# M4Depth: A motion-based approach for monocular depth estimation on video sequences

Michaël Fonder  
University of Liège  
Michael.Fonder@uliege.be  

Damien Ernst  
University of Liège  
DErnst@uliege.be  

Marc Van Droogenbroeck  
University of Liège  
M.VanDroogenbroeck@uliege.be  

## Abstract

Getting the distance to objects is crucial for autonomous vehicles. In instances where depth sensors cannot be used, this distance has to be estimated from RGB cameras. As opposed to cars, the task of estimating depth from on-board mounted cameras is made complex on drones because of the lack of constraints on motion during flights. In this paper, we present a method to estimate the distance of objects seen by an on-board mounted camera by using its RGB video stream and drone motion information. Our method is built upon a pyramidal convolutional neural network architecture and uses time recurrence in pair with geometric constraints imposed by motion to produce pixel-wise depth maps. In our architecture, each level of the pyramid is designed to produce its own depth estimate based on past observations and information provided by the previous level in the pyramid. We introduce a spatial reprojection layer to maintain the spatio-temporal consistency of the data between the levels. We analyse the performance of our approach on Mid-Air, a public drone dataset featuring synthetic drone trajectories recorded in a wide variety of unstructured outdoor environments. Our experiments show that our network outperforms state-of-the-art depth estimation methods and that the use of motion information is the main contributing factor for this improvement. The code of our method is publicly available on GitHub; see [https://github.com/michael-fonder/M4Depth](https://github.com/michael-fonder/M4Depth).

## 1. Introduction

Estimating reliable depth maps is an essential task for the planning of unmanned aerial vehicles (UAV) trajectories. However, lightweight, reliable for a wide range of depths, and energy efficient depth sensors for outdoor use do not exist to date. Distances between objects and the camera therefore need to be inferred rather than be measured. Except for some pathological configurations and when in presence of motion blur, depth can be inferred from vehicle motion and video stream of an on-board RGB camera. Theoretically, for a static scene, depth can be perfectly calculated by triangulation if the exact camera motion and the frame-to-frame displacement of each pixel are known, which is not achievable in practice.

Research has been carried to exploit recent progress in deep learning to infer depth directly from a sequence of images without having to rely on a physical model for the camera displacement. The results of these methods are encouraging on datasets and benchmarks created for autonomous car applications [7]. However, discarding all motion information has a major drawback; depth estimation methods then have to rely on some sort of semantic priors to produce their outputs, which makes them scene specific. In particular, the constrained camera motion and the particularities of an environment of autonomous cars applications reduces the complexity of the depth estimation task. It is therefore unsure if these methods would perform equally well for to a wider variety of motion types and environments such as the ones encountered during drone flights.

When using temporal information, depth estimation can
be seen as a triangulation problem. Mathematical algorithms for sparse point triangulation such as the one used for visual odometry include a frame-to-frame keypoint tracker and an iterative algorithm that performs the triangulation itself based on the displacement of both the camera and the keypoints [23]. Since triangulation is based on motion, the use of semantics is limited to keypoints detection and tracking. These algorithms are, however, computationally expensive when tracking a large number of points and are not able to take scene constraints into account due to their sparsity. This makes them poor candidates for dense depth estimation.

In this work, we look at the benefits that motion information can bring to depth estimation for drone applications when it is embedded inside of a Convolutional Neural Network (CNN) instead of being considered as an additional input. More precisely, we propose a new modular architecture that is designed to use motion and temporal information inside of the network.

The paper is organized as follows. First, we precisely formulate the addressed problem and review a number of the works published in the field of depth estimation. We then present our method, a modular deep CNN for end-to-end depth estimation that embeds a spatial re-projection layer and uses a time-recurrent feedback in each of its levels. Next, we present our experimental setup and analyze the performance of our method. For this, we establish a baseline comprising several state-of-the-art methods for depth estimation that are able to estimate a depth map for \( d \) from \( h_t \) as follows:

\[
\hat{d}_t = D(h_t),
\]

such that \( D \in \mathcal{D} \), with \( h_t = [I_0, [I_1, T_1], \ldots, [I_t, T_t]] \).

We want to find the function \( D \) in this set that best estimates \( d_t \) with respect to the ground truth \( d_t \). By defining our neural network, we implicitly select a subspace of \( \mathcal{D} \) in which the parameters for \( D \) will be searched for by gradient descent.

Several metrics were introduced by Eigen et al. [4] to assess the performance of a depth estimation method. Since we are considering autonomous vehicles applications, errors on the estimate for closer objects have a higher impact than the errors occurring for objects in the background of the scene. We therefore want to minimize the error relatively to the distance of the object. The RMSE log distance metric [4] has this property. We therefore search for the candidate \( D \) that minimizes this metric when comparing its outputs to the ground truth.

The network is trained in a supervised fashion on a dataset made of recorded drone trajectories for which every RGB frame of a trajectory comes with the corresponding ground-truth depth map and camera position. In accordance with good machine learning practices, the performance of the proposed architecture is assessed on a set of data that remains unknown to the training phase.

2. Problem Statement

Our problem statement is as follows. We consider a drone equipped with an RGB camera and an Inertial Measurement Unit (IMU) flying in an unseen environment that is assumed to be completely static. The camera is rigidly attached to the drone and its intrinsic parameters are supposed to be known and constant. We introduce the following components and notations:

- \( I_t \) is an RGB image of size \( H \times W \) recorded by the camera at timestep \( t \). Images are assumed to have the following properties: 1) motion blur and rolling shutter artifacts are negligible; 2) the camera focal length \( f \) is known and constant over a flight; 3) camera shutter speed and gain are unknown and can change over time.
- \( T_t \) is the transformation matrix encoding the motion of optical center of the camera from timestep \( t - 1 \) to \( t \). This matrix is assumed to be known.
- \( d_{ij,t} \) is the distance (in meters) of the surface recorded by the pixel at coordinates \((i, j)\) of the frame \( I_t \) to the camera along its \( z \) axis.

Using these notation, a depth map is defined as

\[
d_t = \{d_{ij,t} \mid i \in [1, H], j \in [1, W]\}.
\]

We denote by \( h_t \), the complete history of the information recorded by the drone up to timestep \( t \), from the beginning of a flight. We define a set \( \mathcal{D} \) of functions \( D \) that are able to estimate a depth map for \( d_t \) from \( h_t \) as follows:

\[
\hat{d}_t = D(h_t),
\]

3. Dataset

Our problem statement expresses the need for a dataset to explicitly perform the training and the testing of our method. A large majority of datasets providing RGB+D and motion data focus on ground vehicles (see [7, 22, 24]), which makes them unsuitable for our needs although most depth estimation methods are exclusively tested on them. Indeed, in ground vehicles datasets, images feature limited motion and the visual content is constrained. These datasets are not appropriate for our purpose since we target drone applications and the challenges that come with them, among which the exploitation of a 6 degrees of freedom (6DoF) motion and varied visual environments.

To the best of our knowledge, only two datasets provide data appropriate for drone applications, namely EuRoC MAV [1] and Mid-Air [5]. The former features recordings from real sensors and was entirely recorded in two rooms of an industrial building. The latter is a synthetic
dataset recorded in unstructured environments under varied weather and lighting conditions.

While the testing of our method on real data is potentially attractive, the EuRoC MAV dataset has two major drawbacks: (1) with only 22 minutes of recordings, it falls short compared to other larger datasets and (2), more importantly, the train and test data are recorded in the same place which makes it impossible to predict the performance for places never seen during training. On the other hand, with its 79 minutes of flight replicated in varying climate conditions, the Mid-Air dataset is much larger. Additionally, all trajectories are recorded in different places of the virtual environments which makes it much more suitable to train and test methods for robustness to previously unseen visual content. For these reasons, we have decided to validate our method on the Mid-Air dataset.

4. Related work

Being recent, the Mid-Air dataset currently lacks a proper baseline for depth estimation. In this section, we briefly review the state of the art in depth estimation and select a few methods that will compose our baseline on this dataset. The chosen methods are representative of different families detailed hereafter.

**Depth from a single picture.** Estimating depth from a single picture has a long history. Its main focus consists in on estimating depth from a single RGB picture. If the first methods were fully handcrafted with unsatisfactory results, the growth of machine learning and the development of CNNs has led to a massive improvement in the quality of the depth estimates. All current state-of-the-art methods are indeed based on CNNs, the main difference between them being the architecture and the used training procedure. As this field of research has been surveyed multiple times [8, 9, 18, 31, 34], we refer the reader to these surveys for comprehensively detailed descriptions of the methods.

Some general observations have been made in these survey papers which are worth noticing. Estimating depth from a single picture comes with a major drawback in general and especially for autonomous vehicle applications. Since the problem is ill-posed, networks have to heavily rely on priors to compute a suitable proposal. Such dependency on priors leads to a lack of robustness and generalization. Therefore, methods of this family need to be fine-tuned for every new scenario or environment encountered in order to be able to produce reasonable estimates.

A good method for single image depth estimation is Monodepth, a network proposed in [9]. We select it for our baseline since it has already been used as a baseline in the presentation paper of Mid-Air.

**Depth from an image sequence.** Attempts to use the temporal information featured in an image sequence for depth estimation are newer. Recent proposals [14, 20, 28, 33] have been made to include recurrence in the networks to make use of temporal information for improving estimates. Current proposals mainly replace certain layers by recurrent layers in existing CNN architectures.

As such, current depth estimation from sequence methods still badly use any kind of temporal information. Furthermore, the lack of input about the real camera motion makes these methods completely unsuitable for estimating the proper scale for depth without relying on semantic priors. Even worse, there is no proof that the scale of the outputs remains constant over the sequence. This is particularly problematic for autonomous vehicle applications.

We select the ST-CLSTM network proposed by [33] for our baseline because this method achieved state-of-the-art performances in this category on the KITTI dataset [7] when it was published and because there exists an implementation of it on GitHub [30].

**Use of motion information.** When motion is used by depth estimation methods, it is exclusively exploited to build a loss function for unsupervised training [8, 15, 17, 20, 28]. In these cases, motion is learnt by a separate network, also in an unsupervised fashion, and depth is still estimated without any direct motion information. As the core of the depth estimation networks does not change for these methods, estimations suffer from the same issue as their counterparts that do not use motion.

One notable exception to these observations is the idea proposed by [16]. This method uses the unsupervised loss based on motion estimation to fine-tune the network at test time. This method features outstanding performance, although they are achieved at the cost of a large computational burden. Furthermore, due to its architecture, this method cannot estimate depth before the whole video sequence is available. It is therefore not designed to run in an online fashion, which makes it inappropriate for autonomous vehicle applications.

For this category, we use Monodepth2 [8] and the method proposed by Wang *et al.* in [28] as baselines. The former is built upon Monodepth [9] while the latter proposes a supervised method showing promising results. Additionally, the code for both methods was released by their respective authors.

**Structure from motion.** Structure from motion is a research field that developed in parallel with depth estimation. The idea here consists in reconstructing 3D shapes from a batch of 2D pictures that capture the scene to reconstruct with different points of view. Reconstruction is achieved by explicitly modeling the relative camera position between the pictures of the set. Approaches for performing this task are varied [19] and are, by their nature, not all suitable for real-time depth estimation. Some of them however appear to be compatible for depth estimation on sequences [27, 32] while others were specifically designed to work on image
The blue convolution blocks encode the image while the orange ones are in charge of producing depth estimates.

The approaches proposed by [3] and [26] are similar. They both propose a three-stage network. Their stages are an image-encoding network followed by the computation of a cost volume that is finally processed by a depth estimation network. The purpose of the cost volume consists of providing the costs for matching a point in an image with a series of candidates in another image. The cost volume in both [3] and [26] is built by a plane-sweeping method [2, 6]. This relies totally on a precise relative camera positioning. Any imprecision in the camera positioning will skew the cost volume and severely alter the depth estimation that follows.

While the multi-stage nature of these two methods makes them arduous to train, they present the major advantage of estimating depth without directly using any semantic cues. This naturally makes them resilient to visual novelties. They would have made an interesting addition to our baseline, but their extremely long training time (one whole week according to [26]) convinced us that it would be hard to get their performance on Mid-Air in a reasonable period of time.

5. Our M4Depth method

Similar to the ideas developed for structure from motion, we base our method on a notion of cost volumes generated by using motion information to decouple depth estimation from semantic information. However, we use an approach different than the plane-sweeping method [2, 6] to build them. Instead of using a multi-stage network, we propose to build a lighter multi-level architecture that can be trained in an end-to-end fashion. In this section, we detail our complete architecture and the loss function used for training.

5.1. Network definition

Depth estimation can be seen as a point triangulation problem when camera motion information is paired with the video sequence. Approaching depth estimation as such allows for a reduction in the dependency of estimates on semantic priors. Point triangulation is inherently an iterative process. Instead of simply iterating on a full network as proposed in [27] or bypassing the iteration by proposing a full range of candidates as in [3, 26], we propose approaching the iterative process in a similar manner as the PWC-Net [25].

PWC-Net [25] is a neural network that was developed for optical flow. Optical flow is defined as the pattern of apparent motion of image objects between two consecutive frames caused by the movement of these objects and the camera. It takes the form of a 2D vector field where each vector is a displacement vector showing the movement of points from the first frame to the second one. Methods capable of recovering the optical flow are, by definition, able to perform frame-to-frame dense point tracking.

As for point triangulation, optical flow estimation can be approached as an iterative process. In PWC-Net, the iterative process is embedded in the architecture itself. The architecture is a modified version of the U-Net architecture with skip connections introduced by [21] where each level of the decoder has to produce an optical flow estimate. This estimate is used to compute a cost volume based on the feature maps produced by the encoder for the two frames between which the optical flow has to be computed. The cost volume and the estimate are then passed to the next level of the decoder whose purpose consists of refining the estimate. The levels of this architecture are generic and can be stacked at will to obtain the desired network depth.

Based on the idea of PWC-Net, our approach consists of a modified version of the U-Net architecture, where each level in the decoder part of the network has to refine the depth estimate produced by the level preceding it in the architecture. We use the same encoder as PWC-Net, our contribution is located in the decoder part of the network.

At its input, each level of the decoder receives information from current and previous time steps. This information is first ingested by a preprocessing unit that has it converted in such a way that allows the decoder to focus on the refinement of the depth estimate it receives. This preprocessing unit does not contain any learnable parameters. Once processed, the data is fed to a small convolutional subnetwork whose purpose is to estimate depth from the data it receives. We give a block overview of our architecture in Fig. 2.

The operations performed by a preprocessing unit are further detailed in Fig. 3. The first operation it performs consists in using the depth estimate and the motion information to spatially reproject the feature and depth maps inherited from the previous time step. This allows one to spa-
volitionally realign data according to the depth estimate. If the estimate is imprecise, the realignment will not be perfect. A cost volume is then computed between the spatially reprojected feature map and the feature map coming from the encoder for the current frame. Finally, all the information that will be fed to the learning part of the decoder is concatenated.

Let us now describe the building blocks of our architecture and motivate their design.

Reprojection Layer. This layer does not have any learnable parameters. It spatially moves vectors of a feature map to some new location on the sensor plane, the one to which the spatial point encoded by each vector would be projected when seen from a different point of view by the camera. In our architecture, we reproject vectors of the previous timestep to their new location in the current frame.

The camera motion between consecutive frames is encoded by the transformation matrix $T_t$, and can be broken down in a rotation matrix $R_t$ and a translation vector $t_t$:

$$T_t = [R_t|t_t].$$

Taking into account this camera motion $T_t$ and for the pinhole camera model, a point in space projected at coordinates $(i_t, j_t)$ in the current frame $t$ will be linked to its previous coordinates $(i_{t-1}, j_{t-1})$ in frame $t - 1$ by the following relation:

$$d_{i,j}(i_{t-1}, j_{t-1}) = K R_t \begin{bmatrix} (i_t - c_x)/f \\ (j_t - c_y)/f \end{bmatrix} d_{i,j}(i_t, j_t) + t_t,$$

where $K$ is the $3 \times 3$ camera intrinsic matrix, $f$ the camera focal length, and $(c_x, c_y)$ the coordinates of the principal point of the camera.

Moving the vectors across the frame is performed trough a warping operation using bilinear interpolation, as described in [25]. For the $l$-th level of our architecture, the warping of a feature map $f$ can be expressed as follows:

$$f_{l}^{t}(i_{t}, j_{t}) = f_{l}^{t}(i_{t-1}, j_{t-1}),$$

where the coordinates $(i_{t-1}, j_{t-1})$ are computed using Equ. 4 and the upscaled depth map produced by the previous level, $d_{l}^{t+1}$. Any imprecision on $d_{l}^{t+1}$ will lead to an imprecise spatial reprojection. When detected properly, these reprojection inaccuracies can be used to detect and correct inaccuracies in the depth estimate.

As such, this layer leads to gradient explosions during training. Instabilities are due to the gradients computed on the depth map used for computing the reprojection. To prevent this issue and to allow the network to train properly, we do not propagate the gradients of the reprojection layer to the depth map.

Cost Volume Layer. This layer does not have any learnable parameters. Its purpose consists in detecting any reprojection imprecision induced by the reprojection layer. For this, it builds a volume where each value represents the cost of matching two given feature vectors. The cost of matching two feature vectors $x_1$ and $x_2$ is defined as their correlation:

$$\text{cost}(x_1, x_2) = \frac{1}{L} x_1^T x_2,$$

where $L$ is the length of the feature vectors. The volume is built by computing this cost for each vector of a feature map $f_1$ with all vectors located within a spatial neighbourhood $r$ in another feature map $f_2$ of same dimensions:

$$\text{cost}_v(f_1, f_2, r)(i, j) = \{ \text{cost}(f_1(i, j), f_2(i + k_i, j + k_j)) \} \forall k_i, k_j \in \{-r, \ldots, r\}.$$

When computed for feature maps of two successive frames, the cost can be used as a similarity metric for matching points between these two frames and therefore detect potential misalignements left by the reprojection layer. The purpose of each level of our architecture being to correct the residual depth error, we assume that matching feature vectors are located in each others neighbourhood after reprojection. For each level $l$ of our architecture, we compute a cost volume for a neighbourhood of 4 pixels:

$$\text{cv}_l = \text{cost}_v(f_{\text{enc}_l}, f_{\text{enc}_{l-1}}|u, 4).$$

It is important to note that a range $r$ at the $l$-th level of the architecture will correspond to a range of $2^l r$ in the input image due to the reduction of the spatial dimension induced by the image encoding layers.

As our cost volume is built by taking into account the entire neighbourhood of a pixel, it is robust to potential reprojection errors that could occur because of imprecision in motion information. Corresponding vectors in consecutive frames are, indeed, likely to be matched together despite an
error on motion information with our architecture design. The same cannot be said for cost volumes built based on the plane-sweeping method. When compared to the plane-sweeping method, our method however has one drawback: as opposed to the ones built with the former, matches in our cost volumes do not provide direct information on the corresponding depth. The position of a match has to be combined with motion information and previous depth estimates to derive information about depth.

**Depth Estimator.** This sub-network consists of seven convolutional layers in charge of producing a depth estimate. The inputs of the depth estimator are the feature map of the current image, the cost volume \(cv^l_t\), the wrapped depth estimate produced at the previous time step \(d^w_{t-1,ij}\), and the upscaled depth estimate produced by the previous level of the architecture. Additionally, each pixel is given its grid coordinates on the sensor plane and camera motion information.

The convolutional layers have the same number of filters for all levels of the architecture \(128, 128, 128, 96, 64, 32, 1\) and are followed by a leaky ReLU activation unit, excepted the last one that remains deactivated. All convolution filters have a kernel size of \(3 \times 3\) and a stride of 1. In order to ease the convergence, depth maps are encoded in a log space at the input of the sub-network. The depth estimator is taught to produce its output in a log space.

5.2. Loss function definition

Since the levels of our architecture are stackable at will, the architecture can have any depth. In the following paragraph, we detail our loss function for a network that is made of \(M\) levels.

We could train this network by using our performance metric directly as a loss function. It is however possible to design a loss function that features better convergence properties. In order to train such a deep architecture efficiently, we use a custom loss function that aggregates several weighted loss terms.

For each frame in a sequence, we compute the \(L1\) distance on the logarithm of the depths for the estimate produced by each level. Doing so eases the training of the deepest layers in the architecture. Since intermediate depth maps have a lower resolution, we resize the ground truths by bilinear interpolation to match the dimensions of the estimates. The resulting terms are aggregated together through a weighted sum:

\[
L_{1t} = \frac{\alpha}{HW} \sum_{l=1}^{M} \sum_{d_{ij} \in d^l_t} 2^{l+1} |\log(d_{ij}) - \log(d^l_{ij})| ,
\]

where \(\alpha\) is a weighting parameter. We set it arbitrarily to 0.64.

We use this loss function to define the total loss for the complete sequence. For a sequence on length \(N\), the total training loss for our depth estimation network is computed as follows:

\[
L_{\text{tot}} = \frac{1}{N} \sum_{t=1}^{N} (L_{1t}) + \gamma |\Theta|_2 ,
\]

where \(|\Theta|_2\) is an L2 regularising term on the weights of the parameters in network and where \(\gamma\) is the weighting parameter of this regulariser.

6. Experiments

6.1. Experimental setup

In the next paragraphs, we detail the dataset settings used for testing our architecture and give the complete set of specifications of the training parameters for our network.

**Dataset.** With the test set proposed in the original paper [5] being quite small, we propose an alternate train/test split. In order to guarantee a representative test set, we divide the train set proposed in the original paper as follows: trajectories whose number is a multiple of 3 (including the 0 one) are allocated to the test set. This de facto creates a two-third/one-third split of the original train set for our train/test sets.

We subsample the frame rate by a factor 4 (from 25 to 6.25 fps) to increase the motion experienced by the camera between two frames, and we divide the long trajectories in non-overlapping sub-sequences of 8 frames in both the train and test sets. This results in 8,704 trajectories for training and 4,352 for testing. For all our experiments in this work, we resize the pictures and the depth maps to a size of \(384 \times 384\) pixels. We use bilinear interpolation to resize color images and the nearest neighbour method for depth maps.

**Training.** With everything else properly defined, we can detail our training procedure. We use a He initialization [12] for our variables and an Adam optimizer [13] for the training itself. We keep the default parameters proposed by both of them. All our trainings are performed on GPUs with 12 GB of VRAM with a batch size of 3 sequences and for 200 k iterations. For the learning rate, we begin the training process by setting it to \(10^{-4}\). We then divide it by 2 every 60 k iterations. Depending on the network parameters, a full training can require up to 40 hours to complete. In order to assess the raw capabilities of our method, no data augmentation was used during training.

6.2. Baseline

With the Mid-Air dataset being relatively new, there are currently no performance reports for depth estimation methods on it. A baseline is however required in order to assess the benefits of our approach. We motivated a selection of candidate methods to use as references in section 4. As a
In order to assess the use of temporal information by our network, we perform an additional set of tests. We unroll our network over several numbers of time steps for training and report their performance on the test set for different sequence lengths. To ensure the coherence of the obtained scores, we compute the performance only for the last image of each of the 4,352 test sequences. When testing a network on a sequence of length $N$, we therefore give it the last $N$ frames of each test sequence.

The results of this experiment are reported in Table 2. They show that temporal information is important to get good performances. Networks working on sequences perform far better than the ones working with a single image. Performances get marginally better as the length of sequences increases. They however reach a plateau passed a sequence length of 4 frames. This could indicate that our architecture is limited in its ability to use recurrent information. Additionally, increasing the length of the training sequences does not seem to lead to better performances.

At inference time and without taking data loading operations into account, an NVidia Tesla V100 GPU needs 29 ms to process a single frame for a non-optimized TensorFlow implementation of our network with 6 levels. This corresponds to 34 frames per second and therefore already meets real-time constraints. In its current implementation, our network requires 808 MB of memory to run.

### 6.4. Discussion

The gap between performances of our method and that of the baseline raises a fundamental question. Since some baseline methods also work on video sequences in a supervised fashion [28, 33], why does our model perform so well compared to them? In the next paragraphs, we propose some explanations.
Figure 4: Qualitative comparison of the depth estimates produced by M4Depth with 6 levels with the ones produced by baseline methods. Even if its outputs lack details in areas with a lot of abrupt depth transitions, our method is globally able to recover details and depth much more accurately than baseline methods, even for challenging environments such as forests.

In the methods we chose for our baseline, two of them work on sequences. They should therefore perform better than the other ones. This intuition is verified on the KITTI dataset. The performances reported in the original papers show a clear gap between the different methods, with the ones predicting depth from a single image achieving the worst scores. However, we observe that three out of the four baseline methods achieve almost the same performance when compared on the Mid-Air dataset.

A reasonable hypothesis is that current recurrent methods work well for constrained motion, but fail to properly use temporal information for more varied motion. When compared to a drone flight for which the camera motion has up to 6DoF, the motion of a camera mounted on the rooftop of a car is much more constrained. A car driving on a flat road has indeed only 3 degrees of freedom in practice, 2 for translations in the horizontal plane and one for rotations around the vertical axis, everything being constrained by the physics of the car.

This hypothesis is supported by the scores shown in Table 2. Our network, when deprived of temporal information, performs similarly to the baseline. This further supports the idea that the ability of current recurrent networks to properly use temporal information on Mid-Air is limited and highlights the benefits motion information can bring to the quality of the estimates.

7. Conclusion

In this work, we presented a new method, named M4Depth, for performing depth estimation on sequences recorded by a camera whose motion has up to six degrees of freedom. Our method uses motion information as well as time recurrence to produce its estimates. We made this possible by embedding spatial feature reprojection layers at each level of our architecture. Our network is relatively lightweight, runs in real time, and can be trained in an end-to-end fashion.

In order to analyse the performance of M4Depth, we have established a baseline on Mid-Air, a drone dataset, by retraining state-of-the-art depth estimation methods on it. Results show that M4Depth substantially outperforms the baseline, even in its most basic form. Our experiments show that the use of temporal information is the key contributing factor for this performance improvement.

Despite this improvement, we have noticed that our architecture has a limited ability to exploit the whole trajectory history. In the future, we will investigate alternative architectures that could better use this history. Also, we will test if data augmentation is useful for M4Depth and possible variants.
A. Camera model and geometry

Our work relies on some hypotheses on the optics of the camera. In this section, we proceed to detail the mathematical model used for our camera.

![Pinhole camera model visualization](image)

We use the pinhole camera model. With this model the camera is simply represented by a sensor plane and a focal point, the optical centre of the camera, that is used as the camera origin, as illustrated in Fig. 5. In a perfect camera, the focal point is located at some point on the principal axis, the axis intersecting the sensor plane perpendicularly on its central point. In computer vision, the distance separating the focal point from the sensor plane is called the focal length and is expressed as a multiple of a sensor pixel width.

A pinhole camera is fully characterised by 5 intrinsic parameters that are:

- \( f_x \) and \( f_y \), the focal lengths along the x and y axis, respectively;
- \( s \), the skew factor of a pixel;
- \((c_x, c_y)\), the coordinates of the principal point on the camera sensor

In a perfect camera pixels are square. In this case, \( f_x \) and \( f_y \) are equal to each other and \( s \) is equal to zero. In a perfect camera, the principal point is located at the centre of the sensor. These parameters can be assembled in a projection matrix \( K \) as follows:

\[
K = \begin{bmatrix}
    f_x & s & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}.
\]  

In practice, the 3D coordinates of a point are rarely expressed within the camera frame of reference. It is more common to know the position of a point with respect to a given frame of reference and to know the camera pose with respect to the same frame of reference. If we express the camera position by a vector \( p \) of size 3 and the camera orientation by a 3 \times 3 rotation matrix \( R \), the position \((X, Y, Z)\) of a point in space can be expressed with respect to the camera frame of reference by applying the following transformation:

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = \begin{bmatrix}
    R^T & -R^T p \\
    0 & 1
\end{bmatrix}
\begin{bmatrix}
    X \\
    Y \\
    Z
\end{bmatrix},
\]  

where \( T \) denotes the matrix transpose operator. In this equation, \( T \) is referred as the transformation matrix.

B. Reprojection layer mathematical details

In this section, we further detail and motivate our reprojection layer. The reprojection layer involves two distinct poses of the same camera. We denote these poses by \( C_1 \) and \( C_2 \). \( C_2 \)'s pose is expressed relative to the pose of \( C_1 \) and is encoded by the transformation matrix \( T_2 \) with:

\[
T_2 = \begin{bmatrix}
R & t \\
0 & 1
\end{bmatrix}.
\]  

Let us assume that some visual information is encoded for each pixel of \( C_1 \) and that this information is not directly available to \( C_2 \). A point \( P \) seen by \( C_1 \) and projected on the sensor plane in \((i_1, j_1)\) has a high probability of also being seen by \( C_2 \), but is projected in a different location on the sensor, \((i_2, j_2)\). When \( P \) is visible to both cameras, the information encoded for this point should be similar for both camera poses. The purpose of the reprojection layer is to transfer the encoded information from \( C_1 \) to \( C_2 \) by using the geometrical constraints of the system.

To be able to transfer information from one projection to the other, we first need to find the relation that links \((i_1, j_1)\) and \((i_2, j_2)\). From Eq.12, it can be seen that recovering the full 3D coordinates of a point whose projection coordinates \((i, j)\) and depth \( z \) are known is trivial if the intrinsic matrix is known (which is often the case in computer vision). Assuming that \( P \) is located at a depth \( d_2 \) of \( C_2 \), its 3D coordinates \((X, Y, Z)\) can be expressed in the camera referential are obtained by using the camera intrinsic matrix \( K \) and the right-angle theorem as follows:

\[
(i, j) = \left( \frac{\alpha}{z}, \frac{\beta}{z} \right) \quad \text{with} \quad \alpha = K_{11} x + K_{12} y + K_{13} \quad \text{and} \quad \beta = K_{21} x + K_{22} y + K_{23}.
\]

In practice, the 3D coordinates of a point are rarely expressed within the camera frame of reference. It is more common to know the position of a point with respect to a given frame of reference and to know the camera pose with respect to the same frame of reference. If we express the camera position by a vector \( p \) of size 3 and the camera orientation by a 3 \times 3 rotation matrix \( R \), the position \((X, Y, Z)\) of a point in space can be expressed with respect to the camera frame of reference by applying the following transformation:

\[
\begin{align*}
x &= X \\
y &= Y \\
z &= Z
\end{align*}
\]

where \( T \) denotes the matrix transpose operator. In this equation, \( T \) is referred as the transformation matrix.
This leads to an issue for applying Equ. 18.

As a reminder, we want to transfer information from $C_1$ to $C_2$. This means that the sensor plane of $C_1$ is populated with the information that is to be transferred. In such a configuration, the information on the sensor plane can be made continuous by using interpolation. Indexing the sensor plane of $C_1$ with coordinates defined in the real domain therefore becomes possible and Equ. 18 can be used without any modification.

In summary, our reprojection layer computes coordinates $(i_1, j_1)$ from $(i_2, j_2)$ and a depth estimate $d_2$, samples the information located at these coordinates on the sensor plane of $C_1$ by using bilinear interpolation, and then copies it to the coordinates $(i_2, j_2)$ for $C_2$. In practice, in our architecture, we respectively use the notations $C_{t-1}$ and $C_t$ instead of $C_1$ and $C_2$.

### References


