all points that are outside of this boundary. The LiDAR odometry returns the sensors position and this is used as the centre of the "trimming box" (x and y positions only, the height of the trimming box begins from the ground). The dimensions of this box are $20 \times 10 \times 3$ m and do not change as the UAV maintains a constant altitude. This is only possible when the operating conditions of the LiDAR are known, which is unlikely in a real-world application.

In this paper, hierarchical clustering using Euclidean distance d_E is used to identify the number of clusters in the point cloud. Hierarchical clustering builds up a hierarchy of clusters which requires the space being clustered to be divided into discrete sub divisions. This is done by making use of a 3D grid subdivision. The clustering procedure is split in two parts. The first creates a ROI, the second clusters objects within this ROI. The only input parameter that can be adjusted is the d_E. Several values between 1-2 m were tested to determine which yielded the best results. The optimum value is dependent on the number of targets in the ROI, and the typical size of the objects and the typical distance between them. In cases where there is only a single target d_E must be large enough to group all data points into a single cluster. Assuming the ROI only contains data points of objects of interest (*i.e.*, the ground plane removal and trimming is successful), there is no upper limit for d_E for frames with a single target. In this case, any increase in this parameter will simply yield the same result; grouping all data points into a single cluster. However, the assumption of complete plane removal and trimming is not always valid, and occasionally parts of the ground remain within the ROI. This is due to undulations in the ground that cause some data points to be considered outliers of the ground plane. In these cases, a high d_E would group these outlying data points with the data points belonging to the object, an undesirable outcome. It was found that a d_E of 2 m was successful in 96% of cases where only 1 target was present. This would also ensure that the cluster did not group other data points that may still be within the ROI. This value is not appropriate for scenarios involving multiple targets, as it would often group them into a single cluster, nullifying the process of clustering.

In cases with multiple targets, adjusting only the d_E parameter is not sufficient to yield accurate results consistently. This is due to each individual target being split into multiple clusters at a d_E below 0.5. Yet at values above 0.5, multiple

Algorithm 1: Clustering for point cloud object hypothesis generation.

Result: List of point cloud clusters C
Imput: point cloud data P, where $p_i = [x, y, z] \in P$;
Create a kd-tree representation for P;
Setup an empty list of clusters C*, and a queue of points to be checked* Q;
for *every point* $p_i \in P$ **do**
  Add p_i to the current queue Q ;
  for *every point* $p_i \in Q$ **do**
    Search for the set P_i^k of point neighbours of p_i s.t. a sphere with radius r, d_E ;
    For every neighbour $p_i^k \in P_i^k$ check if the point has already been processed, else add it to Q ;
  end
  When the list of all points in Q has been processed, add Q to the list of clusters C, and $Q \leftarrow \varnothing$;
end
Return *The algorithm terminates when all point* $\in P$ *have been processed*;

targets begin to be grouped together if they are in close proximity to each other. This effect is amplified by the relatively low resolution of the LiDAR[2], as each target only consists of two or three of the LiDARs channels. Due to the quite large gaps between the channels, data points belonging to target 'A' may actually be closer to some data points belonging to target 'B'. This would result in some of these data points being grouped into the incorrect cluster. However, if d_E is reduced below 0.5 m, the individual targets begin to be clustered as multiple objects making it impossible to dynamically determine which clusters belong to which target. The clustering was successful only 57% of the times when the targets were within 1 m of each other. The steps of the algorithm have been summarised in the Algorithm 1.

D. Environment representation using Occ.Grid mapping

The clustering algorithm returns a $N \times 3$D array containing all the data points belonging to a cluster. The cluster classification can be performed using, basically, two approaches: (i) by inputting the array containing all the cluster data points as an input to a 3D-model *e.g.*, PointNet [14]; (ii) representing the 3D cluster as a series of 2D projections and inputting those 2D "images" into a traditional CNN. The first approach retains all of the features within the data which theoretically would result in more accurate results. However, this approach requires more post processing to be done on the clusters. More precisely, each cluster would need to be reduced/enlarged such that all clusters contain the same number of data points and hence can be input into the same CNN, as the input layer of the CNN must remain constant. The increased complexity, the potential that the dataset may be too small to obtain reasonable results and the difficulty on 3D-points annotations, were all factors which led us to select the second approach.

The reduction of a 3D space to a 2D space can simply be done by projecting the data points onto a given plane as shown in Fig. 3 which creates a silhouette of the object on

[2]The dataset was acquired using a Velodyne VLP with 16 channels.

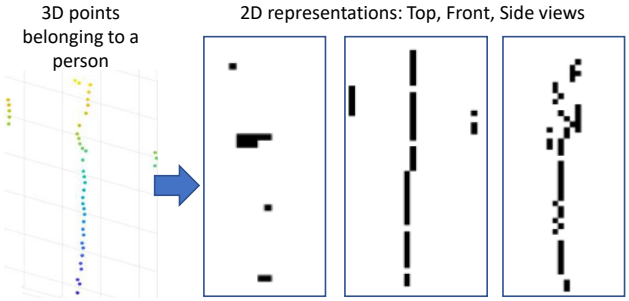

Fig. 3. 3D-LiDAR cluster of person and the multi-view 2D representation: Top (T), Front (F), Side (S).

the plane. The three Cartesian planes have been chosen as the 3 planes for projection. They are the *xy*-plane (top/down view), the *xz*-plane (front view) and the *zy*-plane (side view). Instead of a direct projection, an occupancy grid mapping [15] process was used to create a lower-resolution projection image, in order to reduce the number of input variables and thus the complexity of the following CNN algorithm. This is a process in which the projection target plane is broken down into a number of discrete cells according to the desired accuracy. The clusters are placed in the centre of the grid map by aligning the cluster centroid with the grid map centre. The data points are then projected onto the grid. The cells of the grid are then assigned a value depending on the properties of the points projected into the grid cell, which when visualised corresponds to a shade of grey. In the simplest case of binary grids, each cell containing at least one data point is considered "occupied" (black) and all other cells are considered empty (white). Such grids are generally used to create a top/down map of a robot's environment, required for navigation. However, in the context of this study the grid was used to generate an image which could then be passed onto the classifier.

There are two types of 2D occupancy grids: binary and probability grids [16]. Probability grids assign a probability (percentage) of a given cell to be occupied. These grids convert to greyscale images (0 = white, 1 = black, shades of grey in-between). Binary grids assign a single value to each cell, either occupied or empty. Initially a probability grid was used, where the probability value assigned to a cell corresponds to the normalized height of the data points assigned to the grid cell. Normalization here means that 0 = lowest "height" in cluster, 1 = highest "height" in cluster. The term "height" refers to the axis perpendicular to the projected plane. However, it was found that in terms of the classifier accuracy, there was no difference between a binary or probability grid. Therefore, only binary grids were used when comparing the various CNNs. In the creation of grid maps, there are two parameters than can be changed; the size of the grid and the resolution (size of the cells). The grid maps used here were all of equal overall size 3×3 meters. The grid resolution refers to the number of cells per meter, which determines the size of each cell.

TABLE I

NUMBER OF EXAMPLES PER VIEW (T,S,F AND TSF), IN TOTAL (*TOT*) AND IN EACH SUBSET (*Sub*). PERCENTAGES OF THE TRAINING, TEST AND VALIDATION SETS FOR THE CNN-MODEL

		Single View	Object	Human	Train. Set [%]	Valid. Set [%]	Test Set [%]
Augmented	*TOT*	3124	1612	1512	60	15	25
Data	*Sub*	806	428	378	-	-	-
Non-Augmented	*TOT*	510	225	285	60	15	25
Data	*Sub*	12t	56	71	-	-	-

TABLE II

CNN ARCHITECTURE DESCRIPTION.

N#	Layer	Description
1	Image Input	$90 \times 90 \times (1/3)$, 'zerocentre' normalization
2	Convolution	83×3 conv., stride [1 1], padd. [1 1]
3	Batch norm.	Batch normalization
4	ReLU	ReLU
5	Max Pooling	2×2 max pool., stride [2 2], pad. [0 0 0 0]
6	Convolution	163×3 conv., stride [1 1], padding 'same'
7	Batch norm.	Batch normalization
8	ReLU	ReLU
9	Max Pooling	2×2 max pool., stride [2 2], pad. [0 0 0 0]
10	Convolution	323×3 conv., stride [1 1], padding 'same'
11	Batch norm.	Batch normalization
12	ReLU	ReLU
13	Full connected	2 fully connected layer
14	Softmax	Softmax
15	Output	Classification w/ CrossEntropy

E. CNN-based classification

A CNN was trained using the same dataset with 3 different grid resolutions using 10, 25 and 50 cells, which corresponds to a cell side length, respectively of 10 cm, 4 cm and 2 cm. The front projection for the same cluster in 3 resolutions is shown in Figs. 4. Each CNN was tested on the same test dataset (combination of positives and negatives) according to Table I. The comparisons amongst the different CNNs - discussed in the next section - have all been trained on images produced with a grid resolution of 25 cells, as it has produced the best results.

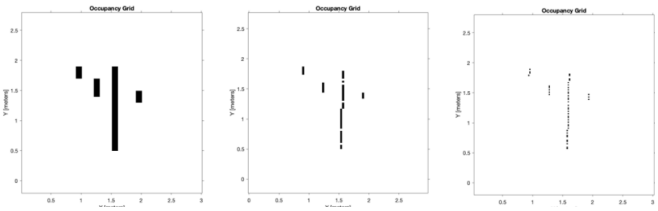

Fig. 4. 2D representations, using grid mapping technique, with resolutions of 10, 25 and 50 cells respectively.

III. EXPERIMENTS AND CLASSIFICATION RESULTS

UAV mounted LiARs have only recently gained traction and as such there has not been a large amount of research done in this field. As a result, there are no publicly available datasets, requiring therefore to collect our own.

In order to mimic the real-world situations that the airborne LiDAR may encounter, a total of 30 scenarios were created. The scenarios included a mix of humans (positives) and objects (negatives) as well as single target and multi target situations. Due to the narrow and long nature of the point cloud (shown in Fig. 5), the shape of a person may vary slightly whether they enter the LiDAR field of view from a perpendicular or parallel direction w.r.t. the longitudinal axis of the point cloud. Hence each scenario was carried out at both orientations. The scenarios includes 5 adults (4 male, 1 female) to make up the positive class and a total of 10 unique objects to make up the negative class.

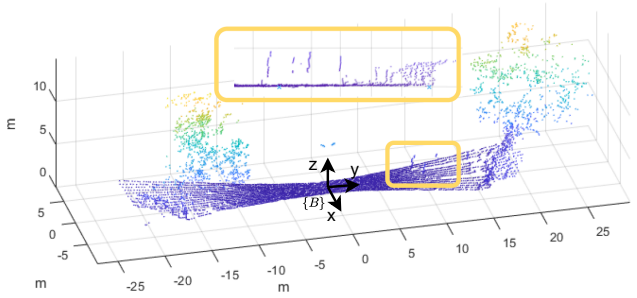

Fig. 5. A point cloud frame in the $\{B\}$ coordinate system, where it is possible to visualize some people.

As samples for the negative class, objects were chosen that are typically found outside, specifically in urban environments in which this type of technology is likely to be used. Due to the limited number of total objects, it was ensured that the chosen objects were of different shapes and sizes to maximise the diversity of the dataset. The positive class is limited to examples of adults, who are all of similar height (between 160-185 cm). A point cloud does not have colour or texture and hence the only difference it can pick up between different people would be their size (height/shape). Table I summarizes the dataset information.

A. CNN-based classification

The classification of the grid maps is performed using a deep-CNN model. The grid maps (hereafter called images) were generated slightly oversize to 90×90 cells on purpose, to ensure that all data from each cluster was covered completely as the algorithm used fails if any data point lies outside the grid area. However, this results in a large amount of empty space around the majority of objects, increasing the size of the image and hence the required number of nodes in the neural network.

The dataset contains three sets of images: top (T), front (F) and side (S) projections, where each one represents a channel. Multiple CNNs were designed and their accuracies compared to determine which of the three projections yields the best result. The amount of information within a single projection is limited, so the images were stitched together to form one (3 channel) image (TFS). To minimize the chance of *overfitting* the data was augmented. The adjustments can be achieved in many ways *e.g.*, the images can be cropped, rotated, mirrored, rotated or translated. In our case, cropping or translating the image is not particularly useful. This is because the algorithm dynamically generates the grid maps

with the same method, resulting in the image to be classified to always have the same size with the object in the centre of the grid map.

The chosen method to augment the data was to apply small rotation steps about all three axes. Each cluster was rotated about its centroid over a range of a given degree steps. The augmentation resulted in the 225 (objects/negatives) and 285 (humans) unique grid maps being converted into a total of 1,612 and 1,512 images. Although still relatively limited, this is enough data to train the neural network. The input layer must have the same number of nodes as the number of pixels/cells in the image. In this case the input layer is $[90 \times 90 \times 3]$ for multi-channel (concatenated views) and $[90 \times 90 \times 1]$ for the single images. The number of output layers on the network is equal to the number of classes; in this case the classifier simply differentiates humans from all other objects, requiring a 2-node output layer. The specific parameters of the CNN models/architecture, the layers and hyper-parameters are given in Table II.

B. Results and discussion

When training and verifying the learning models, the total dataset must be split into a training set and testing set. It is crucial that the examples in the testing set are different to those in the training set (for a fair test, the network should not have seen the examples before). This can be done by simply splitting the set based on a percentage (70% training, 30% testing), or by using a *cross validation* technique as in this paper. The cross-validation approach splits the total dataset into equally sized parts; in this case the data was divided into 4 subsets. 4 CNN-models (same architecture) are then trained on the 4 combinations of training/testing data available. For example; CNN subset 1 trains on subsets 2,3,4 and tests on subset 1, CNN subset 2 trains on subsets 1,3,4 and tests on subset 2 and so on. As stated earlier, it is important that the test set is unique from the training sets, including augmented images thus, the augmented images of each parent image were assigned to the same subset.

A CNN was trained on each of the four image types (T, F, S and TFS) with the non-augmented and augmented training set. The testing accuracy of the 4 CNNs (for each subset combination) within each image type are averaged to give a more accurate result. The full results are provided in Table III. Except for the Side-view, the accuracy for the Augmented model is, on average, significantly higher than those for the non-augmented model; specially for the Top-view. As generally, more training data is expected to improve the performance of a CNN. In terms of AUC (calculated from the ROC curves shown in Figs 6 and 7), the performance obtained with the models are to be read with caution as the testing set is not large enough to have captured all the possible variations of real-world scenarios. Notice that all the CNN models were tested on non-augmented testing set.

Although the algorithm operates successfully, there are improvements that can be made to enhance its robustness and speed. Clustering has the highest failure rate amongst the components in the algorithm and hence this sub-algorithm

TABLE III

PERFORMANCE MEASURES, IN %, FOR EACH "VIEW" ON THE AUGMENTED AND NON AUGMENTED MODELS.

	Augmented CNN				Non-Augmented CNN			
View:	S	F	T	TFS	S	F	T	TFS
Accuracy	95.5	97.1	78.2	98.8	96.5	97.8	69.6	95.8
AUC	99.2	99.7	91.0	99.9	99.5	99.7	83.9	99.7
Precision	92.3	96.1	87.0	98.4	95.4	97.1	97.1	99.4
Recall	99.6	98.6	79.8	99.5	98.2	98.9	66.9	93.7
F-score	95.7	97.3	81.5	98.9	96.7	98.0	78.68	96.4

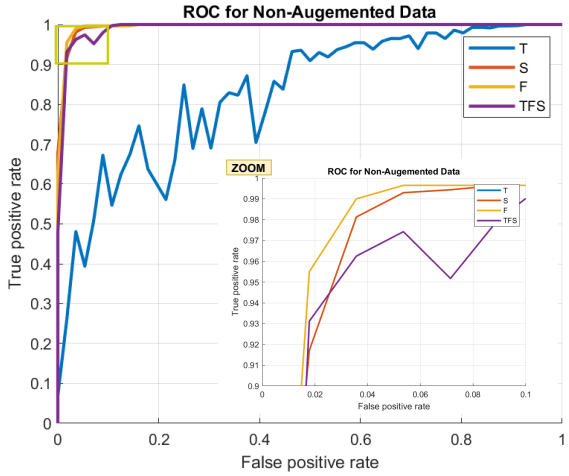

Fig. 6. ROC curves for the CNN models trained on the non-augmented dataset.

should be revised or replaced by an alternative. Improved clustering would also have positive knock-on effects on the classification. The current algorithm assumes that the entire ROI has the same "flat" ground plane, however this is not representative of real environments, where the ground is often uneven and multileveled. A new data alignment strategy would need to be implemented to work effectively in these cases. Depending on the application and required accuracy, an improved (high resolution) sensor could be used which should yield better results.

The classification/detection components of the algorithm can be improved by linking them through a feedback loop. This is especially necessary when tracking multiple targets in close proximity, where the clustering or classification components may fail individually. The estimated position of an object can be used to aid in the clustering and classifying of the object. The number of output classes would have to be increased significantly for the proposed approach to be applicable in more challenging situations. This would require obtaining a significantly larger dataset, especially for non-human objects (negatives). To increase the robustness, the dataset should include a variety of environments with significantly different weather and lighting conditions.

IV. CONCLUSION AND FUTURE WORK

This is a unique research topic and seems to be promising because the cost and size of LIDAR technology is being cur-

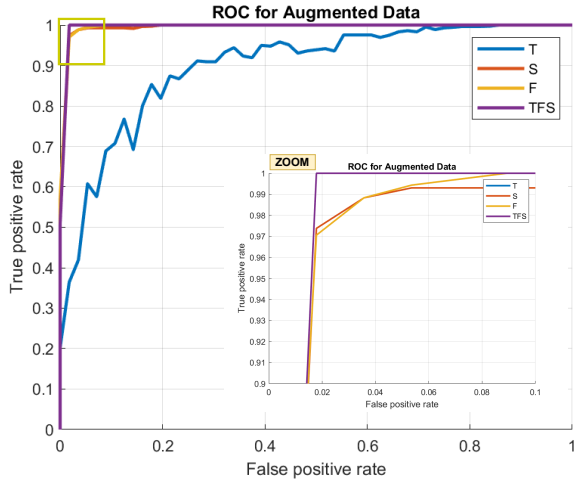

Fig. 7. ROC curves for the augmented dataset.

rently seeing a large reduction whilst keeping the resolution high. On the other hand, with the increasing capacity of deep learning approaches to extract rich and useful information from 3D LiDAR data, new applications will foster the use of airborne LiDAR sensor in small drones.

This study is focused on the design of algorithms that output the classification of a target (human vs non-humans). Data from a UAV mounted 3D LiDAR was collected in real-world conditions and a binary classification dataset was built. A pipeline comprising pre and post-processing stages and object classification are proposed. Direct and augmented CNN models were trained, tested and compared according to a cross-validation strategy. Detailed experiments are described and the results are promising.

In real-world applications, UAVs require a navigation system that takes the targets location as an input and outputs the vehicles required actions. Only with such a system could allow a 3D LiDAR-equipped UAV autonomously track and follow a person; this could be considered in the future work. Alternatively, this algorithm could be adapted for use on other vehicles or in alternative environments. Finally, a more sophisticated approach like Complex-YOLO [17] could be explored as well.

V. Acknowledgments

This paper is based on the first author's dissertation [11]. This work has been supported by the project MATIS - CENTRO-01-0145-FEDER-000014, Portugal, and partially supported by FCT through grant UID/EEA/00048/2019.

References

[1] Z. Yan, T. Duckett, and N. Bellotto, "Online learning for human classification in 3D LiDAR-based tracking," in *IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 864–871.

[2] A. Rangesh and M. M. Trivedi, "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 4, 2019.

[3] C. Premebida, O. Ludwig, and U. Nunes, "Lidar and vision-based pedestrian detection system," *Journal of Field Robotics*, vol. 26, no. 9, pp. 696–711, 2009.

[4] L. Wallace, A. Lucieer, and C. S. Watson, "Evaluating tree detection and segmentation routines on very high resolution uav lidar data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 12, Dec 2014.

[5] F. Azevedo, A. Dias, J. Almeida, A. Oliveira, A. Ferreira, T. Santos, A. Martins, and E. Silva, "Lidar-based real-time detection and modeling of power lines for unmanned aerial vehicles," *Sensors*, vol. 19, no. 8, 2019.

[6] M. Tulldahl, H. Larsson, G. Tolt, F. Bissmarck, C. Gronwall, and J. Nordlof, "Application and capabilities of LiDAR from small UAV," in *Laser Radar Technology and Applications XXI*, M. D. Turner and G. W. Kamerman, Eds., vol. 9832, International Society for Optics and Photonics. SPIE, 2016.

[7] M. U. de Haag, C. G. Bartone, and M. S. Braasch, "Flight-test evaluation of small form-factor LiDAR and radar sensors for sUAS detect-and-avoid applications," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sep. 2016.

[8] L. Zheng, P. Zhang, J. Tan, and F. Li, "The obstacle detection method of uav based on 2D lidar," *IEEE Access*, vol. 7, 2019.

[9] G. Omans, "Phoenix LIDAR systems overview," 2018. [Online]. Available: https://www.phoenixlidar.com/team/#phoenix-overview/

[10] A. Hussey and Velodyne, "Velodyne LiDAR partners with YellowScan for integrated LiDAR for UAVs," 2017. [Online]. Available: https://velodynelidar.com/press-release/velodyne-lidar-partners-with-yellowscan-for-integrated-lidar-for-uavs/

[11] J. N. C. Hayton, "Human identification and tracking using an airborne lidar," Master dissertation, Loughborough University, 2019.

[12] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in *Proceedings of the RSS Conference*, 2014.

[13] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, 1981.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[15] A. Elfes, "Occupancy grids: A stochastic spatial representation for active robot perception," *ArXiv*, vol. abs/1304.1098, 2013.

[16] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[17] M. Simony, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-YOLO: An Euler-region-proposal for real-time 3D object detection on point clouds," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.