2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

AirFisheye Dataset: A Multi-Model Fisheye Dataset for UAV Applications

Pravin Kumar Jaisawal[1], Stephanos Papakonstantinou[1] and Volker Gollnick[1]

***Abstract*— Drone applications require perception all around the vehicle to avoid obstacles during navigation. Due to the weight and computation limitations on UAVs, using a large number of sensors, such as numerous cameras, could be prohibitive. In such scenarios, usage of fisheye cameras with a wider field of view is very beneficial. Despite the usefulness of fisheye camera for UAV applications, not much work has been carried out to develop perception algorithms for fisheye camera. One of the main problems being the lack of publicly available omnidirectional datasets in relation to drone flight. With this paper, we address this gap by presenting AirFisheye dataset, which is applicable for tasks such as segmentation, depth estimation and depth completion, among other tasks required for autonomous drone navigation. Also, a generic framework for creating synthetic fisheye images is provided. Furthermore, we propose a novel occlusion correction algorithm that removes incorrectly projected LiDAR point clouds into the camera image due to the viewpoint variation of both sensors. We release about 26K images and LiDAR scans along with annotations. Baseline code and supporting scripts are available at `https://collaborating.tuhh.de/ilt/airfisheye-dataset`**

I. INTRODUCTION

In the field of autonomous driving or flying vehicles, high-performance and efficient algorithms are required for perception tasks, such as the detection of obstacles, *e.g.* drones [1] and navigation tasks. Such complex algorithms are usually based on deep learning, which requires a large amount of data for better performance.

Availability of public datasets, *e.g.* KITTI dataset [2], Cityscapes dataset [3], ApolloScape dataset [4], *etc.* has pushed the development of various state-of-the-art algorithms for autonomous driving. Similarly, for autonomous navigation in Unmanned Aerial Vehicles (UAVs), developing such perception and navigation algorithms require datasets that comprise several sensor data such as camera, LiDAR, Inertial Measurement Units (IMUs), RADARs, *etc.* Recently, few datasets such as Mid-Air [5], TartanAir [6] were introduced to encourage the development of such algorithms for drone applications. However, all of these publicly available datasets use pinhole camera as the main sensor, and no data are available for the fisheye camera.

Fisheye cameras perceive the environment with a field of view (FOV) that is greater than the FOV of normal perspective cameras. This characteristic of fisheye cameras is particularly useful for UAV applications, where there are weight and computational limitations. However, most of the available fisheye multi-modal datasets are collected in urban environments for autonomous driving applications. A challenge in these available datasets is that the data distribution is often constrained due to fixed sensor locations in the body frame of the ego vehicle. In contrast to autonomous cars, sensors such as cameras and LiDAR are mounted on the gimbal of the UAV, which can be used to change the orientation of these sensors. Hence, models trained in ground based autonomous driving datasets may not generalize for drone applications.

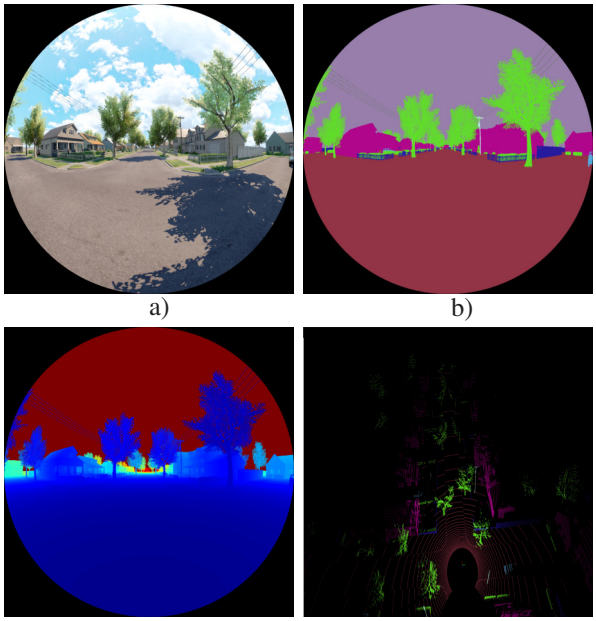

Fig. 1. Sample sensor data collected for AirFisheye Dataset, a) Fisheye RGB image b) Segmentation label c) Depth map d) LiDAR scan with segmentation information (top view)

Data collection in the real world requires careful sensor setup and calibration. One of the key challenges in generating a dataset for the collected real world data is to obtain pixelwise semantics and depth labels. Generation of such dense labels is an expensive and time-intensive effort. For instance, according to [7], the generation of a single pixel-wise labelling by a human annotator takes up to 45-60 minutes. Moreover, these (manually) generated labels suffer from inaccuracies or incompleteness, *e.g.* in the case of generation of depth labels.

We address these shortcomings by recording data using a drone from the photo-realistic environment rendered using the Unreal Engine [8]. The drone is flown at different altitudes (0-65m) with different gimbal orientations (0-45°) to

*This research was supported by the Hamburgische Investitions- und Förderbank (IFB Hamburg), grant number: T-ZW-M28-IFB-2001

[1]Institute of Air Transportation System, Technical University of Hamburg, 21073 Hamburg, Germany
Primary contact: `pravin.jaisawal@tuhh.de`

TABLE I

SUMMARY OF WIDELY USED DATASETS WITH EITHER FISHEYE CAMERA OR (DATASET FROM) DRONE FLIGHT

Dataset	**AirFisheye (Proposed)**	Mid-Air [5]	EuroC MAV [13]	TartanAir [6]	WoodScape [14]	KITTI-360 [15]	SynWoodScape [16]
Resolution	1536 × 1536	1024 × 1024	752 × 480	640 x 480	1280 × 966	1400 × 1400	1280 × 966
Real/Synthetic	Synthetic	Synthetic	Real	Synthetic	Real	Synthetic	Synthetic
Camera motion	**drone flight**	drone flight	drone flight	drone flight	Car drive	Car drive	Car drive
Camera Type	**Fisheye**	Perspective	Perspective	Perspective	Fisheye	Fisheye	Fisheye
Fisheye FOV	190°	N/A	N/A	N/A	190°	180°	190°
Semantic Seg.	✓	✓	✗	✓	✓	✓	✓
Depth map	Dense	Dense	Sparse	Dense	Sparse	Dense	Dense
LiDAR	✓	✗	✗	✓	✓	✓	✓
IMU	✓	✓	✓	✓	✓	✓	✓

simulate viewpoint variations experienced during flight. The gimbal orientations are chosen such that the drone observes most of the ground scene. Drone control and sensor data collection are carried out using the AirSim plugin [9]. Since there is no support for fisheye camera in Unreal Engine, a framework proposed by [10]–[12] is used for the generation of fisheye images. As the generation of fisheye from cubemap is very generic, it can also be used for other applications.

We release about 26K images along with dense semantics labels and depth labels, along with synchronized 64-channel LiDAR scans (including semantic information). These datasets are recorded with some lighting variations; however, data are only collected for good weather conditions at the moment. We intend to progressively augment these available data with varying climatic conditions and dynamic objects. Four photo-realistic environments are used for the overall data collection process to diversify the data.

The main contributions of this paper are (1) a framework to record the dataset and generate fisheye images, (2) creation of a synthetic dataset containing more than 26K frames, along with synchronized LiDAR scans, (3) an algorithm to remove occluded LiDAR point clouds for LiDAR to camera image mapping, and (4) baseline results and metrics on tasks such as semantic segmentation, depth estimation and depth completion.

II. RELATED WORKS

Datasets play an essential role in encouraging the development of novel, cutting-edge algorithms for autonomous vehicle applications. There are several datasets available in the literature that either use a fisheye camera or use drone as its main actor. An overview of these publicly available datasets is provided in Table I.

In regards to the aerial dataset, the Mid-Air [5] dataset proposes a synthetic dataset for two unstructured environments containing camera data and their dense labels. Despite those facts, a perspective camera is used as the main sensor and does not include LiDAR data. EuroC MAV [13] dataset provides a real world aerial dataset with LiDAR and RGB images, yet it also uses a perspective camera. Moreover, dense pixel-wise labels for tasks, such as segmentation or depth estimations, are either unavailable or incomplete. The most complete dataset for aerial applications is introduced in TartanAir [6] dataset with camera and LiDAR scans (although synthetically generated using depth image). The main drawback of this dataset is the use of a perspective camera. Recently, datasets such as CarPK [17], UAVDT [18], FishEye8K [19], *etc.*, have been introduced. Nevertheless, the main focus of these datasets lies on object detection from aerial images and lacks dense labels.

If the available dataset contains fisheye images, these mostly come from car drives, such as WoodScape [14], KITTI-360 [15], SynWoodScape [16], *etc.* Most implementations fix the sensor setup to the car or vehicle, unlike in drone applications, where sensors are typically mounted on a gimbal. Using such datasets for developing perception or control algorithms has the main drawback of not being able to generalize well for aerial applications. The main gap in this regard is the unavailability of fisheye camera dataset, particularly for drone applications. This gap motivates the creation of a fisheye dataset for the development of algorithms for drone applications, *e.g.* depth estimation, depth completion, segmentation, SLAM, *etc.*

Sensor fusion between camera and LiDAR data generally requires mapping of LiDAR data into camera image. One of the main problems in direct mapping from LiDAR into a camera image is that some of the point clouds which are not visible in the camera frame also get mapped. Such incorrectly mapped point clouds create problems for the perception module. There are several occlusion correction approaches available in the literature. Common approaches involve either a segmentation approach with 3D clustering [20] or a distance based segmentation approach with a morphological filter [21]. In both of these approaches, a series of temporary images are created based on distance. In [22], a mask is generated for every projected point cloud and is used to prevent other points from being placed in the mask area of the image. The main drawbacks of these approaches include being more memory intensive due to the usage of multiple images and having a higher dependency on image properties. Additionally, using fixed size masks, as in [22], can lead to the removal of valid point clouds.

III. AIRFISHEYE DATASET

The AirFisheye dataset is a synthetic dataset containing fisheye images along with LiDAR scans for the perception and navigation of the drone in the structured/unstructured environment. It contains about 26K images captured from

Fig. 2. Samples of generated AirFisheye Dataset showing images from different environments with different gimbal orientations

four photorealistic environments, namely *Modular Neighborhood Pack* [23], *Modular House* [24], *Modular Building Set* [25], and *Downtown West Modular Pack* [26]. These environments mostly represent structured urban settings with some natural unstructured scenes. These environments are modified slightly for correct stencil allocation. A few assets, such as vehicles, *e.g.* cars, lorries, vans, *etc.*, were also added to these environments.

Sunny weather and daylight are chosen as the lighting conditions for recording the data. The brightness of the light and shadow intensity in the simulator varies based on the location of the sun or the light source. As fisheye images are created by combining images from 5 cameras, post-processing steps, such as vignette, bloom, and lens flare in the simulator are turned off because post processing steps can cause discrepancies in pixel intensity, particularly at the edges where images from two neighbouring faces of the cubemap lie.

The second column of Table I summarizes multi-model sensor data and their corresponding ground truth provided by the proposed dataset. The dataset is collected during drone flight with different altitudes (up to 65m) and changes in orientation (up to 45°). Fig. 2 shows samples of the dataset taken at these configurations, *e.g.* images in the last column in this figure represent images taken at altitudes of 30-65m and at 15°-45° gimbal orientation.

Care has been taken to ensure diversity in our dataset in terms of variation of environments, lighting conditions, and viewpoint variation. Our dataset is decently large. Such characteristics of the dataset are beneficial for the generalizability of trained models.

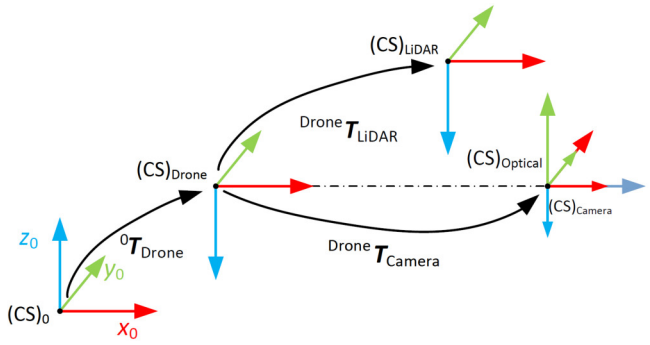

Fig. 3. Coordinate systems and transformations of the recording platform

A. Dataset acquisition

Fig. 3 illustrates the coordinate frames of the sensor setup. To record the data, drone flight paths were recorded by manually flying the drone in the simulator. In total, 12 trajectories are saved. Subsequently, these saved trajectories are utilized for recording data in the second step.

Dataset acquisitions are carried out at 10 frames per second. During each data capture, 5 RGB images, 5 segmentation images, 5 depth images, each of size 1024×1024 along with 64 channel LiDAR scans with 270° FOV LiDAR are taken. Later, these images are used for creating fisheye images. In addition to these data, IMU data, position information of the camera and LiDAR are also recorded. The positional information of the camera and LiDAR are used to remove the synchronization problem of both sensors arising from motion.

B. Fisheye image from cubemap

A cubemap generally uses 6 faces of a cube where the environment is mapped with a 90° view frustum. However, for our work, only 5 images are captured using 5 cameras whose origin lie at the same location but are pointing in different directions, for instance, the positive and negative directions of X, Y and Z coordinates, in the simulator. Back camera (negative Z) is omitted. These captured images are assembled in a particular fashion (see Fig. 4-a) to create a cubemap.

To create a fisheye image, a unit sphere needs to be created based on the desired output fisheye image dimension. The pixel coordinates of an image are initially converted to normalized coordinates (-1 to +1), which are later transformed to polar coordinates (r-ϕ). Then, the value of the radial distance r is mapped linearly to FOV/2 of the fisheye camera, which is used to represent the polar angle of unit sphere θ. Finally, using polar angle θ and azimuthal angle ϕ, a section of the unit sphere can be created. For each point on the sphere represented by a pixel of the output fisheye camera, ray tracing is performed to compute the mapping into cubemap (see Fig. 4-b). A lookup table is created to store the corresponding cubemap pixel for each output fisheye pixel and is used to speed up the mapping process for the remaining images.

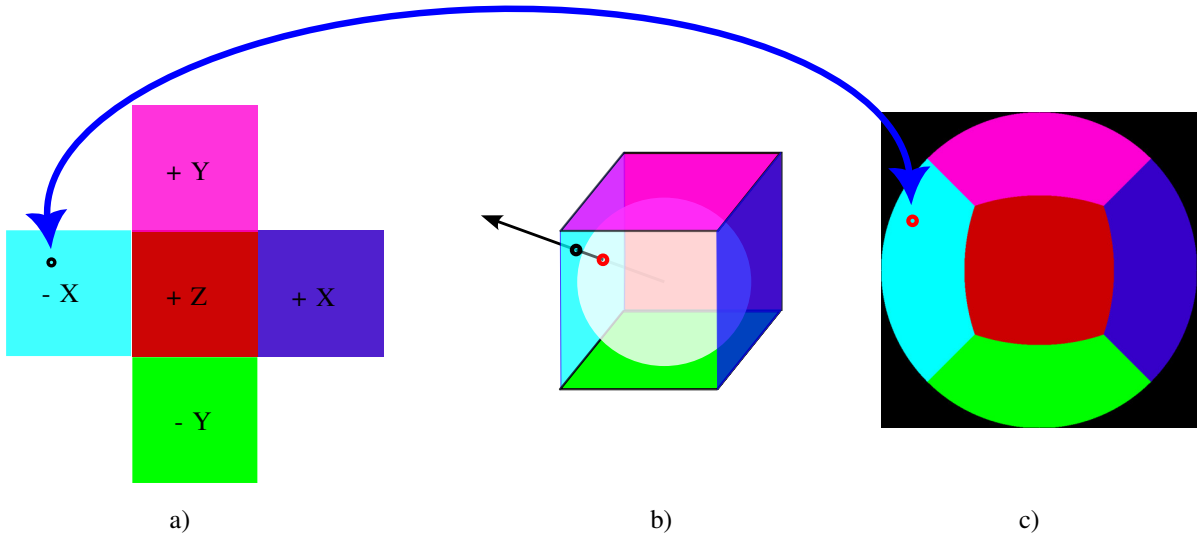

Fig. 4. Generation of fisheye image from cubemap image, a) Cubemap image from 5 cameras b) Mapping process *e.g* black pixel from cubemap image is mapped into red pixel in fisheye image c) Output fisheye image

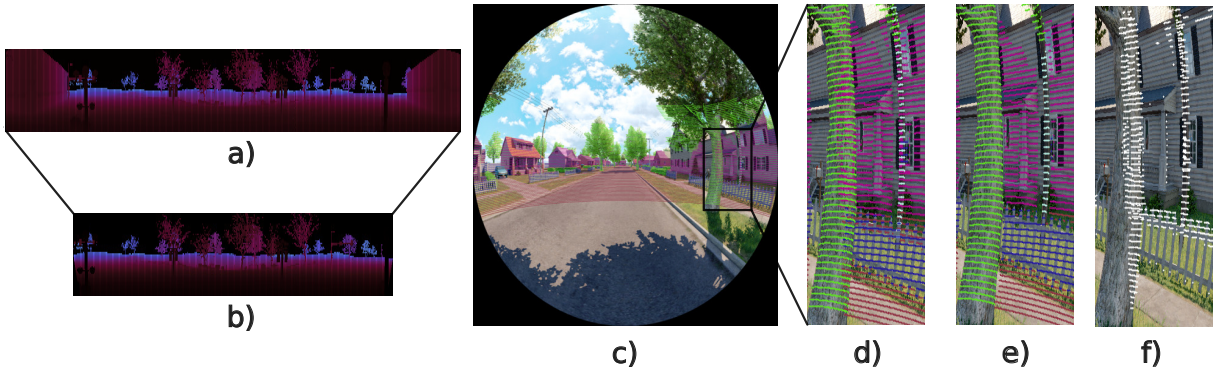

Fig. 5. a) Distance based color mapped image by arranging point cloud based on I_{LiDAR} b) Distance based color mapped image by arranging point cloud based on I_{proj} c) LiDAR point cloud mapped into fisheye image without occlusion removal d) enlarged section from Fig. 5-c to illustrate problem of occlusion e) Fig. 5-d after occlusion correction f) occluded point cloud from Fig. 5-d

C. Occlusion correction algorithm

Different viewpoint locations of the camera and LiDAR is the main reason for occlusion when LiDAR point clouds are mapped into a camera image. As mentioned in section II, detecting and correcting occlusion is crucial for accurately perceiving the environment. Our approach to solving occlusion is based on the observation that the order of point clouds changes when occlusion occurs. Unlike existing methods in the literature, occlusion detection and correction are performed directly on the point cloud. The main trick of performing occlusion correction directly on the point cloud is based on our second observation that the rotation of point clouds does not cause occlusion. The process of occlusion correction is described in detail in the subsequent paragraphs of this section.

The first step involves establishing the initial order of point clouds in the LiDAR frame. To achieve this, the points in the LiDAR frame, P_{LiDAR}, are represented in a spherical coordinate system (r, θ, ϕ). Subsequently, using (θ, ϕ) and the LiDAR specification, such as vertical FOV (VFOV), horizontal FOV (HFOV), number of channels/rings (N_C), and number of points per channel (N_{PC}), P_{LiDAR} is organized into a matrix format ($N_C \times N_{PC}$). This initial arrangement (order) is referred to as I_{LiDAR}. Each entry of I_{LiDAR} contains index of each LiDAR point in P_{LiDAR}. In this context, the term "order" refers to some sort of arrangement principle, *e.g.* ascending order of θ when one moves from left to right within each row of I_{LiDAR} and similarly, ascending order of ϕ along each column, when one moves from channel, $c_i = 0$ to channel, $c_i = N_C$. When we replace each index of I_{LiDAR} with corresponding distance (r) value of LiDAR point, we get Fig. 5-a.

Since the FOV of the fisheye camera is smaller than the FOV of LiDAR, a few LiDAR points will not be mapped into the image. Such unmappable point clouds should be removed from P_{LiDAR} to save computation. To achieve this, P_{LiDAR} is projected into a camera image to extract indices of all projectable points. Based on this step, a sub-matrix of I_{LiDAR} containing indices of only projectable point clouds is extracted. We refer to this matrix as I_{proj} and the projectable

point cloud as P_{proj}. Fig. 5-b represents a section of Fig. 5-a, with only point clouds from P_{proj}. Now, using our second observation, each point cloud in P_{proj} is translated to the camera frame. Following this, the cartesian coordinates of these translated point clouds are converted into spherical coordinates $(\tilde{r}, \tilde{\theta}, \tilde{\phi})$. We also create a distance matrix (R_{cam}) by replacing each index in I_{proj} with distance value $(\tilde{r})$ of corresponding LiDAR point from the camera frame.

In the third step, changes in order are to be determined. To accomplish this, two new matrices (θ_{final} and ϕ_{final}) of the same size as I_{proj} are created. These matrices are initialized with NaN and will be used to fill it with θ and ϕ values of each non-occluded point in similar order as that of I_{proj}. Starting from a point cloud with the smallest distance in R_{cam}, its index location (i, j) in I_{proj} is extracted. Since the first entry of LiDAR point cannot be occluded, the $\tilde{\theta}$ and $\tilde{\phi}$ value of this LiDAR point is added at the same index location (i, j) in θ_{final} and in ϕ_{final} matrices, respectively. This distance value is then removed from R_{cam}. For other points, point cloud with the next smallest distance is extracted, and it is used to retrieve index (i, j) value from I_{proj}. Since every non occluded point cloud preserves the order, we check if this point cloud satisfies two conditions: 1) $\tilde{\theta}$ of this point cloud should be greater than every θ lying to the left, *i.e.*, [0, j) and similarly smaller than every θ lying to the right, *i.e.*, (j, end] in i^{th} row of $theta_{final}$ matrix 2) $\tilde{\phi}$ of this point cloud should be greater than every entry below i^{th} row within LiDAR vertical resolution (VFOV$/(N_C - 1)$) and similarly smaller than every entry lying above i^{th} row in ϕ_{final}. Subsequently, this distance value from R_{cam} is removed.

For other minor details, please check our implementation on our GitLab page.

D. Data splitting

In order to reduce the dataset bias, collected data is carefully split into train, validation and test data based on the location of the environment where the data was captured. Overall, the ratio of train:val:test is 65:19:16.

IV. BENCHMARK

In this section, experiments and metrics used for three tasks, namely semantic segmentation, depth estimation, and depth completion, are briefly described. The images are downsized to 512×512 for training and evaluation. All experiments are conducted in the *PyTorch* framework and are trained on a single 32GB V100 GPU. Baseline codes for these experiments are provided on our GitLab page.

A. Semantic Segmentation

Our dataset contains pixel-wise segmentation labels for 23 object categories. Some of the major categories are sky, ground (where drones can land), houses, and vegetation (bushes, trees). Invalid pixels lying on the boundary of a fisheye image are considered as unlabelled pixels. Apart from these categories, the remaining categories comprise around 2% of the total pixels. We merge a few smaller categories for simplification. In total, 13 training categories were created. The proportions of each category are shown in Fig. 6.

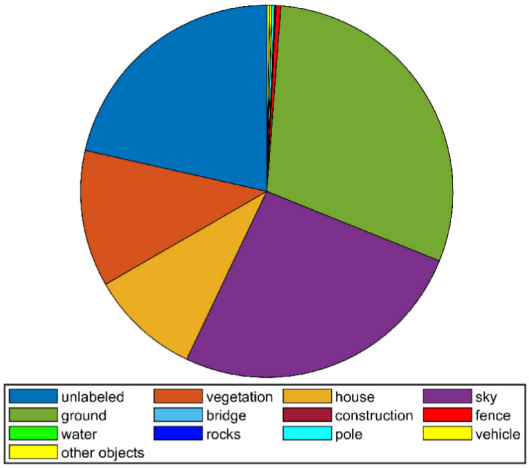

Fig. 6. Objects categories used for training

TABLE II
SUMMARY OF BASELINE RESULTS FOR DEPTH TASKS ON TEST DATA

Monocular Depth Estimation

Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
0.132	2.575	13.324	0.270	0.865	0.950	0.972

Depth Completion

Abs Rel ↓	RMSE ↓	MAE ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
0.0834	6.971	2.377	0.934	0.971	0.983

We use DeepLabV3+ [27] with ResNet50 as the network backbone to generate the baseline result. We applied random horizontal flipping and random color jitter for data augmentation. The original model was fine tuned to our dataset by training with cross-entropy loss and SGD [28] optimizer for 25 epochs. We employ a commonly used mean Intersection over Union (mIOU) metric to report the performance of the trained model. We achieve mIOU of 51.2% on the test dataset. Qualitative results on sample test images are shown in Fig. 7.

B. Monocular Depth Estimation

Since we have recorded our dataset from the simulator, we have ground truth labels for all objects in the simulator. However, we limit the depth up to 120m for the training and evaluation process. This maximum depth value was chosen based on the range of LiDAR.

We use AdaBins network [29] with ResNet50 as the backbone for the training of depth estimation model. Similar to segmentation, we apply augmentation techniques such as random horizontal flipping and random color jitter. The model was trained for 25 epochs with AdamW [30] optimizer and SI loss proposed by Eigen [31]. We choose a bin size of 256 and a batch size of 16 for the training process.

To report the results of training, we use standard metrics from [31] for monocular depth estimation. These include root mean squared error (RMSE), RMSE log, absolute relative

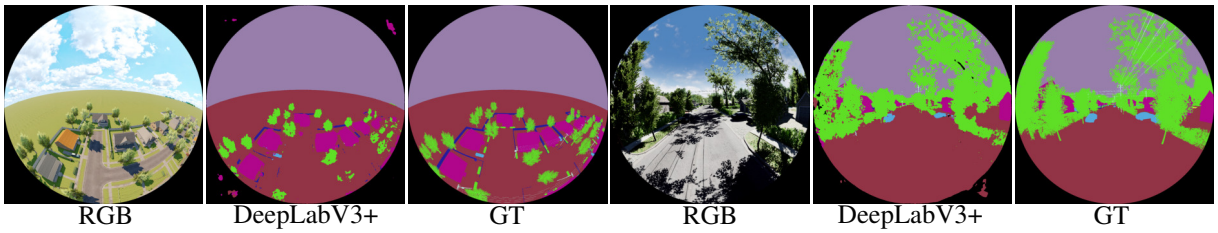

Fig. 7. Qualitative results of semantic segmentation using DeepLabV3+ on AirFisheye dataset

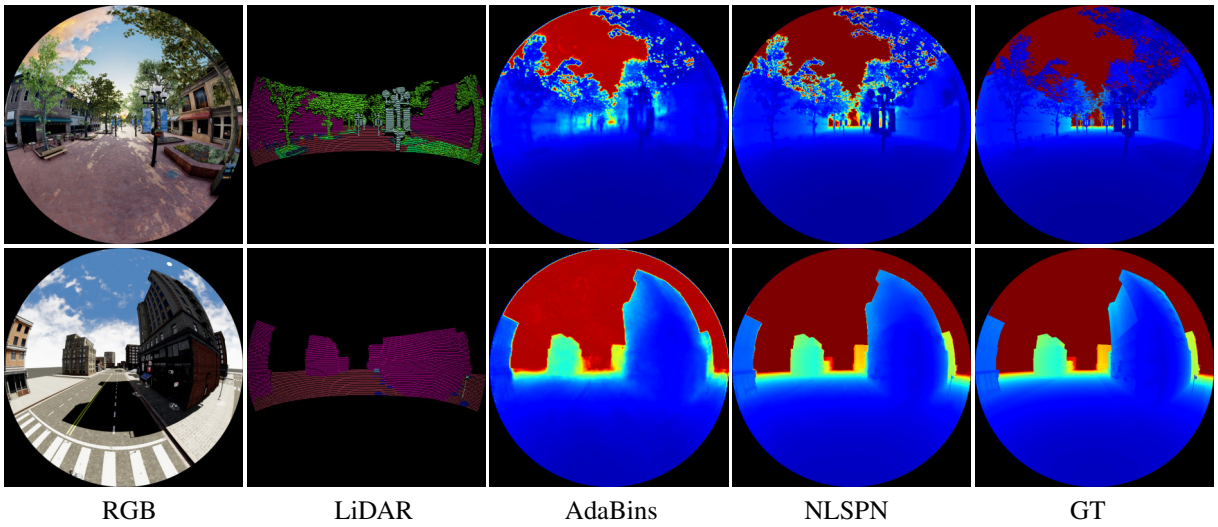

Fig. 8. Monocular depth estimation and depth completion results on AirFisheye dataset. Compared to monocular depth estimation using AdaBins network, depth completion using NLSPN network predicts depth with more finer details.

error (abs Rel), squared relative difference(sq Rel), and threshold accuracies ($\delta_1, \delta_2, \delta_3$). The evaluation results on the test data are reported in Table II.

C. Depth Completion

LiDAR point clouds are transformed into image frame following the occlusion correction algorithm mentioned in section III-C to create LiDAR depth maps for training. Also, for depth completion task, we limit the maximum depth to 120m.

We use NLSPN network [32] with ResNet34 as the backbone for training the depth completion model. We use a batch size of 6, propagation steps of 18, and $\mathbf{Tanh}-\gamma-\mathbf{Abs}-\mathbf{Sum}^*$ [32] for affinity normalization. The number of non-local neighbours was set to 8. The network is trained for 25 epochs with reconstruction loss (ℓ_1 loss + ℓ_2 loss) along with AdamW optimizer.

Except for RMSE log and square relative error, all metrics mentioned in IV-B are used for evaluation. In addition, we also report evaluation results using mean average error (MAE). Evaluation results on test data are reported in Table II. One should be able to notice a significant improvement in depth prediction task, *e.g.* RMSE error reduces from 13.324 to 6.971 when LiDAR data is also used.

V. CONCLUSIONS

In this paper, we highlighted the lack of fisheye data for drone applications and proposed a synthetic AirFisheye dataset to fill this gap. The dataset consists of RGB images, synchronized LiDAR point clouds, along with dense labels for segmentation and depth estimation. We also proposed an occlusion correction algorithm to remove incorrectly projected LiDAR point clouds into image. Furthermore, experiments were carried out to report baseline results for the task of semantic segmentation, depth estimation and depth completion.

Observing the benefits of fisheye cameras for drone application, this paper aims to encourage the development of state-of-the-art algorithms for fisheye images. In future work, we plan to augment the dataset with data from other climatic conditions, *e.g.* rain, fog, snow, *etc.*, and dynamic data such as persons, animals, birds, and other aerial participants, *e.g.* balloons, drones, *etc.* We also plan to evaluate data sim-to-real transferability.

ACKNOWLEDGMENT

The authors would like to thank Thorsten Ehlers and Klaus Lütjens from DLR/LV, as well as Benjamin Lewandowski and Stefan Milz from Spleenlab GmbH for valuable suggestions and recommendations during the development of this work. The authors would also like to thank Gerrit Pauls for constructive criticism of the manuscript.

REFERENCES

[1] M. Wisniewski, Z. A. Rana, and I. Petrunin, 'Drone model classification using convolutional neural network trained on synthetic data', Journal of Imaging, vol. 8, no. 8, p. 218, 2022.

[2] A. Geiger, P. Lenz, and R. Urtasun, 'Are we ready for autonomous driving? The KITTI vision benchmark suite', in 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3354–3361.

[3] N. Gählert, N. Jourdan, M. Cordts, U. Franke, and J. Denzler, 'Cityscapes 3D: Dataset and Benchmark for 9 DoF Vehicle Detection', arXiv [cs.CV]. 2020.

[4] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, 'The apolloscape open dataset for autonomous driving and its application', IEEE transactions on pattern analysis and machine intelligence, 2019.

[5] M. Fonder and M. Van Droogenbroeck, 'Mid-Air: A Multi-Modal Dataset for Extremely Low Altitude Drone Flights', in IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019, 2019, pp. 553–562.

[6] W. Wang et al., 'Tartanair: A dataset to push the limits of visual slam', in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 4909–4916.

[7] V. Badrinarayanan, F. Galasso and R. Cipolla, "Label propagation in video sequences," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 2010, pp. 3265-3272, doi: 10.1109/CVPR.2010.5540054

[8] 'Epic Games. Unreal Engine web site'. [Online]. Available: https://www.unrealengine.com/en-US. [Accessed: 12-Jul-2023].

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, 'AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles', CoRR, vol. abs/1705.05065, 2017.

[10] 'Computer Generated Angular Fisheye Projections'. [Online]. Available: http://paulbourke.net/dome/fisheye/. [Accessed: 12-Jul-2023].

[11] 'Converting to/from cubemaps'. [Online]. Available: http://paulbourke.net/panorama/cubemaps/#3. [Accessed: 12-Jul-2023].

[12] A. R. Sekkat, Y. Dupuis, P. Vasseur, and P. Honeine, 'The OmniScape Dataset', in 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1603–1608.

[13] M. Burri et al., 'The EuRoC micro aerial vehicle datasets', The International Journal of Robotics Research, vol. 35, no. 10, pp. 1157–1163, 2016.

[14] S. Yogamani et al., 'WoodScape: A Multi-Task, Multi-Camera Fisheye Dataset for Autonomous Driving', in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[15] Y. Liao, J. Xie and A. Geiger, "KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 3, pp. 3292-3310, 1 March 2023, doi: 10.1109/TPAMI.2022.3179507.

[16] A. R. Sekkat et al., "SynWoodScape: Synthetic Surround-View Fisheye Camera Dataset for Autonomous Driving," in IEEE Robotics and Automation Letters, vol. 7, no. 3, pp. 8502-8509, July 2022, doi: 10.1109/LRA.2022.3188106.

[17] M.-R. Hsieh, Y.-L. Lin, and W. H. Hsu, 'Drone-based object counting by spatially regularized regional proposal network', in Proceedings of the IEEE international conference on computer vision, 2017, pp. 4145–4153.

[18] D. Du et al., 'The unmanned aerial vehicle benchmark: Object detection and tracking', in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 370–386.

[19] M. Gochoo et al., 'FishEye8K: A Benchmark and Dataset for Fisheye Camera Object Detection', in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 5304–5312.

[20] S. Schneider, M. Himmelsbach, T. Luettel and H. -J. Wuensche, "Fusing vision and LiDAR - Synchronization, correction and occlusion reasoning," 2010 IEEE Intelligent Vehicles Symposium, La Jolla, CA, USA, 2010, pp. 388-393, doi: 10.1109/IVS.2010.5548079.

[21] V. R. Kumar et al., 'Monocular fisheye camera depth estimation using sparse LiDAR supervision', in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 2853–2858.

[22] J. S. Berrio, M. Shan, S. Worrall, and E. Nebot, 'Camera-LiDAR integration: Probabilistic sensor fusion for semantic mapping', IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 7637–7652, 2021.

[23] 'Modular Neighborhood Pack'. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/modular-neighborhood-pack. [Accessed: 12-Jul-2023].

[24] 'Modular House'. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/modular-houses. [Accessed: 12-Jul-2023].

[25] 'Modular Building Set'. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/modular-building-set. [Accessed: 12-Jul-2023].

[26] 'Downtown West Modular Pack'. [Online]. Available: https://https://www.unrealengine.com/marketplace/en-US/product/6bb93c7515e148a1a0a0ec263db67d5b. [Accessed: 12-Jul-2023].

[27] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 801–818.

[28] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in International conference on machine learning, 2013, pp. 1139–1147.

[29] S. F. Bhat, I. Alhashim, and P. Wonka, 'Adabins: Depth estimation using adaptive bins', in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 4009–4018.

[30] I. Loshchilov and F. Hutter, "Fixing Weight Decay Regularization in Adam," CoRR, vol. abs/1711.05101, 2017, [Online]. Available: http://arxiv.org/abs/1711.05101

[31] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," Advances in neural information processing systems, vol. 27, 2014.

[32] J. Park, K. Joo, Z. Hu, C.-K. Liu, and I. So Kweon, 'Non-local spatial propagation network for depth completion', in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16, 2020, pp. 120–136.