Mid-Air: A multi-modal dataset for extremely low altitude drone flights

Michaël Fonder
University of Liège
michael.fonder@uliege.be

Marc Van Droogenbroeck
University of Liège

Abstract

Flying a drone in unstructured environments with varying conditions is challenging. To help producing better algorithms, we present Mid-Air, a multi-purpose synthetic dataset for low altitude drone flights in unstructured environments. It contains synchronized data of multiple sensors for a total of 54 trajectories and more than 420k video frames simulated in various climate conditions. In this work, we motivate design choices, explain how the data was simulated, and present the content of the dataset. Finally, a benchmark for positioning and a benchmark for image generation tasks show how Mid-Air can be used to set up a standard evaluation method for assessing computer vision algorithms in terms of robustness and generalization. We illustrate this by providing a baseline for depth estimation and by comparing it with results obtained on an existing dataset. The Mid-Air dataset is publicly downloadable, with additional details on the data format and organization, at ***http://midair.ulg.ac.be****.*

1. Introduction

The last decade has seen a continuously growing interest for autonomous vehicles of all types, including cars and Unmanned Aerial Vehicles (UAVs), commonly known as drones. The main challenge posed for methods trying to achieve vehicle autonomy relies in their capacity to understand their state and environment. This can only be done by properly analyzing and combining information provided by various sensors such as cameras, radars, or Inertial Measurement Units (IMUs).

Currently, the best methods for flying drones automatically rely on machine learning algorithms and, for vision-related tasks, involve neural networks and deep learning. This is due to their ability to learn complex patterns from examples by bypassing the need of an explicit analytic model. The common weakness of this approach lies in the amount of data required for training such networks. Furthermore, if the training data does not completely represent the task, there is a risk of experiencing poor performance in practice. Stated otherwise, an algorithm trained with insufficiently varied samples will not generalize properly.

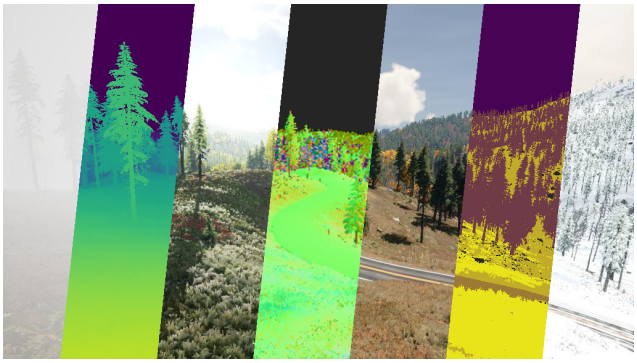


Figure 1: Extract from our Mid-Air dataset. Simulated data when flying our drone in a scene includes, from left to right, an RGB image under a foggy weather, the depth map, an RGB image for a spring sunset, the normal map, an RGB image of a clear sky weather during fall, the semantic segmentation map, and an RGB image in a cloudy winter.

Our main motivation for building a new dataset, named Mid-Air, is to provide a large multi-modal dataset for machine learning tasks such as visual odometry, simultaneous localization and mapping, depth estimation, semantic segmentation, or stereo disparity estimation. These tasks can be targeted individually or simultaneously in a multi-task set up. With Mid-Air, we not only provide a basis for training and benchmarking algorithms, but also for innovation, by providing a new type of data.

1.1. Existing datasets

Various datasets exist for developing computer vision tasks [5, 6, 18, 22, 25, 26, 28], but only a few of them are large enough for machine learning algorithms or provide data of several sensors for multi-task learning. The Kitti dataset [9, 11, 12, 19] is probably the most complete one and the current reference in the field. Due to a lack of alternatives, most methods that jointly estimate two types of information, such as image depth and camera egomotion [17, 31] for example, are cursed to be trained and/or

Datasets	Mid-Air	Kitti [9, 11, 12, 19]	Virtual Kitti [10]	Synthia [14, 24]	RGB-D SLAM [29]	EuRoC MAV [4]
Number of trajectories	54	71	50	7	19	11
Number of frames	119k* (@25Hz)	44k (@10Hz)	21k* (@10Hz)	7k* (@5Hz)	48k (@30Hz)	27k (@20Hz)
Total duration	79 min	73 min	35 min	23 min	27 min	22 min
Resolution	1024×1024	1382×512	1242×375	960×720	640x480	752×480
Data type	synthetic	real	synthetic	synthetic	real	real
Camera motion	**drone flight**	car drive	car drive	car drive	hand-held	drone flight
Environment type	**unstructured**	city	city	city	indoor	indoor
Climate variations	yes	no	yes	yes	no	no
IMU data	yes	yes	no	no	yes	yes
GPS	yes	yes	no	no	no	no
Depth map	dense	sparse	dense	dense	dense	sparse
Stereo disparity map	yes	yes	no	no	no	no
Surface normals	yes	no	no	no	no	no
Semantic segmentation	yes	yes	yes	yes	no	no
Instances segmentation	no	yes	yes	yes	no	no
Optical flow	no	yes	yes	no	no	no

Table 1: Comparison between our Mid-Air dataset and similar datasets usable for multi-task learning. The symbol * denotes that several additional RGB videos, for different conditions, are available for a same trajectory.

tested on this dataset alone.

Table 1 summarizes some common multi-modal datasets and compares them to our Mid-Air dataset (see first column), according to the amount of data, the acquisition conditions, and types of data. Table 1 also highlights that the largest and most complete datasets were mainly designed for autonomous car applications. In fact, they provide data of sensors mounted on a car driving in urban environments. This raises a concern about the generalization potential of methods trained and tested on such datasets to other use cases such as autonomous drones. Cars are indeed nonholonomic vehicles which move in constrained environments, *i.e*. roads. Therefore, sensors mounted on them only encounter a limited subset of motion types, and cameras do not completely explore their 3D environment. As a result, algorithms trained on these datasets may be strongly biased towards autonomous cars applications. This motivates the need for the development of a dataset specific to flying drones for unstructured environments.

1.2. Mid-Air: a new synthetic dataset

We present a synthetic dataset for unstructured environments with navigation and vision sensors mounted on board of a flying quadcopter; the main characteristics of this dataset, named Mid-Air, are summarized in Table 1.

Using a drone instead of a car for recording allows to have a variety of camera motions and poses which would not be possible with a car. As shown in Table 1, our aim is not only to offer an alternative to current datasets for flying drones, but also to provide material to develop algorithms for autonomous vehicles which could be generalizable to a wider variety of motions.

With this purpose in mind, focus was put on providing data to train algorithms for robustness to visual changes; two new specific benchmarks are presented to explicitly test the latter. For this purpose, we exploited the control offered by the simulator to record the same flight trajectories several times, with varying climate conditions (different seasons, time of day, and weather conditions).

Generating data synthetically also helps to record ground-truth data which is more reliable or even impossible to capture with real sensors. In Mid-Air, we do not only provide common dense ground-truth visual maps such as depth maps, but also introduce surface normal maps, a new type of dense ground-truth data previously unseen in a dataset. This innovates by paving the way for the development of methods for surfaces normal estimation tasks.

To generate images, we used a real-time rendering software. Despite being physically inaccurate, real-time rendering techniques have reached a level of visual likelihood and realism which competes with simple ray-tracing algorithms while offering the benefits of reduced render time compared to their physically accurate alternatives. We detail the implications of this choice in the next section. For the rendering pipeline, we used the Unreal Engine [8], a game engine, in combination with Airsim, a plugin developed by Shah *et al*. [27] for this engine. Airsim is a multirotor drones simulation framework and provides an API to control the simulation.

2. Sensors simulation

Building a synthetic dataset for flying drones in unstructured environments requires a precise model of the drone and its sensors. For the physics of the drone, we rely on the default quadcopter drone model provided by the Airsim simulator. However, due to some shortcomings of the simulator, we had to re-implement some sensor models from scratch or to tweak others to gain more control on the gener-

ated data and increase their accuracy. In this section, we detail the modifications which were made to the default models provided by Airsim and elaborate on the reasons which led us to make these modifications.

2.1. Accelerometer and gyroscope

The accelerometer and the gyroscope, two sensors constituting the core of the inertial measurement unit (IMU), are essential for stabilization and more generally for flying a drone. Unfortunately, both are not free of imperfections; they are prone to bias, bias drift, and measurement noise. It is commonly assumed that the bias, b_t, behaves as a Gaussian random walk process, and that the measurement noise follows a Gaussian (or normal) distribution with a zero mean. Accordingly, for a sampling period dt, the relationship between the ground-truth measurement m_{GT} and the sensor measurement m_{sens} for one axis is as follows:

$$\begin{aligned} m_{\text{sens}} &= m_{\text{GT}} + \nu_n + b_t \text{ where } \nu_n \sim \mathcal{N}(0, n) \text{ and} \\ b_t &= b_{t-1} + \nu_b \text{ where } \nu_b \sim \mathcal{N}\left(0, b_0\sqrt{\frac{dt}{t_a}}\right), \end{aligned} \tag{1}$$

where n, b_0 and t_a are parameters that can be determined by carrying an Allan diagram analysis (see [30]) and which differ for each individual sensor.

These parameters can however not be modified by the API of Airsim. Since we wanted to generate trajectories with different IMU settings, we reimplemented this model. Prior to each flight, a new set is drawn randomly within bounds that are representative of the variations typical for different IMU models. The choice of the order of magnitudes to use was guided by experimental measurements given in different studies of various sensor models [15, 20, 21, 23].

It is important to note that both the accelerometer and the gyroscope are also prone to axis misalignment and scaling factor issues. Unfortunately, these two imperfections appear to be far less studied experimentally. For this reason, we preferred to rely on a model commonly adopted for control applications despite the fact that it neglects these parameters.

2.2. GPS receiver

GPS receivers are heavily used for positioning in the context of autonomous driving and driving assistance. They are indeed able to regress their absolute position based on the satellites that are in line of sight. The regression process has three steps. First, the sensor computes the delay between the time of emission of GPS signals by each satellite and their time of arrival to the sensor. This delay is derived from the data encoded in the signals. After that, the sensor uses these delays to estimate the distance separating it from each satellite. These distances are called pseudoranges. Eventually, the sensor can triangulate its position with an optimization method based on the positioning knowledge it has about GPS satellite positions and the pseudoranges estimated during the previous step.

A GPS model. GPS positioning would be perfect if pseudoranges could be computed precisely. Perfect distances would be obtained if all clocks were perfectly synchronized and if the speed of light was known all along the path between a satellite and the receiver. However, this is not the case in practice. Clocks have indeed small, but measurable offsets. Furthermore, the atmosphere is composed of several layers which interact differently with the light and therefore modify its speed and path. Without loss of generality, the relationship between the estimated and real pseudorange for a given satellite and a given carrier frequency, ρ_e and ρ_r respectively, can be expressed as follows (see [1]):

$$\rho_e = \rho_r + c\,(\delta_i - \delta_R) + \Delta_I + \Delta_T + \nu\,, \tag{2}$$

where c is the speed of light, δ_i and δ_R are the satellite signal inaccuracy and the receiver clock offset respectively, Δ_I and Δ_T are the delays induced by the ionosphere and the troposphere respectively, and ν is the receiver measurement noise. Since the ionosphere induces a delay which is inversely proportional to the frequency [1], Δ_I can be estimated only by using several carrier frequencies. Other inaccuracies and delays cannot be estimated by the receiver. For those reasons, a good GPS receiver simulation can only be obtained when these imperfections are accurately modeled.

Shah *et al.* [27] do not provide any detail on the model built in Airsim, which makes it hardly reliable when trying to simulate real world conditions. For this reason, we developed our own model and had to choose between a model for a single-frequency or a dual-frequency receiver. As the latter are currently not widespread, we built a custom single-frequency receiver model derived from the dual-frequency model and implementation given by Agarwal and Hablani [1]. The adaptation is straightforward and consists in simply removing all the corrections enabled by the use of several carrier frequencies. More precisely, we removed the ionosphere delay correction term from the estimated pseudoranges used for the optimization process.

Satellites visibility. To assess the satellites which are in line of sight, the authors of [1] make the assumption that the Earth is a perfect, smooth ellipsoid and that there are no obstacles. This is hardly acceptable for drones that fly at low altitudes. Hence, we decided to refine the model by considering the shape of the 3D environment generated by the simulator. Our idea consists in adding a wide-angle depth camera pointing upwards the drone. This creates a map of areas of the sky which are occluded by obstacles. Assuming that the starting position of the drone was mapped to an arbitrary location on the Earth surface, and since we perfectly know the position and the attitude of the drone, it is then

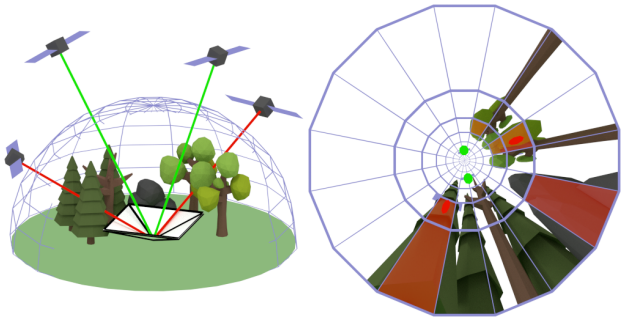


Figure 2: Illustration of the method used to determine which GPS satellites are in line of sight. The left picture shows a 3D perspective of the scene. The right picture shows what is seen by the receiver. Green and red dots correspond to projected satellites positions which are respectively in line of sight or not according to our method. Areas overlaid with red do not contain any pixel belonging to the sky.

possible to project the satellite positions on this map and determine if some satellites are occluded by the presence of obstacles. A simple rule would be to discard a satellite as soon as it is occluded by an obstacle along the path. This rule therefore assumes that all obstacles are perfectly stopping the signals. However, several studies [3, 16] contradict this assumption for trees. Since the environments used for our database contain a lot of them, another rule was developed. Because sky occlusion maps are missing information about the amount of GPS signal absorption and availability, we defined an empirical rule illustrated in Figure 2. It basically consists in dividing the sky in several areas. A satellite is then considered as occluded if no sky portion is visible in the whole area on which it is projected.

2.3. RGB camera image rendering

As mentioned previously, we used the Unreal engine for rendering the images of our dataset, because it offers a good trade-off between rendering time and visual accuracy. Hereafter, we discuss the consequences on the visual accuracy in terms of shading, geometrical limitations, and level of details. It is important to elaborate on the consequences of our choices as they might have an impact on the generalization of learning algorithms to real-life scenarios. We also explain why we believe they are acceptable.

Shading. The Unreal engine uses an enhanced version of the deferred shading algorithm. This shading method was designed for real-time rendering. For this reason, shadings are not rendered with physically accurate equations, but rather with simplified models which were targeting a good visual likelihood. Such models are well suited for diffuse surfaces but have difficulties to deal with surfaces which interact with light rays in other fashions. This is especially true for reflective and refractive materials. Nowadays, the preferred method for faking reflections consists in projecting and capturing the surrounding of an arbitrary area on a cubemap and to project this cubemap on the surface of all objects present within this area. With this method, reflections are skewed and light rays do not bounce on reflective objects. Refractions on the other side are often approximated by simple transparent materials and will not deflect light rays as in the real world. In our dataset, those reflections and refractions concerns arise only for water planes and are almost nonexistent for other features of our environments. The risk of bad generalization due to this factor is therefore limited.

Geometrical limitations. In addition to shading limitations, there are some geometrical limitations. Since computers have a finite amount of memory, it is unrealistic to populate a virtual world with an infinite amount of different objects. What is done in practice is to use a limited subset of assets and to replicate it with basic modifications (scale and orientation) over the environment where needed. This has obvious implications for semantic segmentation algorithms and must therefore be kept in mind when using synthetic datasets in general.

Eventually, game engines come with a further memory and computation cost saving feature that is called the "Level of Details" (LoD). This feature reduces the amount of faces to be displayed by simplifying the geometry of objects more and more aggressively as the distance to the camera increases. This simplification is dynamic and leads to object reinstantiations during runtime. It could therefore be harmful for computer vision learning algorithms relying on video streams. As this feature is mandatory for large environments, we took special care in tuning it such that the reinstantiation only occurs past a distance at which the visual impact on the RGB capture becomes minimal.

2.4. Synthetic sensors / 3D and semantic sensors

Working with a simulator enables to gather information about the environment and the 3D world scene which is not possible to capture with real sensors. For example, we can think about perfect dense depth maps, perfect and automatically annotated segmentation maps or even normal maps.

They can be created by exploiting features implemented for deferred shading and gathered by means of the Airsim API. The only major difficulty is that objects using transparency do not appear on normal maps. This is due to the intrinsic design of the deferred rendering pipeline. Since the only transparent objects present in our datasets are water planes, we decided to solve this issue by assigning a perfectly vertical normal to all pixels of the map belonging to a water plane.

Stereo ground truths. Since we perfectly know the virtual environment, it is also possible to generate ground truths

Algorithm 1 Occlusion mask generation.

For each pixel $p_{i,j}$ of $depth_{camera1}$:
  Compute 3D position of $p_{i,j}$ relatively to camera 1;
  Express 3D position of $p_{i,j}$ relatively to camera 2;
  Project $p_{i,j}$ on camera 2 sensor plane;
  Get coordinates $[k, l]$ of corresponding projection
  if $depth_{camera1}\,[i, j] > depth_{camera2}\,[k, l]$:
    Surface corresponding to $p_{i,j}$ is hidden to camera 2;

for stereo disparity maps and occlusion masks. The disparity d expressed in pixels can indeed be inferred from a single planar depth map[1] Z expressed in meters by using the following equation:

$$d = \frac{fb}{Z}\,, \tag{3}$$

where f is the focal length of the corresponding camera in pixels, and b is the baseline between the two cameras expressed in meters.

To calculate the occlusion mask, we have to determine the areas of a picture taken by one of the two cameras which are not visible by the second camera. This mask can be inferred using the planar depth maps corresponding to the scene seen by each camera and is detailed in Algorithm 1. It is important to note that the disparity map and occlusion mask are always specific to one of the two cameras.

3. Dataset presentation

After the details related to the simulation environment, we now present the design and content of our Mid-Air dataset. In particular, this section explains the different design rules. We first introduce the drone setup that has been used to record the trajectories. Then, we give some details about the environments used for the flights. Finally, we present an overall view of the dataset content.

3.1. Drone setup

The physical drone model used for the flights is the default quadcopter model of Airsim with customized sensors types and placements. We used three different cameras: a front-looking camera placed on the X-axis of the drone, another front-looking camera with a 1-meter baseline compared to the first for stereo applications and a last camera looking downward. The latter can be especially useful for visual odometry and SLAM algorithms [2] and was therefore added to the common front-looking cameras. The left-stereo camera, the IMU, the GPS receiver and the down-looking camera are placed exactly at the same location (see Figure 3 for a schematic view of the drone setup). This choice, even if unrealistic, should not have any impact on the learning algorithms and greatly eases the use of the dataset since no additional translations are required when working with several sensors at the same time.

[1]Contrary to a true depth map which gives the Euclidean distance between an object and the focal point of the camera, a planar depth map gives the distance of an object to the plane which passes through the camera focal point and whose normal corresponds to the Z camera axis.

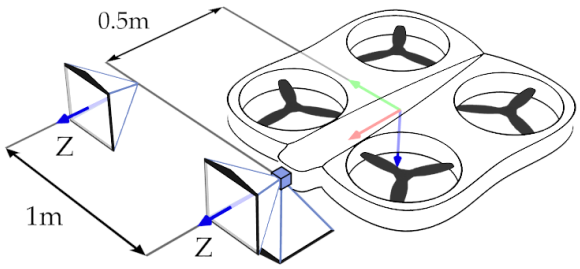


Figure 3: Sensor locations on the drone used to generate our dataset. Cameras are represented by the pyramids; the blue cube shows the IMU and the GPS receiver locations.

The cameras use the pinhole camera model to capture images. They are therefore perfectly calibrated by default. This is once again an advantage since it enables to remove the camera calibration method out of the equation when comparing different algorithms. Due to render engine limitations, cameras act as global shutter cameras and do not present any motion blur issues. They are all set to capture images at a rate of 25 Hz with a field of view of 90 degrees. In addition to RGB data, the left stereo camera captures a semantic segmentation map, the depth map, the normal map, the stereo disparity map, and the occlusion mask. All images have a size of 1024×1024 pixels, except the normal maps which have a size of 512×512 pixels.

The IMU measurements refresh rate is set to 100 Hz and the GPS receiver updates its position every second. The parameters of the IMU are randomly drawn before each flight and the initial bias is logged for each trajectory. The same yields for the initial GPS position. The latitude, longitude and altitude are drawn uniformly in the ranges of $[0, 60]$ degrees, $[-180, 180]$ degrees and $[-500, 500]$ meters respectively. The GPS satellites having an orbital period of 12 hours, we also randomized the initial trajectory time to get different satellite positions.

3.2. Environments setup

As stated earlier, our dataset aims to provide data to train and test algorithms for robustness in unstructured environments. This goal can only be achieved if the data is varied. To guarantee enough variety, we used two different large-scale environments displaying varied features (see Figure 4) and different weather setups. The following subsections present the features of the setups in which we flew our drone.

Figure 4: Samples of our dataset illustrating the variety of environments used to generate the visual information.

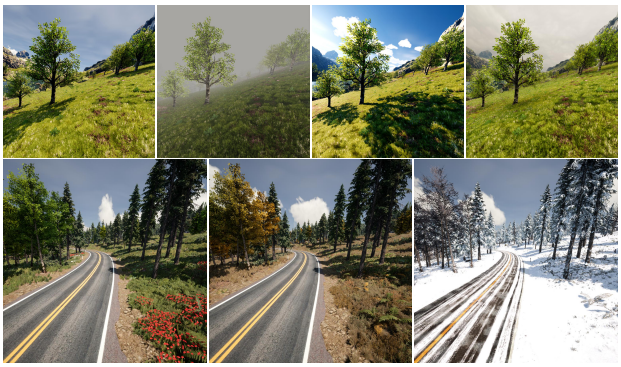


Figure 5: Samples of our dataset showing the different climate setups. The top row shows the four simulated weathers. The bottom row illustrates the seasons.

3.2.1 Landscapes

The first used environment is the map given in the Kite demo of the Unreal engine. It features a mountain landscape with several lakes and forests (see the first row of Figure 4). Its size, which reaches almost $100\,\mathrm{km}^2$, guarantees to find places with varied characteristics and features. Its topography makes it perfect for training algorithms to deal with uneven grounds and height variations.

The second environment is made up of the two demo maps provided by the PLE plugin for the Unreal engine. Together they cover an area of roughly $10\,\mathrm{km}^2$ and feature an hilly landscape with forests and some lakes, as for the first environment. In addition, both maps are crossed by a road (with signs, barriers, ...) and a railway track. The specificity of these maps lies in the possibility to change the season and therefore to change the visuals of the environment. Samples of these maps are shown in the second row of Figure 4.

3.3. Climate conditions

For the climate settings, we have two tunable modalities: the weather and the season. The weather parameter mainly affects the sky color as well as the illumination of the scene. The season parameter, on the other hand, mostly affects the colors present in the environment. Since nature variety is extremely large, possibilities of configurations are endless. In order to keep a tractable size for the dataset, we restricted ourselves to a carefully chosen subset of scenarios.

We chose to simulate four distinct weathers and three seasons. To generate the different weathers, we use the TrueSky plugin for the Unreal engine. It allows to create volumetric and dynamic clouds able to cast shadows on the map. They therefore have a realistic behavior, which is important for video applications. For the seasons setups, we relied on the presets of the PLE plugin, given hereafter:

Clear sky at midday. The illumination is typical for a normal use case in sunny weather. Shadows are harsh and sky color is mainly blue.

Overcast sky. The illumination is dim and the shadows are almost absent. This and the gray sky color provide a realistic representation for cloudy weathers.

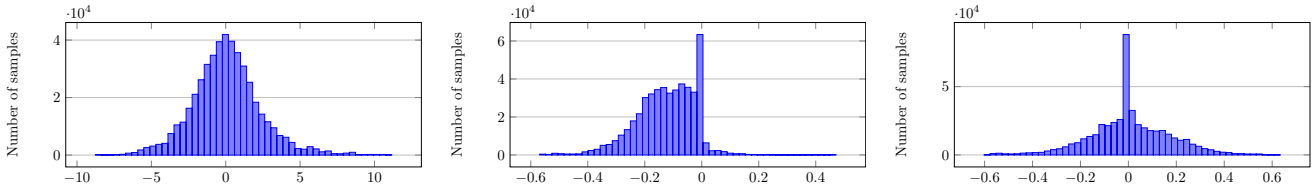
Sunset. This weather was chosen for its challenging illumination conditions. The shadows are indeed heavily elongated. This creates areas which are completely shadowed and therefore require good illumination robustness to be parsed correctly. In addition, the sun is low and can enter the field of view, which creates simulated sunglares.

Fog. This weather is challenging for algorithms targeting visual odometry. Since visual features fade with distance, algorithms have less visual cues to rely on to correct the state estimation, leading to less accurate state corrections. This weather is also valuable for testing the robustness of image generation tasks since it tends to desaturate colors.

Seasons. We found out that all seasons do not contribute equally to the diversity of the dataset. Summer was discarded due to its resemblance with spring and fall. Including it would not have added any significant variety to the dataset, while discarding it allows to reduce the dataset size. We kept the spring, fall, and winter seasons. Spring features trees with green leaves and luxuriant ground vegetation. Fall differentiates itself from spring with trees with yellow leaves and dried ground vegetation. Finally, winter is interesting for its trees without leaves and an environment covered with snow.

3.4. Scenarios and format

After having properly defined the environments and setups, we manually flew the drone in the simulator using an RC controller connected to the computer through USB and recorded 5 hours of flight. We then extracted 79 minutes out of it. These 79 minutes correspond to 54 trajectories of equal length, *i.e.* 1.47 minute each. The first 30 are captured in the Kite demo environment and the remaining ones in the PLE environment. Figure 6 displays the distribution of some characteristics of the camera motions which have



(a) Vertical velocity $[ms^{-1}]$ (b) Pitch angle $[rad]$ (c) Roll angle $[rad]$

Figure 6: Histograms displaying the distribution of some features of the camera motions present in our dataset.

more variety in our dataset than in datasets captured with a car. For these datasets, the corresponding histograms would all have a shape similar to a normal distribution with a mean of zero and a variance extremely close to zero.

Each trajectory record is then rendered several times, once for every climate scenario. In practice, this translates into rendering trajectories belonging to the Kite demo environment once for each weather setup and once for each season for those belonging to the PLE environment. Since there can be some differences due to objects animation between each render, all data streams are recorded simultaneously at each run.

The data includes the ground-truth positioning information, *i.e*. the position, velocity, acceleration, attitude and angular velocity, the IMU sensor measurements, the estimated GPS position along with complementary information on the GPS signal such as its dilution of precision and the number of visible satellites, and finally the camera data, *i.e*. the left, right and down-looking RGB images and the segmentation, depth, normals, disparity and occlusion maps corresponding to the left camera. Our semantic segmentation dataset contains a total of 12 different classes (animal, tree, dirt ground, rocky ground, ground vegetation, boulder, water plane, man-made construction, road, train track, road sign, other man-made objects).

Sensors data is stored in a common `hdf5` dataset file while the pictures are saved independently in several subdirectories. This enables a good dataset handling and eases data accesses. RGB pictures are stored in JPEG files and maps are stored as 16-bit float matrices encoded with a lossless PNG format. On average, the set of 8 images recorded at each frame weights less then 2.5 MB. This is roughly 6 times less than the space required for the same data stored in uncompressed raw format. All additional information about data layout and organization are given on the website of our dataset. To ease the data layout understanding and parsing process, we also provide several example scripts with the dataset.

4. New benchmarks

Since efficient communication on results obtained on a dataset can only be achieved if all the community uses the same separation between data used for developing new methods and the data used to test their performance, we propose two additional sets to be used only for comparing results. Both sets use the same sensor models and provide the same data as the one defined for the training dataset. It should be said that other splits between training, validation and test data are possible, but this requires a consensus of the scientific community. Our intention is, later, to detail evaluation methodologies that are discussed and accepted by the community based on Mid-Air.

4.1. Benchmark for positioning tasks

To test the performance on positioning tasks such as visual-inertial odometry or SLAM algorithms, we provide three additional trajectories recorded in an unseen environment. To assess the robustness of tested algorithms, all trajectories were recorded for three weather conditions, *i.e*. clear sky, fog, and sunset. As explained in the next paragraphs, each trajectory has its own specificity so that performance scores should be reported independently for all nine possible scenarios.

Ideally, generic positioning algorithms have to be robust to visual novelties. That is why we recorded our trajectories in an environment with strong visual differences compared to the training data. For this, we used a modified version of the Landscapes Mountain demo map for the Unreal engine.

The first trajectory has a length of 5 minutes and was generated by manually flying the drone, as for the training data. The two other trajectories have a length of 10 minutes and were generated synthetically. They follow both the same path except that, for one of them, the drone is looking towards where it is going and, on the other, the yaw of the drone is shifted by 90 degrees. This allows to have a scenario where the visual features are moving towards the camera, and another where they are scrolling from left to right in the frame.

The synthetic path was chosen to be challenging and consists in a circular trajectory with a varying height, radius and angular velocity with no periodicity. This is an extreme case because the IMU measurements constantly change, and no loop closure is possible to periodically correct the state estimation. Therefore, failing to properly use visual cues to correct the state estimate will lead to significant drift.

4.2. Benchmark for visual maps generation tasks

For image generation tasks, such as semantic segmentation or depth estimation, we provide eleven additional 13s long trajectories, excluded from the training dataset. Five of them belong to the Kite map and are rendered four times, one for each weather setup. The six remaining were recorded in the PLE maps and are rendered once for each of the chosen seasons. This choice preserves a data distribution which is close to the one of the training set.

To analyze the robustness of methods to visual changes, we recommend to report results on this benchmark separately for each weather and season setup. If the scores are similar for all scenarios, the method will be assumed to be robust to visual changes. On the other hand, a difference in performance will highlight some robustness deficiencies.

Baseline example. To illustrate our approach, we applied this benchmark to the challenging task of monocular depth estimation. For this, we decided to train a method whose results are close to the state-of-the-art on the Kitti dataset and for which the code for training is publicly available. We chose the deep neural network given by Godard *et al*. [13]. We trained it for 15 epochs on our complete dataset, *i.e*. approximately 420k samples. Results including the post-processing method of [13] are reported in Table 2.

We can see that even the performance on the training dataset is poor when compared to the scores obtained on the Kitti benchmark. Training it for 5 additional epochs did not bring any significant performance improvement. We therefore conclude that our dataset has a higher complexity than that of the Kitti dataset, leaving room for further improving methods. Additionally, the results can be grouped in clusters corresponding to the environments with one performing significantly better than the other. Since there are more samples for the Kite environment than for the PLE one, this observation indicates that this method tends to learn features specific to an environment rather than general patterns and may therefore have poor generalization capabilities. It is however worth mentioning that the network did not overfit since scores on the training and the test data are close. Moreover, performance is similar for proposed robustness scenarios, which tends to indicate that the method is relatively robust to visual changes.

4.3. Overall limitations

Despite all our efforts, we did not address two specific situations which might be important for generalization.

The first one is that our dataset contains only few moving objects. Motion is present only through the agitation of the vegetation due to the wind and through the few animals present in the environments. It means that a learning algorithm working on video sequences will be poorly prepared for image changes which are not due to perspective and camera motion.

Test data	Abs Rel	Sq Rel	RMSE	RMSE log
Kitti	0.114	0.898	4.935	0.206
Training set	0.335	8.892	11.444	0.343
Kite sunny	0.250	3.743	10.281	0.317
Kite cloudy	0.225	3.183	8.922	0.280
Kite sunset	0.328	6.570	11.484	0.350
Kite foggy	0.255	3.586	9.495	0.305
PLE fall	0.887	22.801	17.022	0.620
PLE spring	0.792	20.864	17.285	0.613
PLE winter	0.764	20.745	17.771	0.614

Table 2: Results of the network of [13], trained and tested according to our benchmark proposal, on standard metrics to be minimized. The first line gives results reported in [13] for the same network trained on the Kitti and Cityscapes datasets, and tested on the Kitti dataset with the split given by Eigen *et al*. [7]. Performance scores for the training set are computed only for frame numbers multiple of 100.

A second limitation is that the used simulator does not model wind nor drone vibrations. These two parameters however have a significant impact on the IMU. The former creates accelerations which are not induced by the drone propellers while the latter adds additional noise terms to the measurements made by the sensors. Both can have an impact on the generalization of algorithms relying on IMU data. Our dataset should nonetheless be useful to get a reliable performance score for simple scenarios and therefore to get a first overview on the potential of any tested method.

5. Conclusion

We introduce a new synthetic dataset, named Mid-Air, featuring 79 minutes of drone flight recorded several times with different climate conditions. The content of our dataset consists in multiple synchronized modalities providing data for positioning tasks such as SLAM or visual odometry as well as for pure computer vision tasks such as depth estimation or semantic segmentation. While being specifically designed for flying drones in unstructured environments, its size (more than 420k individual frames) and content makes it also useful for training and testing generic machine learning algorithms.

Large datasets such as ours pave the way for building new benchmarks to evaluate algorithms achieving single or multiple application tasks, which should be addressed simultaneously to cope with the environmental complexity. Hence, after discussing the sensor models and elaborating on the simulated scenarios, we propose two possible benchmarking approaches and illustrate one of them by giving the performance of a known algorithm that serves as a baseline. This proposal is opened to discussion and modification in order to adjust it to the needs of the scientific community.

References

[1] S. Agarwal and H. Hablani. Automatic aircraft landing over parabolic trajectory using precise GPS measurements. *Int. Conf. & Workshop on Emerging Trends in Technology (ICWET)*, 1(7):38–45, Feb. 2011. 3

[2] M. Aqel, M. Marhaban, M. Saripan, and N. Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1897–1923, Oct. 2016. 5

[3] A. Bastos and H. Hasegawa. Behavior of gps signal interruption probability under tree canopies in different forest conditions. *Eur. J. of Remote Sensing*, 46(1):613–622, 2013. 4

[4] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016. 2

[5] D. Butler, J. Wulff, G. Stanley, and M. Black. A naturalistic open source movie for optical flow evaluation. In *Eur. Conf. Comput. Vision (ECCV)*, volume 7577 of *Lecture Notes Comp. Sci.*, pages 611–625. Springer, Oct. 2012. 1

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 3213–3223, Las Vegas, NV, USA, June 2016. 1

[7] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Adv. in Neural Inform. Process. Syst. (NIPS)*, pages 2366–2374, 2014. 8

[8] Epic Games. Unreal Engine web site. https://www.unrealengine.com. 2

[9] J. Fritsch, T. Kuehnl, and A. Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *IEEE Intell. Transp. Syst. Conf. (ITSC)*, pages 1693–1700, The Hague, Netherlands, Oct. 2013. 1, 2

[10] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. VirtualWorlds as proxy for multi-object tracking analysis. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 4340–4349, Las Vegas, NV, USA, June 2016. 2

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. of Robotics Res.*, 32(11):1–11, Sept. 2013. 1, 2

[12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 3354–3361, Providence, RI, USA, June 2012. 1, 2

[13] C. Godard, O. Mac Aodha, and G. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 6602–6611, Honolulu, HI, USA, July 2017. 8

[14] D. Hernandez-Juarez, L. Schneider, A. Espinosa, D. Vázquez, A. López, U. Franke, M. Pollefeys, and J. Moure. Slanted stixels: Representing San Francisco's steepest streets. In *Brit. Mach. Vision Conf. (BMVC)*, pages 1–12, London, United Kingdom, Sept. 2017. 2

[15] A. Hussen and I. Jleta. Low-Cost Inertial Sensors Modeling Using Allan Variance, May 2015. 3

[16] G. Lachapelle, J. Henriksen, and T. Melgard. Seasonal effecf of tree foliage on GPS signal availability and accuracy for vehicular navigation. In *Int. Techn. Meeting of the Satellite Division of the Institute of Navigation*, pages 527–532, Sept. 1994. 4

[17] R. Mahjourian, M. Wicke, and A. Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 5667–5675, Salt Lake City, UT, USA, June 2018. 1

[18] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 4040–4048, Las Vegas, NV, USA, June 2016. 1

[19] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 3061–3070, Boston, MA, USA, June 2015. 1, 2

[20] C. Naranjo. Analysis and modeling of MEMS based inertial sensors, 2008. 3

[21] K. Nirmal, A. Sreejith, J. Mathew, M. Sarpotdar, A. Suresh, A. Prakash, M. Safonova, and J. Murthy. Noise modeling and analysis of an IMU-based attitude sensor: improvement of performance by filtering and sensor fusion. In *Proceedings of the SPIE*, volume 9912, pages 1–10, July 2016. 3

[22] D. Olid, J. Facil, and J. Civera. Single-view place recognition under seasonal changes. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, Madrid, Spain, Oct. 2018. 1

[23] L. Pupo. Characterization of errors and noises in MEMS inertial sensors using Allan variance method, 2016. 3

[24] G. Ros, L. Sellart, J. Materzynska, D. Vázquez, and A. López. The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 3234–3243, Las Vegas, NV, USA, June 2016. 2

[25] A. Saxena, S. H. Chung, and A. Ng. Learning depth from single monocular images. In *Adv. in Neural Inform. Process. Syst. (NIPS)*, pages 1161–1168, Dec. 2006. 1

[26] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithm. In *IEEE Workshop on Stereo and Multi-Baseline Vision*, pages 131–140, Kauai, HI, USA, Dec. 2001. 1

[27] S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Fields and Service Robotics*, volume 5 of *Proceedings in Advanced Robotics*, pages 621–635, Nov. 2017. 2, 3

[28] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *IEEE Int. Conf. Comput. Vision Workshops (ICCV Workshops)*, pages 601–608, Barcelona, Spain, Nov. 2011. 1

[29] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ Int. Conf. Intell. Robots and Syst. (IROS)*, pages 573–580, Vilamoura, Portugal, Oct. 2012. 2

[30] O. Woodman. An introduction to inertial navigation. Technical Report 696, University of Cambridge, Aug. 2007. 3

[31] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *IEEE Int. Conf. Comput. Vision and Pattern Recogn. (CVPR)*, pages 340–349, Salt Lake City, UT, USA, June 2018. 1