Robust Learning of a depth map for obstacle avoidance with a monocular stabilized flying camera

Clément Pinard

université PARIS-SACLAY

Parrot

ENSTA ParisTech université PARIS-SACLAY

NNT : 2019SACLY003

Thèse de doctorat

Apprentissage robuste d'une carte de profondeur pour l'évitement d'obstacle dans le cas des caméras volantes, monoculaires et stabilisées

Thèse de doctorat de l'Université Paris-Saclay
préparée à Ecole nationale supérieure de techniques avancées

Ecole doctorale n°573 Approches Interdisciplinaires: Fondements, Applications et Innovation (INTERFACES)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 24 juin 2019, par

CLÉMENT PINARD

Composition du Jury :

Yann Gousseau Enseignant-Chercheur, Télécom Paristech (TSI)	Président
Nicolas Thome Professeur, CNAM (CEDRIC)	Rapporteur
Patrick Pérez Directeur de recherche, Valeo.ai	Rapporteur
Pascal Monasse Enseignant-Chercheur, ENPC (IMAGINE)	Éxaminateur
Laure Chevalley Ingénieur, Parrot	Co-encadrante
David Filliat Directeur de recherche, ENSTA (U2IS)	Co-encadrant
Antoine Manzanera Enseignant-Chercheur, ENSTA (U2IS)	Directeur de thèse

Acknowledgements

Ce document clôt trois ans (et demi!) de thèse pendant lesquels j'ai pu travailler à la fois au sein du laboratoire U2IS de l'ENSTA et de l'équipe Computer Vision de Parrot. Cette thèse, dans le cadre d'une convention CIFRE a pour moi été très fructueuse, et je suis très fier d'avoir participé à ce projet.

Je remercie ainsi premièrement les gens sans qui le projet de thèse n'aurait pas pu se faire, à savoir l'entreprise Parrot, à travers Benoît Pochon et Laure Chevalley, l'ENSTA, à travers David Filliat et Antoine Manzanera, et enfin l'ANRT grâce à qui j'ai pu profiter de 4 mois de vacances forcées en attendant leur accord pour le début du contrat CIFRE. Je remercie d'autant plus Antoine, David et Laure qui m'ont accompagné et dirigé durant cette thèse, en réalisant le tour de force de s'intéresser à mes travaux, s'y investir tout en me laissant particulèrement libre dans mes choix de recherche. Merci à eux aussi pour leurs nombreuses relectures d'articles et de ce manuscrit. Cette thèse, c'est un peu la votre aussi!

Dans la suite logique, après qui ont permis à cette thèse de commencer, merci à ceux qui ont permis son dénouement, en constituant mon jury de thèse. Merci donc à Patrick Perez et Nicolas Thome d'avoir accepté d'être mes rapporteurs, et merci à Pascal Monasse et Yann Gousseau d'avoir accepté d'être examinateurs. Vos différentes remarques et objections ont permis à ce manuscrit de grandement gagner en qualité, en lisibilité et en rigueur scientifique.

Je remercie également les gens avec qui j'ai eu l'occasion de travailler de près ou de loin durant ces trois ans, à Parrot et à l'ENSTA, grâce à qui j'ai pu garder un pied dans la recherche et un pied dans l'industrie, et que je peux compter non seulement dans mes collegues mais aussi dans mes amis.

Je remercie d'ailleurs tout particulièrement à Parrot:

- Léa Vauchier et Alexandre Briot, avec qui la meilleure équipe spécialisée en machine learning au monde a été créée en 2016

- Louis-Joseph Fournier et Edouard Leurent, qui ont dévelopé la premiere version de l'algorithme d'évitement d'obstacle à partir d'une profondeur et Aurélien Barre, qui a développé la librairie de streaming Vidéo à Parrot. C'est grâce à eux que la preuve de concept finale a pu être mise en place, et que la boucle a pu être bouclée.

- Mylène Tahar, Antoine Tran et Fabien Remond, compagnons de jeu de qualité, le midi devant un plateau comme le soir devant une Nintendo Switch.

- Toute l'équipe Parrot Vision dirigée avec brio par Laure, pour ses discussions scientifiques inspirantes, je leur souhaite de voir leurs différents projets d'étude se concrétiser : Nicolas Renard, Vincent Rapp, Simon Fauchard, Vladyslav Gorbatiuk, Corentin Lacroix, Sergio Rodriguez, Aleksandr Liadov , Qiang Zhang, Denis Giri

- Mais aussi l'ensemble des ingénieurs de Parrot avec qui j'ai passé 4 ans et demi formidables, dont Martin de Gourcuff, Gauthier Rousseau, Mathieu Babel, Romain

Lozachmeur, Marie-Amélie Cousin, Nicolas Texier le roi des Connemaras, Eric Simon, Martin Liné

- Enfin, Aurélia Bouvier pour sa patience, malgré mon incompétence administrative que je lui ai fait subir.

Je remercie tout autant les membres du laboratoire de l'U2IS à l'ENSTA, avec qui j'ai pris plaisir à discuter, prendre des café, préparer des cadeaux de thèse, mais aussi faire un peu de recherche! Merci en particulier à

- Louise Sarrabezolles, Antoine Tran (encore lui !), Pierre-Henri Oréfice, Adrien Matricon, Clément Masson, Céline Craye, Pauline Chevalier, Antonin Raffin, pour les soirées jeux, les restos, ou tout simplement les moments de convivialité passés ensemble.

- Mes différents co-bureaux dans l'ordre chronologique : Gennaro Raiola, Yvon Kerdoncuff, René Traoré, Ashley Hill, Hugo Caselles-Dupré qui ont mis une ambiance de folie en R213 quand je n'étais pas là.

- David Iacob, Moad Kissai, Roxana Agrigoraie, Natalia Diaz, Timothée Lesort, Julien Alexandre Dit Sandretto, Olivier Mullier, Thibault Toralba, Florence Carton, Alexandre Chapoutot, Goran Frehse.

Merci ensuite à tous mes amis et leur présence à mes côtés du ces trois années, et notamment, les amis de Supélec, les rageux du CG2, la plus belle bande d'idiots que je connaisse, les amis de Stan, de Bourges et de la bourgade de Cosne sur Loire.

Merci à mes parents, mes soeurs et conjoints, Solenne et Irène pour leur aide, notamment pour l'organisation du cocktail de soutenance qui fut très réussi grâce à eux.

Merci plus généralement à toute ma famille, mes connaissance et tous les facteurs extérieurs, qui ont ont contribué à faire de moi la personne libre et curieuse que je suis et qui ont rendu ce travail possible.

Je finis cette montagne de gratitude en remerciant infiniment une certaine connaissance, prénommée Solenne Rezvoy, qui est, comme le disait si bien le philosophe Franck Dubosc, "Une bouteille d'oxygène : toujours sur mon dos, mais indispensable à ma survie". Merci pour ton énergie et ton dévouement durant ces deux années et demi à vivre ma thèse avec moi. Tu as été une de mes inspirations pour donner le meilleur de moi-même pour ce projet comme pour mes projets futurs.

Contents

1 Introduction **1**
1.1 Motivations of this thesis . . . 1
1.1.1 Anatomy of a consumer Drone . . . 2
1.1.2 The problem of obstacles avoidance . . . 2
1.2 Depth perception for navigation in robotics . . . 4
1.2.1 Considered sensors for depth perception . . . 4
1.2.2 Active sensors advantages and limitations . . . 5
1.2.3 Vision based depth perception . . . 6
1.3 Deep Learning Approach . . . 7
1.3.1 A brief historic . . . 7
1.3.2 Deep learning with convolutional neural networks for computer cision 8
1.3.3 Deep learning for decision making . . . 9
1.3.4 Interest of deep learning for consumer UAVs . . . 9
1.4 Approach and goal of the thesis . . . 10
1.4.1 Hardware setup . . . 10
1.4.2 Discarding the naive solution . . . 10
1.4.3 Training with increasing complexity . . . 11
1.4.4 Resulting scope of this thesis . . . 12
1.5 Contributions . . . 12
1.6 Outline of this thesis . . . 13

2 Depth estimation from a camera **15**
2.1 Linking depth with reality: the scale factor problem . . . 15
2.2 Depth from motion vs depth from context . . . 16
2.2.1 Depth from context advantages and limitations . . . 16
2.2.2 Depth from motion advantages and limitations . . . 18
2.2.3 Our strategy . . . 18
2.3 Obstacle avoidance oriented depth quality . . . 19
2.3.1 Scale invariant estimation . . . 19
2.3.2 Obstacle avoidance requirements . . . 19
2.4 Anatomy of an error function . . . 20
2.4.1 Standard error vs mean error . . . 21
2.4.2 Usual error functions . . . 22
2.5 Conclusion on error measures . . . 24

3 Creating a depth map with a convolutional neural network **25**
3.1 Geometric definitions . . . 26
3.1.1 Camera plane . . . 26
3.1.2 Pinhole camera model . . . 26
3.1.3 Moving camera . . . 27
3.1.4 Optical flow and disparity . . . 28

3.1.5 Focus of expansion . . . 29
3.1.6 Finding depth from optical flow . . . 30
3.2 Deep learning for optical flow . . . 30
3.2.1 FlowNet . . . 31
3.3 Optical flow for depth generation . . . 32
3.3.1 From disparity to depth . . . 32
3.3.2 From flow divergence to depth . . . 33
3.3.3 Conclusions on limitations . . . 34
3.4 End-to-end learning of depth generation . . . 34
3.4.1 Introducing StillBox, a depth synthetic dataset . . . 35
3.4.2 Dataset set augmentation . . . 37
3.5 DepthNet, a depth fully convolutional neural network . . . 37
3.5.1 The constant speed assumption . . . 37
3.5.2 On the optimal network output . . . 37
3.5.3 General structure . . . 38
3.6 Training details . . . 39
3.6.1 Influence of resolution on training . . . 39
3.6.2 Comparison with DeMoN and depth from optical flow . . . 44
3.7 UAV navigation use-case . . . 45
3.8 Conclusion of this chapter . . . 48

4 Training Depth estimation with auto-supervision 49
4.1 Motivations . . . 50
4.1.1 Training depth networks without ground-truth . . . 50
4.1.2 Outline of this chapter . . . 51
4.2 Core concept for reprojection based optimization . . . 51
4.2.1 The image as a continuous 2D function . . . 51
4.2.2 Optical flow and reprojection based optimization . . . 53
4.3 Retrieving a depth map using reprojection based optimization . . . 54
4.3.1 Training a neural network to output depth with reprojection based optimization . . . 55
4.3.2 Conclusions of reprojection based optimization . . . 56
4.4 Regularization with smooth loss and diffusion . . . 56
4.4.1 Optical flow smoothing . . . 56
4.4.2 Applying smoothing on a depth map . . . 57
4.4.3 Translating physical assumptions into equations . . . 58
4.4.4 Smooth loss presentation . . . 59
4.4.5 Diffusing the output of a neural network . . . 60
4.4.6 Minimizing gradient with $\mathcal{L}_{s\nabla}$ . . . 62
4.4.7 Minimizing Laplacian with $\mathcal{L}_{s\Delta}$ . . . 63
4.4.8 Diffusion of a vector map with divergence minimization . . . 64
4.4.9 Edge-aware smooth loss . . . 65
4.4.10 Final edge-aware forms of considered smooth losses . . . 67
4.5 Dealing with occlusions . . . 68
4.5.1 Inverse warp based optimization *vs* occlusions . . . 68
4.5.2 Filtering occlusion areas from optimization . . . 70
4.5.3 Self filtered depth map inference . . . 70
4.5.4 Backward/forward consistency . . . 71
4.5.5 Depth map consistency . . . 71
4.5.6 Occlusions detections with direct warp instead of inverse warp . . . 72
4.5.7 Conclusions on occlusion considerations . . . 75
4.6 Determination of optimal hyper-parameters with toy problems . . . 77

4.6.1 Scenes presentation . . . 77
4.6.2 Tests on smooth loss regularization . . . 81
4.7 Photometric losses . . . 86
4.7.1 Multi-scale photometric loss . . . 86
4.7.2 Minimal loss sampling . . . 89
4.7.3 Photometric distance . . . 90
4.8 Conclusions on literature review for unsupervised depth training . . . 93

5 Unsupervised DepthNet 95
5.1 Shortcomings of SFMLearner . . . 95
5.2 Workflow presentation . . . 96
5.2.1 Pose estimation . . . 97
5.2.2 Frame stabilization . . . 97
5.2.3 Depth computing and pose normalization . . . 98
5.2.4 Loss functions . . . 98
5.3 Training datasets . . . 98
5.4 Experiments . . . 99
5.4.1 Qualitative results . . . 99
5.4.2 Implementation details . . . 99
5.4.3 Testing methodology for DepthNet . . . 99
5.4.4 Quantitative results . . . 100
5.4.5 Domain adaptation results . . . 105
5.5 Conclusion on unsupervised DepthNet . . . 105

6 DepthNet in the wild 107
6.1 Simplifying the scale factor determination by normalizing DepthNet . . . 107
6.2 Optimal frame shift determination . . . 108
6.3 Multiple shifts inference . . . 109
6.4 Training a specific depth range with a clamped DepthNet . . . 113
6.5 Our proof of concept . . . 113
6.5.1 Model Predictive Control . . . 113
6.5.2 Implementation details . . . 113
6.6 Conclusions for this chapter . . . 115

7 Perspectives and Conclusions 117
7.1 Limitations and future perspectives . . . 117
7.1.1 UAV depth validation dataset . . . 117
7.1.2 Robust smooth and photometric loss . . . 118
7.1.3 Moving scenes . . . 118
7.1.4 Bayesian depth inference . . . 119
7.2 Alternative Perspectives . . . 119
7.2.1 Depth sensing for multi-task learning . . . 119
7.2.2 Robust and evolutive passive depth sensing . . . 119

A Illustration of compensating effect of linking depth scaled factor to movement estimation 123

B Proof on under-estimated depth over-representation when optimizing relative error for a validation set 125

C Proof on smoothing an inverse depth map 127

D A Collection of Proofs on diffusion **129**
D.1 Preamble 129
D.2 Diffusion stability 130
D.3 Diffusion of Inverse 130
D.4 Diffusion of Gradient 131
D.4.1 Laplacian minimization 131
D.4.2 Divergence minimization 133
D.4.3 Isotropic gradient diffusion 133

Bibliography **135**

List of Figures

1.1 Consumer products featuring camera stabilization. From left to right: DJI Phantom (mechanic gimbal), Parrot Bebop (dynamic cropping from fish-eye), DJI Osmo mobile (mechanic handheld guimbal)[1] 3
1.2 Parrot's current Flight Plan interface 3
1.3 DJI's ActiveTrack interface 3
1.4 A "jerky" trajectory (red) and a smoother one (green) 4
1.5 Example of RGB-D data, taken from[Lai+11]. For depth (right), bright colors indicate lower depth (and thus closer object to the camera plane) 5
1.6 Example of a stereo camera placed on a UAV in order to compute a depth map (here, a Parrot Slam Dunk on top of a Bebop2) 7
1.7 First simulation of a neuron, $x_0 \cdots x_n$ are the inputs, while $w_0 \cdots w_n$ and α are changeable parameters.[2] 8
1.8 Graph of milestones complexity regarding domain and task complexity. Arrows indicate a possible reutilization of networks with a fine-tuning fashion. Red points indicate considered milestones in this thesis. 11

2.1 Illustration of scale ambiguity. Aerial view of Chambord Castle and 1/30-scale model replica at *France Miniature*[3] 16
2.2 Ames room, a famous example of forced perspective, taken at *Cité des Sciences, Paris* [4] 17
2.3 Dakar 2019 photograph by Frank Fife / AFP 17
2.4 Example of consecutive frames with a moving object in KITTI dataset [Gei+13] 19
2.5 Maximum error probability guaranteed for standard and mean error. For $f \in \left[ME, \frac{SE^2}{ME}\right]$, mean error gives more information. Above, standard error is the most informative. 21

3.1 Illustration of a pinhole camera. the focal length is the depth of the box. [5] . 27
3.2 A flow field with constant positive divergence, even if the value at the center is a null vector. Figure taken from [HCC17] 29
3.3 Samples of Flying Chairs for Optical Flow training, with more than 22K frame pairs with perfect ground-truth. Same color code as Middlebury dataset [Bak+11] 31
3.4 FlowNet general architecture for end-to-end optical flow generation, illustrations taken from [Dos+15] 31
3.5 Percentiles of optical flow magnitude wise end point error (EPE) of our implementation of flownet, tested on 100 images of Flying Chairs dataset. Mean EPE on Flying Chairs is below 2 pixels. 33
3.6 Some examples of our renderings with associated depth maps, from 0 to 100m. Visible reticle is not comprised in raw images 36
3.7 Comparison between "1+ELU" and "ReLU(1+x)", functions and derived functions 39

3.8 DepthNet structure parameters 40
3.9 General Structure of convolution and deconvolution blocks of DepthNet. . . 41
3.10 Qualitative results of DepthNet for 64x64 images. FOE is indicated by a red cross on Image1 and Image 2 41
3.11 Median depthwise metric, relative and logarithmic errors of DepthNet on 64x64 images with Identity post processing. Shaded areas are delimited by first and third quartiles. 43
3.12 Qualitative results of finetuned $\text{DepthNet}_{64\to128\to256\to512}$ for 512x512 images. FOE is indicated by a red cross on Image 1 and Image 2 43
3.13 Qualitative comparison between DeMoN [Umm+17], FlowNet [Dos+15] and DepthNet. 46
3.14 Depth errors with respect to distance to FOE on our test set. Contrary to FlowNet, DepthNet and DeMoN do not depend on it 47
3.15 some results on real images input. Top is from a Bebop drone footage, bottom is from a gimbal stabilized smartphone video. 47
3.16 Example of depth flickering for the sky. The two frames are $\approx 83ms$ apart. . 48

4.1 Illustration of bilinear interpolation on the unit square. The four corners have fixed z values as indicated in the figure, the values in between are interpolated, and the interpolated values are represented by the color. [6] . . 53
4.2 General workflow architecture of SFMLearner[Zho+17]. (simplified, without regularization) 55
4.3 In this example called Kanizsa Triangle [Kan55], humans instinctively see a triangle above everything else and thus depth discontinuity in areas with no luminance gradient. 58
4.4 Example of a scene with an horizontal plane. Right is the depth of the vertical line depicted on the rendered image. 59
4.5 comparison of four smooth losses TV, TVV, diffusion, and gradient diffusion. Gaussian noise with $\sigma = 0.2$ has been added to input: a sinusoid and an abs function. SGD is used, with indicated learning rates and momentums in each graph title. Color indicates the number of iterations 66
4.6 Perona-Malik anisotropic diffusion of a picture of a flower, $\kappa = 0.3$ 67
4.7 1D Anisotropic diffusion and gradient diffusion of several functions. Dashed line indicates the image function with null gradient, except on $x = 20$ and $x = 80$. Plain line colors indicate number of iterations 69
4.8 Example of occluded elements in a synthetic scene when trying to estimate depth of $\boldsymbol{I}_t$. 69
4.9 Illustration of the direct projection workflow. The pixel $\boldsymbol{p}$ characterizes an occluded view with multiple possible depth values 73
4.10 Illustration of occlusion detection module. **NaN** values are white colored. Displacement is $\frac{\sqrt{2}}{2}[-1, 1, 1]$: the camera goes down left and forward, field of view is $90°$. First row: presentation of θ_t and theoretical results on this perfectly smooth depth map. Notice the foreground is also occluded because it goes out of field of view. Second row: presentation of on ν, a noisy equivalent of θ_t, with a Gaussian noise with std of 0.1 $\nu_t = \theta_t + \mathcal{N}(0, 0.1)$. Result of direct warping (center) and filtered depth map with index back method. Last row: warp back of warped depth $\widetilde{\nu}_{t\to t+1}$, and filtered depth map using the warp back method, followed by an erosion of 2 pixels. 76

4.11 Our two toy problems "Flower" (top) and "Horizon" (bottom). For each scene, first row are the three frames of the scene, and second row is the ground truth depth of the center frame. Images are 200×200 pixels. Scenes have no rotations. Flower scene has a displacement of $\frac{1}{10}[1, 1, 0]$ (camera goes right and down, so the image goes left and up), Horizon scene has a displacement of $\frac{1}{10}[0, 0, 2]$ (camera goes forward). Camera is a standard centered pinhole with $90°$ of field of view. For optimization depth starting points are equal to background, i.e. 10m 79

4.12 First row: Illustration of optical flow construction, and how a perfect depth produces imperfect warping. A "ghost" of the foreground can be seen on occluded areas, which the reprojection based optimization will wrongly try to solve. Second row, our two imperfectly warped frames from I_{t-1} and I_{t+1} to I_t. Last row, the differences between warped images and I_t are the occlusion areas. We can see that the non-zero areas don't cover the same pixels. 80

4.13 optimization results when only photometric loss is applied for depth optimization. Mean logarithmic error is worse at the end than at the beginning for both cases. For each scene, from left to right and top to bottom: depth result, difference with ground truth, final inverse warp of I_{t-1} and final inverse warp of I_{t+1} 81

4.14 Hyper-parameter search for λ and κ for our two scenes. An ideal (λ, κ) value should maximize the metric $\mathcal{M}$ 82

4.15 Logarithmic error (LE) for our two scenes, using different values of λ and κ. 83

4.16 Result of a photometric optimization with optimal weights λ and κ for our 6 different smooth losses on the Flower scene after 10^4 iterations. Learning rate is 0.15 84

4.17 Result of a photometric optimization with optimal weights λ and κ for our 6 different smooth losses on the Horizon scene after 10^4 iterations. Learning rate is 0.15. 85

4.18 Comparison of a simple optimization (first two rows) and an optimization using occlusion module (last two rows). Optimal parameters for regular diffusion loss are used. Blacked parts of warped images are ignored thanks to our occlusion module. Thanks to this, the usual ghost effect of depth on background is avoided. 87

4.19 Result on Flower scene, with regular diffusion smooth loss. No ghost effect can be seen around the foreground depth, but it performs poorly on foreground depth. 90

4.20 Example of change of luminosity inside a video, when entering the tunnel, the camera rises the exposition, and corresponding point have now different colors. 91

4.21 graph of the function $f : (x, y) \mapsto \frac{2xy+C}{x^2+y^2+C}$ used for l and c in SSIM (here, C is 0.01^2). This function is only convex around its maximum, which make its gradient based optimization difficult 93

5.1 General workflow architecture. *target* and *reference* indices (t and r) are chosen randomly for each sequence (even within the same batch); output transformations of PoseNet are compensated so that $\widehat{\boldsymbol{T}}_{t\to t}$ is identity. d_0 is a fixed nominal displacement. 97

5.2 Comparison between our method and SFMLearner. [Zho+17] on StillBox and KITTI [Gei+13]. 100

5.3 Subjective comparison of disparity maps between SFMLearner [Zho+17] and our method on a small UAV dataset. 101
5.4 Some failure cases of our method on KITTI. First column is a detail of a larger image. The foreground car is moving forward and it's detected as far away, while the background car is moving toward us and is detected as close. Second column is a poorly textured road. 101
5.5 Comparison between our method and SFMLearner [Zho+17] on a normal example from KITTI [Gei+13] and its upside-down variation. 105

6.1 Estimation-wise log error of supervised DepthNet$_{64\rightarrow128\rightarrow256\rightarrow512}$ on StillBox. 108
6.2 Example of a multi-modal depth distribution scene with corresponding depth: foreground is much closer than mean depth and background further. 110
6.3 graph of function f, defined in equation 6.3 110
6.4 Multiple shifts architecture with n parallel depth inferences . Numeric integration, given a desired displacement d_{opt} gives the closest possible displacement between frames, denoted d^* , along with corresponding shift δt. The fusion block computes pixel-wise weights from $\beta_0, \cdots, \beta_{n-1}$ to make a weighted mean of $\beta_0 d_0^*, \cdots, \beta_{n-1} d_{n-1}^*$ 112
6.5 real condition application of the multi-shift algorithm with DepthNet. Top left is input. Last row are raw outputs of the network (from 0 to 100, corresponding to a β from 0 to 1). Top right is fused output, capped up to $100m$. Numbers in the upper left corner of raw inputs indicate respectively temporal frame shift and measured displacement magnitude in meters. The UAV is flying forward at a speed of $1m.s^{-1}$ and an altitude of $\approx 12m$. 114
6.6 results for synthetic 256x256 scenes with noisy orientation. DepthNet has been tested with $n = 1$ and $n = 2$, DepthNet Clamped with $n = 1 \cdots 3$ Tiny DepthNet Clamped with $n = 1 \cdots 4$. Y axis is Mean Relative Error (MRE), X axis is inference speed, in ms on a Quadro M1000M GPU. 114
6.7 Experimental setup for our proof of concept 115
6.8 Experimental Scene where the drone is expected to travel around obstacles when desired trajectory collides with them 115

7.1 Multiple different but correlated semantic levels of a particular scene. From left to right: input, semantic segmentation masks, instance masks, depth. Figure taken from Kendall et al. [KGC18] 120

D.1 Left: diffusion of the noisy function $x \mapsto \frac{1}{2+sin(x)}$ (bottom) and its corresponding inverse (top). Center: diffusion of the inverse of the noisy function $x \mapsto 2 + sin(x)$ (bottom) and the resulting function (top). Right: Evolution obtained by interleaving diffusion (10 iterations, learning rate of 0.4) and minimization of difference with diffused. 131
D.2 Gradient diffusion equivalent of figure D.1. 132

List of Tables

2.1 $Me()$ is the median operator, and δ is a validation measure (e.g. L1 distance) 20
2.2 Comparison of standard and mean errors and corresponding maximum for the tail distribution of f 21
2.3 Considered Losses Summary 24

3.1 Datasets sizes 35
3.2 StillBox dataset parameters 35
3.3 Comparison of three possible depth representations for the network output 38
3.4 Quantitative results for 64x64 images. 42
3.5 quantitative results for depth inference networks. FlowNetS is modified with 1 channel outputs (instead of 2 for flow), trained from scratch for depth with Still Box 42
3.6 Size (millions of parameters) and Inference speeds (fps) on different devices. Batch sizes are 1 and 8 (when applicable). A batch size of 8 means 8 depth maps are computed at the same time. DeMoN was tested only on a 980Ti. On the second table, a quick comparison between tested GPUs . 44
3.7 Quantitative comparison between the three methods. FlowNetS $\rightarrow$ depth row is deducing depth from optical flow and displacement (perfectly known) 45

4.1 Best hyper parameters found and resulting quality indicators, when occlusion detection module (ODM) is active and when it is not. Red measures are to maximize, blue ones are to minimize 88

5.1 Quantitative tests on KITTI[Gei+13] Eigen split [EPF14]. Measures are the same as in Eigen et al. [EPF14]. For blue measures, lower is better, for red measures, higher is better. For training, K is the KITTI dataset [Gei+13], S is the Still Box dataset. For scale factor, GT is ground truth, P is pose. When scale was determined with pose, we discarded frames where GPS uncertainty was greater than $1m$. For supervision, D is depth and O is orientation. $\rightarrow$ denotes fine tuning. 102
5.2 Quantitative tests on StillBox, no pretraining has been done. The supervised DepthNet is here to give an hint on a theoretical limit since it uses the same network, but with depth supervision 103
5.3 Quantitative tests on upside-down KITTI [Gei+13]: sky is down and ground is up. No training has been done. Constant Plane outputs the same depth for every pixel 106

Mathematical conventions

Throughout this thesis, unless specified otherwise, mostly for adequation with litterature notation, we take the following notation convention:

- Scalar will be written as simple lowercase.
- Vectors will be written bold and lowercase. For example $\boldsymbol{p} = (x, y)$. An exception can occur, for ease of notation when considering a 3D point $\boldsymbol{P}$ and its projection in 2D $\boldsymbol{p}$.
- Scalar fields will be written as simple uppercase. Exception for depth and inverse depth map, called θ and ξ to keep consistency with literature. For example depth map:
$$\theta(\boldsymbol{p}) = z, \begin{cases} \boldsymbol{p} \in \mathbb{R}^2 \\ z \in \mathbb{R} \end{cases}$$
- Vector fields will be written bold and upper case. For example optical flow $\boldsymbol{F}$, or multi channel image
$$\boldsymbol{I}(\boldsymbol{p}) = \boldsymbol{c}, \begin{cases} \boldsymbol{p} \in \mathbb{R}^2 \\ \boldsymbol{c} \in \mathbb{R}^3 \end{cases}$$
- Matrices will be written bold, uppercase and with a straight font. For example $\mathbf{M} \in M_{2,2}(\mathbb{R})$
- Normalized values will be added a circumflex. It's only applied on vector or scalar fields for which we can compute spatial mean μ and variance σ. For example $\widehat{\boldsymbol{I}}$ is the normalized image $\frac{\boldsymbol{I}-\mu_{\boldsymbol{I}}}{\sigma_{\boldsymbol{I}}}$.
- Estimations will be added a tilde. Generally, it will be compared against a groundtruth. For example the depth map estimation $\widetilde{\theta}$ compared with ground truth θ, or the warped image $\widetilde{\boldsymbol{I}_1}$ obtained from $\boldsymbol{I}_2$ which aims at being the same as the original image $\boldsymbol{I}_1$.
- Spatial gradients of scalar fields will be noted with ∇. For example the gradient of depth θ is $\nabla\theta(\boldsymbol{p} = (x, y)) = \begin{bmatrix} \frac{\partial\theta}{\partial x}(\boldsymbol{p}) \\ \frac{\partial\theta}{\partial y}(\boldsymbol{p}) \end{bmatrix}$.
- For sake of simplicity, we can mention the spatial gradient of a vector field (while we should talk about a Jacobian matrix). For example when talking about image spatial gradient $\nabla\boldsymbol{I}$, what is actually referred is the spatial gradient of each $\boldsymbol{I}$ channel.
- When the scalar field f is a result of a function taking a vector $\boldsymbol{W}$ for argument (for example a neural network with a set of weights $\boldsymbol{W}$), we note $\nabla_{\boldsymbol{W}} f(\boldsymbol{p})$ the gradients of f output at point $\boldsymbol{p}$ with respect to parameters $\boldsymbol{W}$, with $\boldsymbol{p}$ fixed.

- Similarly, the Jacobian $\mathbf{J}_W$ of a neural network with weights W outputting a scalar field f will be the function

$$\begin{array}{rccc} \mathbf{J}_W : & \mathbb{R}^n & \to & \mathbb{R}^p \\ & \boldsymbol{p} & \mapsto & \nabla_W f(\boldsymbol{p}) \end{array}$$

symbols

Here are presented typical symbols and acronyms for our notations. Even if they are specified when brought, this page can serve as a reminder for clarity.

Acronyms

UAV	**U**nmanned **A**erial **V**ehicle
IMU	**I**nertial **M**easurement **U**nit
HUD	**H**ead **U**p **D**isplay
RTH	**R**eturn **T**o **H**ome
RGB	**R**ed **G**reen **B**lue
SLAM	**S**imultaneous **L**ocalisation **a**nd **M**apping
CNN	**C**onvolutional **N**eural **N**etwork
FOE	**F**ocus **o**f **E**xpansion (see below)

Scalars

$\mathcal{L}_p$	Photometric loss, denotes the difference between two images to be minimized
$\mathcal{L}_s$	Smooth loss, denotes the regularization term that tries to constrain a particular 2D map to be smooth.

Error measures (see 2.4.2)

MAE	Mean Absolute Error $\mathbb{E}\lvert\theta - \widetilde{\theta}\rvert$
MRE	Mean Relative Error $\mathbb{E}\left(\frac{\lvert\theta-\widetilde{\theta}\rvert}{\theta}\right)$
MLE	Mean Logarithmic Error $\mathbb{E}\lvert log(\theta) - log(\widetilde{\theta})\rvert$
SAE	Standard Absolute Error $\sqrt{\mathbb{E}(\lvert\theta - \widetilde{\theta}\rvert^2)}$
SLE	Standard Logarithmic Error $\sqrt{\mathbb{E}(\lvert log(\theta) - log(\widetilde{\theta})\rvert^2)}$
P_δ	Precision $P\left(\widetilde{\theta} \in \left[\frac{1}{\delta}\theta, \delta\theta\right]\right)$

Vectors

$\boldsymbol{p}$	2D point in the image
$\boldsymbol{P}$	3D point in the world
$\boldsymbol{\Phi_t}$	Focus of Expansion (also referred to as **FOE**) with respect to camera translation $\boldsymbol{t}$.

Scalar Fields

θ	depth map of a particular image.
ξ	inverse depth map. $\xi(\boldsymbol{p}) = \frac{1}{\theta(\boldsymbol{p})}$

Vector Fields

$\boldsymbol{I}$	Multi channel Image. Usually with 3 channels.
$\boldsymbol{F}$	Optical Flow map

Chapter 1

Introduction

Contents

1.1 Motivations of this thesis 1
1.1.1 Anatomy of a consumer Drone 2
1.1.2 The problem of obstacles avoidance 2
1.2 Depth perception for navigation in robotics 4
1.2.1 Considered sensors for depth perception 4
1.2.2 Active sensors advantages and limitations 5
1.2.3 Vision based depth perception 6
1.3 Deep Learning Approach 7
1.3.1 A brief historic 7
1.3.2 Deep learning with convolutional neural networks for computer cision 8
1.3.3 Deep learning for decision making 9
1.3.4 Interest of deep learning for consumer UAVs 9
1.4 Approach and goal of the thesis 10
1.4.1 Hardware setup 10
1.4.2 Discarding the naive solution 10
1.4.3 Training with increasing complexity 11
1.4.4 Resulting scope of this thesis 12
1.5 Contributions 12
1.6 Outline of this thesis 13

Although already well developed, the consumer UAV (Unmanned Aerial Vehicle) market is still said to show extremely high potential. Thanks to their simplicity of use, flying cameras democratized aerial footage. Today anyone can make a quality aerial footage without deep knowledge in aeronautics.

However, growing concerns on security of drone flights threatens the future growth of this market. Indeed, a consumer drone crashing can be deadly if falling from 100 meters high or more. This is why consumer drone regulations has strengthened by restricting the usage of flying cameras to certified pilots in isolated areas. Thus the enforced regulations diminish the number of potential customers. If we want the democratization process to be complete, we need to gain popular trust by making flights as safe as possible.

1.1 Motivations of this thesis

This work is motivated by improving security for consumer flying cameras by studying the obstacle avoidance possibilities for a UAV.

1.1.1 Anatomy of a consumer Drone

The application context of consumer drones implies particular binding conditions:

1. Design is cost-oriented. Unlike with professional UAVs, our strategy for an obstacle avoidance system should also be cost-oriented to keep the product attractive for consumers.

2. The aircraft needs to be compact. The consumer UAV is encouraged by the law to stay under a particular weight limit (e.g. 800grams in France [Fra16]) so that the consumer does not have to register it.

3. Battery life should be as long as possible, with a typical limit around ~ 30minutes.

An obstacle avoidance system in this context should not rely on heavy or expensive hardware, and should not require too much energy.

Besides, there are several elements that we can already consider reliable because they are vital for a UAV designed to get quality aerial footage. Improving them is not in the scope of this thesis and will be considered solved, at least for our needs.

1. High level commands. For the sake of accessibility, the commands are made as intuitive as possible. Consumer UAVs can usually be controlled with the 6 degrees of freedom, without the need to explicitly control the rotation speed of the motors. More specifically, a UAV can be assimilated to its reference frame for which we can control its rotation and speed.

2. High quality image stabilization. The need for high quality footage makes the frame stabilization an essential feature for flying cameras. As such, the IMU system embedded guarantees an error of orientation such that the frame stabilization, active or passive (see figure 1.1) typically has at worst ≈ 1 pixel, for sensors up to 4K. If we choose work on downscaled images, we can consider the stabilization to be perfect.

3. Accurately calibrated intrinsic parameters. Most likely, frames are perfectly rectified from geometric distortion, with perfectly known optical center and focal.

However, drone speed estimation is often noisy. Indeed, it mostly relies on inertial sensors that actually measure acceleration and GPS signal, which measures position with a precision of several meters. This can be mitigated by other sensors such as a vertical camera, which will measure apparent displacement using odometry and deduce speed from the altitude. Consequently we will not be able to easily link speed vector with its projection on the frame plane, the so-called *flight path vector* in more advanced aircraft heads up displays (HUD).

1.1.2 The problem of obstacles avoidance

Our strategy to improve safety on consumer UAVs is to focus on obstacle avoidance, which is still a challenging problem in embedded system.

[1] Sources:

- `https://www.dji.com/fr/phantom3-4k` retrieved 07/24/2019
- `https://www.parrot.com/fr/drones/parrot-bebop-2` retrieved 07/24/2019
- `https://www.dji.com/fr/osmo-mobile-2` retrieved 07/24/2019

Figure 1.1: Consumer products featuring camera stabilization. From left to right: DJI Phantom (mechanic gimbal), Parrot Bebop (dynamic cropping from fish-eye), DJI Osmo mobile (mechanic handheld guimbal)[1]

Figure 1.2: Parrot's current Flight Plan interface

Figure 1.3: DJI's ActiveTrack interface

Typical use-cases of consumer flying camera can be divided in several subgroups. In this list, we rank the scenarios based on the importance for an obstacle avoidance to not be invasive, thus defining a more difficult use-case.

1. Return To Home (RTH): this use-case is an automatic flight mode with the simple objective of joining a particular 3D point based on its coordinate, generally the point where the UAV took off initially. This is particularly useful for out-of-range scenarios where the user can no longer pilot the drone. In this scenario, the user does not necessarily want to get good footage quality. The priority is to safely retrieve the aircraft.

2. Automatic flight for footage, like e.g. Parrot's Flight Plan (figure 1.2 and Follow me or DJI's ActiveTrack (figure 1.3): these flight modes require the video to be aesthetically pleasing. Flight Plan is similar to RTH, but the drone has to join several way-points in a particular sequence. Follow Me and ActiveTrack need the UAV to follow the user via GPS or Visual tracking.

3. Manual flight: the user has the entire control of the flight. Obstacle avoidance in this case would be only a flying helper, to avoid too dangerous piloting choices while still following the desired direction as closely as possible. This mode is for example featured with DJI's APAS (Advanced Pilot Assistance Systems) [2].

For Manual flight and Flight Plan or Follow me, the good video quality requirements not only refer to image quality, but also cinematography, with a smooth video. As such,

[2]Introduced with their product Mavic Air `https://www.dji.com/fr/mavic-air`, retrieved 07/24/2019

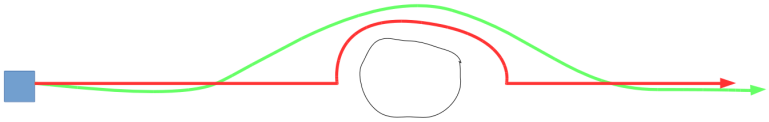


Figure 1.4: A "jerky" trajectory (red) and a smoother one (green)

we want to avoid visible maneuvers as much as possible. This implies both the lowest acceleration and acceleration variation possible (as known as "jerk"). Figure 1.4 shows two trajectories regarding the avoidance of an obstacle when the initial desired trajectory was a straight line. The red one has a very high jerk and acceleration and is not very smooth, which would only be acceptable in the context of RTH, while the green one is much smoother. This is the type of trajectory we want for our obstacle avoidance system.

We can see, however, that the green trajectory is altered well before the red one. The trajectory modification decision thus needs to be planned early, and thus the obstacle to be sensed with a longer range than with the RTH context with the sole purpose of not crashing into the obstacle, regardless of the jerk.

1.2 Depth perception for navigation in robotics

Now that we have established our main navigation requirements for obstacle avoidance for consumer drones, we can study the available techniques and related works to achieve this goal.

The most classical approach for obstacle avoidance relies on two distinct parts, sense and avoid. The first one is focused on perception, and the second one on action. [YZ15]

The perception can rely on several dedicated sensors to get information on the scene structure. The goal is to get the distance of other objects relative to the aircraft. More specifically, given a particular direction, we want to compute a depth map. Just like camera compute a RGB (**R**ed **G**reen **B**lue) pixel map of what can be seen, the depth map is a pixel map with a metric for each pixel, corresponding to the parallel distance relative to the direction where the sensor is pointed at (hence the name depth, and not distance). Figure 1.5 illustrates how we can link image pixels with their depth values. The RGB picture here only serves illustration purpose and for navigation, knowing depth perfectly is sufficient for obstacle avoidance.

It can be noted that the depth map is not the information we want *per se*, but rather the position of possible obstacles with respect to the camera, for which a simple conversion can be done based on the angle of view of a particular point with its depth. However, the depth map format illustrates that for a given orientation, only one depth value is known (the depth of the closest object). We don't want to be able to know the distance of occluded objects, since this would require prior knowledge of the scene.

1.2.1 Considered sensors for depth perception

For a UAV, we can list several solutions for active depth sensing. Contrary to e.g. cameras, these sensors require the emission of a signal (light or sound), whose reflexion will be measured to deduce depth:

- LiDar (**Li**ght **d**etection **a**nd **r**anging) is a rotating light emitter (usually a laser beam) for a receptor that will compute distance from the time for the emitted beam to to be received back. If the direction of the beam is perfectly known, the depth can

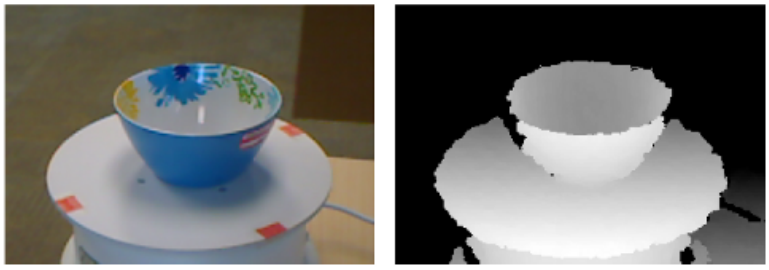


Figure 1.5: Example of RGB-D data, taken from[Lai+11]. For depth (right), bright colors indicate lower depth (and thus closer object to the camera plane)

be deduced. This system has been used for example to get the depth of the KITTI dataset [Gei+13].

- structured light, democratized with the first Kinect, and already equipped in products such as DJI phantom 4. The principle is to project a light pattern from a light source and identify it from a camera with a shifted position. The depth is deduced with triangulation. [Bes88]
- ToF Cameras [Han+12] (**T**ime **o**f **F**light), similar to Lidar but with no moving parts.
- Ultra sound sensors, which uses the same technology as Lidar but with sounds of high frequency. This sensor is very cheap and is already available on almost all consumer drones.

For passive depth sensing, which only requires measuring incoming signal such as light, we can list two camera based solutions:

- Stereo Vision
- Monocular Vision

1.2.2 Active sensors advantages and limitations

Active sensors typically require less complex algorithm to deduce depth from perceived signal: the complexity resides in its hardware conception which is can be done by a specialized provider. Indeed, the emitted signal can be made in a particular pattern so that its detection and identification from receiver is easy.

However, for all active sensors, having a suitable range for smooth obstacle avoidance will require a powerful signal and thus a potentially heavy and power expensive dedicated device, which would greatly lower autonomy, along with increasing weight and manufacturing cost.

Besides, devices based on light emission and reception will suffer greatly for outdoor contexts. Indeed, most of them rely on Infra-Red emission, which is a major light component of sun rays. The receiver will then suffer a lot of noise and will have its typical range greatly reduced. The solution might be to emit more powerful light, but (in addition to power consumption) this is potentially hazardous for the human eye [3]

Finally, once the sensor is integrated with fixed signal emission specifications, range is fixed and thus not flexible to UAV speed. As a consequence, there will be a capped speed above which the drone cannot sense obstacle ahead enough. This is even more problematic to keep smooth trajectories since it needs long-time ahead planning.

[3] As reported e.g. in a 2010 ANSES report `https://www.anses.fr/fr/system/files/AP2008sa0408.pdf`, retrieved 07/24/2019

1.2.3 Vision based depth perception

An interesting alternative is to use a passive sensor, like the camera. The main problem of the image coming from the camera is its extremely low information per pixels, which only gives a RGB color estimate of a particular point. Unlike active sensors, depth deduction from image signal is much more complex because it's less closely linked with image color.

Computer vision is a science field aimed at extracting high level information from these rudimentary "color maps" that are pictures taken from a camera, exploiting not only colors but also spatial and temporal contexts, by working on neighboring pixels in frame sequences or with multiple cameras.

In our context, a depth algorithm from computer vision would be an ideal solution because the camera sensor is already available. Besides, thanks to the UAV's first functionality, the image quality already benefits from heavy engineering to make it visually pleasant, and thus with a high functioning range of luminosity. Calibration, auto-exposition and white balance are thus out-of-scope problems already solved for video quality purpose.

One of the main characteristics of computer vision is the analogy that we can make with human vision, which is known to be one of the most important human sense, taking up to half of our brain's capacity [She+96]. Indeed, one could argue that the camera system is already derived from the eye, by projecting light beams on a plane sensor. The task given for computer vision could then be interpreted as the one given to the human brain to extract critical information from retina receptors intensity of activation, similar to color in image pixels.

Vision based depth sensing algorithms can then rely either on multiple cameras or one camera.

Stereo vision

The first case is a well known setup with two cameras on a stereo rig and already studied [NKG09] and used on several existing consumer products for obstacle avoidance like DJI's APAS already mentionned, or Parrot SlamDunk (see figure 1.6). It is also bio inspired since it reproduces the binocular human view, exploiting parallax to deduce depth. Provided computer vision techniques are able to match corresponding points on both frames (which is already not an easy task, since images only provide color information), a simple triangulation can be applied to deduce the depth. However, just as with structured light, the range of such algorithm heavily depends on image resolution and the distance between the two cameras (called the baseline), parallax being proportional to the ratio baseline/depth. Cameras too close to each other will have trouble sensing long range depth, and stereo camera integration complexity increases with stereo baseline. Typical stereo based obstacle avoidance systems on consumer drones can't guarantee to work at speeds higher than $10m.s^{-1}$, and at this speed they can only perform urgent braking.

Monocular vision

The second case tries to get depth information from a single moving camera, either from appearance or using structure from-motion-algorithms. An in-depth comparison of these techniques is done at chapter 2. This is the use-case we will try to cover, because even if it now requires solving motion in addition to depth, the hardware integration is much easier. It is also flexible to already existing integrated hardware as long as a camera is available. Namely, if a stereo rig is already integrated in a high-end consumer drone, depth from single frame can also be used. This use-case can also be considered bio-inspired, especially in the context of stabilized video because rotation compensation is

Figure 1.6: Example of a stereo camera placed on a UAV in order to compute a depth map (here, a Parrot Slam Dunk on top of a Bebop2)

somewhat related to human vision and vestibulo-ocular reflex (VOR) [DN33]. Our eyes orientation is not induced by head rotation, our inner ear among other biological sensors allows us to compensate parasite rotation when looking toward a particular direction.

Monocular Vision based navigation is actually already a well studied field, especially within the context of SLAM algorithms[Cad+16; MAMT15; KM07; Fu+14]. Prior knowledge w.r.t. scene is used to infer a sparse depth map with its density usually growing over time. These techniques are typically used with unstructured movement, produce very sparse point-cloud based 3D maps and require heavy calculation to keep track of the scene structure and align newly detected 3D points to the existing ones. Although tools like Octomap [Hor+13] can be used, SLAM is not widely used for obstacle avoidance, but rather for 3D scan, where the cinematography and smooth video requirements are clearly ignored. It could also be argued that when trying to avoid obstacle while following a flight plan, mapping will mostly contain information about past elements which are no longer useful.

1.3 Deep Learning Approach

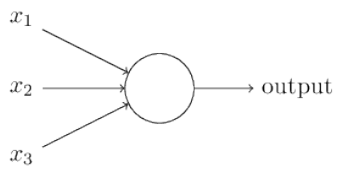
For this work we chose to focus on deep learning methods for vision. The deep learning approach is thus briefly presented here along with motivations to use it.

1.3.1 A brief historic

Just like camera and computer vision, initial motivations for deep learning were bio-inspired. The core idea is to develop a learning algorithm that would capitalize on experience to get better over time.

The first iteration, called the perceptron [Ros58] was just the simulation of a neural networks, with neurons firing an activation or not based on a simple arithmetic operation over its inputs, as presented figure 1.7. We can already see how this structure could be parameterized and perform very different operations based on it.

[4]Figure source: `http://neuralnetworksanddeeplearning.com`, retrieved 07/24/2019



$$output(x_0 \cdots x_n) = \mathbb{1}_{s>\alpha}, s = \sum_i w_i \times x_n$$

Figure 1.7: First simulation of a neuron, $x_0 \cdots x_n$ are the inputs, while $w_0 \cdots w_n$ and α are changeable parameters.[4]

The learning dimension was further improved by making every operation differentiable. That way it was possible to compute the gradient of a network output regarding every one of its parameters thanks to the back-propagation algorithm. The learning strategy was then to define a loss function to minimize and then perform a gradient descent to match the desired output. [RHW+88]

If we define a neural network as a function f taking trainable parameters vector $\boldsymbol{W}$ (called weights), and an input vector $\boldsymbol{I}$ (e.g. a colored image), we can define the result of an inference to be $f(\boldsymbol{W}, \boldsymbol{I})$. The optimization of this neural network then consists in updating the vector $\boldsymbol{W}$ to get a desired output. More specifically, if we define a loss function $\mathcal{L}$ to evaluate the output with respect to a known target, and an evaluation set V of images $\boldsymbol{I}$ and target outputs t, we want to reach the global miminum:

$$\boldsymbol{W}_{final} = \arg\min_{\boldsymbol{W}} \mathbb{E}_{(\boldsymbol{I},t)\sim V} \mathcal{L}(f(\boldsymbol{W}, \boldsymbol{I}), t)$$

V is the set of valid samples on which we want the network to perform, e.g. the set of all possible video frames from a micro UAV. The most standard way of finding this global minimum is to apply a stochastic gradient descent (SGD) [Bot10] on a set of training samples $(\boldsymbol{I}, t)$, separated from the validation set.

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \lambda \nabla_{\boldsymbol{W}} \mathcal{L}(f(\boldsymbol{W}, \boldsymbol{I}), t)$$

Where λ is called the step size and the gradient vector is computed with back-propagation algorithm. Throughout a series of samples $\boldsymbol{I}$ with a way to evaluate the network output, the weights $\boldsymbol{W}$ are updated multiple times to try to achieve minimum loss on the validation set.

It is thus important to note that the learning workflow is essentially based on loss gradient (rather than loss value). Designing a loss is not the same as designing an evaluation function. It should not be forgotten and will be an important topic in chapter 4.

In this very brief presentation, we don't cover all the variation of this very basic gradient descent designed to solve optimization problems such as local minimum or over-fitting [TLL95], and throughout this document, we will mention and use common good practices to make a particular network converge to the desired estimator without extensive justifications.

1.3.2 Deep learning with convolutional neural networks for computer cision

Convolutional Neural Network (or CNN), a particular set of neural network designed by Fukushima et al. [Fuk80] are widely used for computer vision problems with a lot of

success [LeC+98; KSH12]. This architecture tries to extract information from pixels using operations specially designed to link neighboring pixels together instead of having a dense network.

This method has proven extremely useful and scalable for many computer vision benchmarks where deep learning solutions using CNNs are now very competitive. This includes in addition to classification, optical flow [But+12], depth from stereo[SS02], odometry [GLU12], or depth completion [Uhr+17]. [5]

It's also interesting to note that, as shown by Krizhevsky when working on AlexNet [KSH12], the first operations done by the convolutional kernels are very similar to intuitive hand crafted features such as gradient extraction. Semantically, it shows that a CNN can be divided into two parts: the first one being representation learning, where a map of pixels is projected into a map of interesting features for a specified learned tasks, and the second one being the task solving *per se*.

1.3.3 Deep learning for decision making

Recently popularized by DeepMind work on Atari [Mni+15] or Go [Sil+16], deep learning has been used with some success on decision making, showing that highly semantic tasks could be performed as well.

In 2006, LeCun et al. [Mul+06] had already shown that a CNN could be trained to mimic a human behavior on a simple case of off-road driving. This idea was also applied on UAV in order to follow a specified road or pedestrian path [Giu+16; Loq+18]. While obviously not enough for true autonomous driving or flying, these works enlighten the control learning capacity with an easily optimized loss function (here, the difference between human controls and predicted ones).

On the other hand, the field of reinforcement learning [Wat89] showed the potential of a truly Neural network powered autonomous system only supervised by a simple sparse reward function (e.g. positive reward for not crashing, negative reward for crashing). Counter intuitive but efficient strategies can then emerge and beat human level, that would have been impossible with imitation learning.

However, these systems are known to be extremely hard to train, requiring extensive trial and error before being able to shape a meaning reward where no degenerate strategy can be found [Hen+18].

As suggested with DaGGer by Ross et al. [Ros+13; RGB11], a mid point could be human supervision, only when needed, as an emergency intervention. That way more exploration could be done compared to imitation while still having supervision when needed. Of course this needs the constant attention of a human supervisor which is very expensive.

1.3.4 Interest of deep learning for consumer UAVs

One could argue that similarly to dedicated hardware sensors, deep learning vision solutions may not be a good option because most of them are fairly heavy and require a powerful GPU to be run real-time.

It is however reasonable to expect work to be done on very light neural networks, and the development of specialized hardware that can be easily embedded in consumer products like micro UAVs.

[5] A compelling robust challenge was made during CVPR 2018. Unfortunately, no technical report has been made. Leaderboard can be found here: `http://www.robustvision.net/leaderboard.php`, retrieved 07/24/2019

Most importantly, for a product like a flying camera, multiple computer vision tasks are at stake at the same time, e.g. tracking a moving target for a Follow Me/ActiveTrack use-case, odometry or depth sensing. A recent work [Zam+18] showed that several tasks could be combined in the same network in order to perform only one feature extraction that would be useful for both purposes. It then seems reasonable to think about such an architecture as a supplementary motivation to use embedded deep learning.

1.4 Approach and goal of the thesis

1.4.1 Hardware setup

From last sections, we define a minimal hardware context on which any solution proposed by this thesis is supposed to be used.

1. High quality stabilized camera, roughly pointed at the drone displacement direction.
2. No active sensor available, no stereo cameras.
3. Possibility to run embedded CNNs for inference. No fixed computing power is specified, but compact networks will be preferred when possible.

Throughout this thesis, this hardware choice will be taken in count in the different design motivations.

1.4.2 Discarding the naive solution

When considering deep learning for the goal of smooth obstacle avoidance, we could be tempted to test an end-to-end solution that would take monocular images as input, and provide maneuver commands for a safe and smooth flight. This would challenge the traditional workflow of sense-and-avoid by combining the two tasks within the same network, and could potentially result in a very light and responsive solution, the same way an human would avoid obstacle, without the need to explicitly measure objects distance.

Unfortunately, it would be naive to attempt at training a network from scratch for such a task, because of the reward sparsity and the obvious risks for a potential mistake. It goes without saying that the crash and retry strategy which would be inevitable at least at the beginning of the training is not easily scalable, although it has been tested in cluttered spaces at low speed [GPG17].

One could also use a first training pass with imitation learning, as shown in the already mentioned works [Giu+16; Loq+18]. The idea was to construct a dataset for imitation learning in a way that would not involve any UAV flying, but rather humans following normal paths. Constructing an imitation dataset only required humans to walk or bike normally, making the dataset very easy to construct, and the drone learned to imitate the motion of a human head. Unfortunately, these works were only successful in the context of a simple task: following a clear path on the ground, and many use-cases where no ground can be seen will fail, rendering the pretraining useless.

In contrast, comprehensive datasets for imitation learning of obstacle avoidance are very hard and expensive to construct because it involves flying a drone near obstacles numerous times risking human error each time.

In this section we thus discuss a potential path to reach obstacle avoidance by setting intermediate pre-objectives, with easier supervision, in the hope of using reinforcement learning with an already highly functional system.

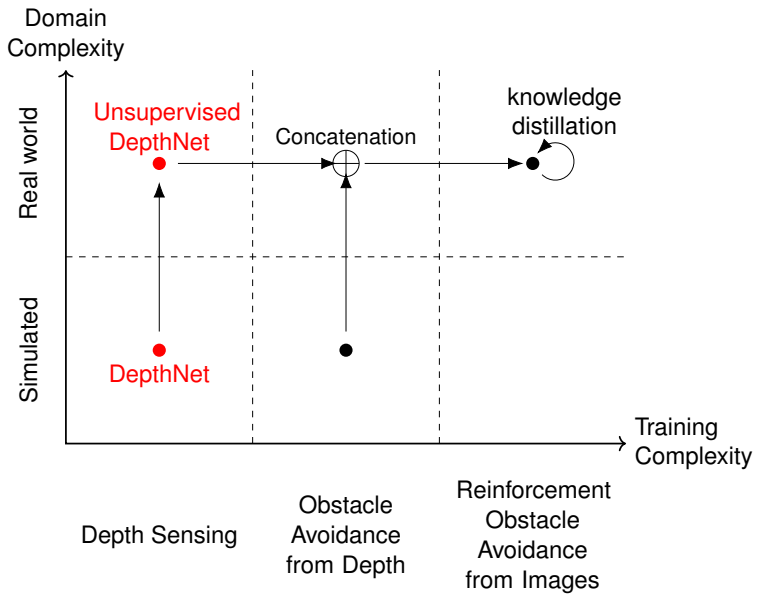


Figure 1.8: Graph of milestones complexity regarding domain and task complexity. Arrows indicate a possible reutilization of networks with a fine-tuning fashion. Red points indicate considered milestones in this thesis.

1.4.3 Training with increasing complexity

This problem can be analyzed along two complexity axes: the domain complexity, and the task complexity. Figure 1.8 presents several tasks.

1. For a given task, the domain can be simplified by using a simulator, where data annotated either with ground-truth or reward is very easy to construct. However, a simulator will necessarily differ from the more complicated reality, and an algorithm with good performances on simulator has chances of performing poorly in reality.
2. Task complexity represents in a sense the sparsity of the useful information for a neural network to learn. Namely, regression problem with ground-truth are deemed much easier to train because each network inference will trigger a loss and thus a weight correction. On the other hand, reinforcement learning systems will only have meaningful rewards once in a while and might take a lot of experience replay to be trained.

From this analysis we decided to consider four different tasks and domains:

- Depth Sensing in simulator. The "easiest" of them, a classic supervised algorithm can be used, this will be the subject of chapter 3.
- Depth Sensing in real scenes. Simple task but with expensive ground truth. A solution to overcome this is to use already known computer vision techniques to get a self-supervised training. This will be the subject of chapter 4.
- Obstacle avoidance in simulator using perfect Depth maps as input. Several avoid algorithms can be used for supervision if the depth is perfect, such as Model Predictive Control [Ric+78; LH17]. Crashing the camera multiple times is not a problem since everything is simulated.
- Obstacle avoidance in real life. This features all levels of complexity, where crashes need to be extremely rare, and constitutes our final goal. A first working solution can

use the concatenation of a depth sensing in real scenes and obstacle avoidance in simulator.

The General strategy can then be summarized by solving the simplest task and adding complexity later on. These tasks still heavily rely on the sense and avoid paradigm since the last network will be the concatenation of a depth sensing and obstacle avoiding from depth. However, it will be possible to train a more compact network in a supervised way to get the same results. This technique is called knowledge distillation. [Rom+15; HVD15]. These milestones are particularly interesting because each of these tasks if solved can be easily used in a classic sense-and-avoid system for a working solution even if the last step is not reached yet. This will be discussed in chapter 6.

1.4.4 Resulting scope of this thesis

The scope of this thesis does not cover all the possible milestones for an end-to-end obstacle avoidance neural network. Instead, we heavily focused on two tasks, by trying to conduct the most exhaustive study possible, while keeping the end goal of obstacle avoidance.

These two tasks are the depth sensing algorithms, inside a simulator and on real video footage, with the possible inclusion in a sense and avoid pipeline.

As such, we won't heavily study control technique for obstacle avoidance. Paradoxically, even if the long term end goal is to replace the sense-and-avoid paradigm to an end-to-end one, the main goal of the thesis will be to develop a reliable depth sensing solution. The knowledge distillation strategy in the end is what makes this strategy still relevant.

1.5 Contributions

This work led to two publications in international conferences and one publication to an international workshop, the two conferences are related to robotic navigation, and the workshop is related to computer vision and geometry.

- [Pin+17a]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "End-to-end depth from motion with stabilized monocular videos". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W3 (2017), pp. 67–74. DOI: 10.5194/isprs-annals-IV-2-W3-67-2017. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W3/67/2017/

- [Pin+17b]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "Multi range Real-time depth inference from a monocular stabilized footage using a Fully Convolutional Neural Network". In: *European Conference on Mobile Robotics*. ENSTA ParisTech. Paris, France, Sept. 2017. URL: https://hal.archives-ouvertes.fr/hal-01587658

- [Pin+18]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "Learning structure-from-motion from motion". In: *ECCV GMDL Workshop*. Munich, Germany, Sept. 2018. URL: https://hal.archives-ouvertes.fr/hal-01995833.

1.6 Outline of this thesis

In addition to this introduction, this thesis is divided in five parts:

- chapter 2 will try to define an evaluation method for depth sensing, with obstacle avoidance in mind. The goal is to discuss current literature on this matter, with its applications and limitations. More specifically, we will see that current benchmarks for depth evaluation might not be suited for the flying camera context, and resulting best strategies for them, using depth from context rather than structure from motion are not ideal for us. In the last section, we will be interested in the amount of information a particular evaluation of an estimator gives on future estimations in the wild.

- chapter 3 will discuss the possibility for a CNN to deduce depth from a pair of stabilized frame. This is the first step in our general long term strategy. After having defined our geometric context, we will further subdivide this problem into an optical flow problem coupled with geometric triangulation, and show that an end-to-end solution (in the context of depth perception) that covers the whole problem at once outperforms the optical flow strategy in the context of a dedicated synthetic dataset created for this thesis. This solution uses a CNN called DepthNet.

- In chapter 4, we aim a finetuning our network trained on synthetic images with real data. After establishing that the training described in previous chapter was only theoretical and cannot be performed to finetune on real world videos where the ground truth is not easily available, we will try to cover the possible strategies for unsupervised learning, using the techniques of photometric reprojection error based optimization. This training technique being relatively new and based on heuristics, we try to describe currently used methods as thoroughly as possible, while linking them with already existing analytical structure from motion methods. More specifically, we discuss the core idea, the possible regularization terms in the optimization, with depth smoothing and occlusion detection, and finally the methods to measure the difference between two frames in a differentiable way.

- chapter 5 will apply the conclusions from previous chapter to perform an unsupervised training of our proof of concept network DepthNet, presented chapter 3. We then show that the resulting network, when used in the evaluation context presented in chapter 2, outperforms other unsupervised solution, mostly thanks to its structure from motion angle of attack in opposition to all other depth appearance related works.

- chapter 6 will present a strategy to use the resulting network with real videos. Namely, we will focus on how to construct an optimal input for DepthNet, and even how to exploit parallel inference to perform a high dynamic range of depth inference. Finally, we will present our proof of concept for obstacle avoidance using DepthNet with an off-the-shelf obstacle-avoidance-from-depth algorithm.

- Finally, chapter 7 will conclude this thesis regarding the long-term end-goal and the strategy that we initially designed, mentioning what issues should be solved. We will try to propose potential future work for this, but also for other perspectives related to this work.

Chapter 2

Depth estimation from a camera

Contents

2.1 Linking depth with reality: the scale factor problem 15
2.2 Depth from motion vs depth from context 16
2.2.1 Depth from context advantages and limitations 16
2.2.2 Depth from motion advantages and limitations 18
2.2.3 Our strategy 18
2.3 Obstacle avoidance oriented depth quality 19
2.3.1 Scale invariant estimation 19
2.3.2 Obstacle avoidance requirements 19
2.4 Anatomy of an error function 20
2.4.1 Standard error vs mean error 21
2.4.2 Usual error functions 22
2.5 Conclusion on error measures 24

This chapter will first focus on the different strategies that can be used to estimate the depth of an image. More specifically, depth from context and perspective will be compared to depth from motion.

On the second part, depth quality evaluation will be discussed in the light of obstacle avoidance context. This particular context will define our strategy on which quality measure is more meaningful, in terms of both safety and path planning.

2.1 Linking depth with reality: the scale factor problem

Measuring depth from the images of a monocular camera is inherently up to a scale factor. For example figure 2.1, when looking at a real castle or a much smaller model replica, the camera frames will look the same, and only other context elements can help estimate depth other than relatively with other pixels. This ambiguity is also appearing with movement estimation, consistently with depth: a video will look exactly the same when filming the real Chambord castle and translating by $10m$, as filming the replica and translating in the same relative direction by $33cm$ (since the replica is at 1/30-scale). It does not mean we can't make estimation, but that all the depth values will only be relative to each other.

As a consequence, in order to get actual depth estimation, an absolute value is needed to serve as an anchor point, a measurement from an other dedicated sensor. The depth can then be deduced from the ratio between the estimated anchor value and the measured one. Possible solutions are:

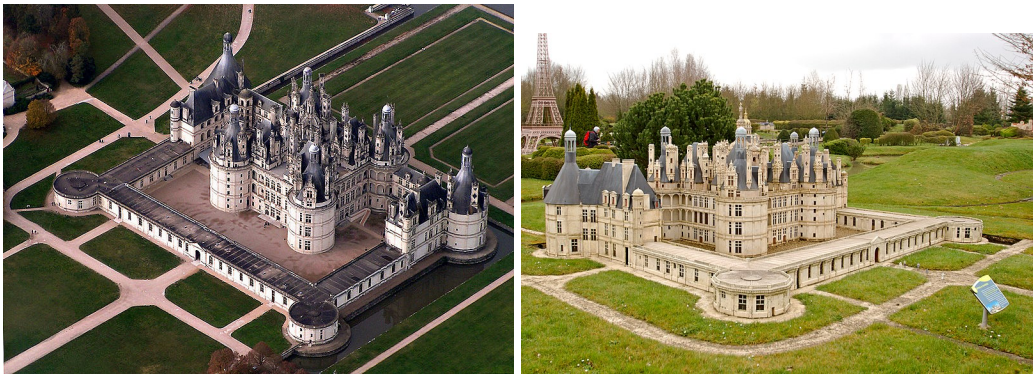


Figure 2.1: Illustration of scale ambiguity. Aerial view of Chambord Castle and 1/30-scale model replica at *France Miniature*[1]

- Measuring depth, at least in one point, with additional sensor such as LiDAR, Time-of-Flight or stereo cameras. This is not a trivial solution and it needs embedded integration, as well as precise calibration to link this measurement with the right pixel in the picture plane.
- Assuming depth consistency across training and testing dataset. This can be particularly useful in datasets like KITTI, [Gei+13], where the camera is always at the same height and looking at the floor from the same angle, but it is irrelevant on a dataset with high pose variability, e.g. UAV videos, and such assumptions will fail.
- Measure movement magnitude, with dedicated sensors, e.g. IMU or GPS in a UAV, or speed from wheels in a car, and compare it to estimated apparent movement, provided both depth and apparent movement are consistently estimated. The resulting ratio will be the scale factor.

The last solution is the preferred one, since movement estimation is a vital measurement in the context of navigation, and thus is almost always available on a UAV.

Besides, as shown on appendix A, a potential error in displacement estimation will also change the depth estimation, partially compensating the speed estimation error. For example if the vehicle underestimates speed magnitude, the depth will also be underestimated and the navigation strategy will try to avoid a potential obstacle more aggressively.

The drawback of this solution is that we need to estimate apparent movement, requiring multiple frames and preferably static scenes, without moving objects.

2.2 Depth from motion vs depth from context

2.2.1 Depth from context advantages and limitations

Depth inference networks have been shown to be able to estimate depth solely from perspective and context. Indeed, with only one image, they were able to get reasonable scale invariant quality measures [Fu+18; SCN08; EPF14; Zho+17]. It has been considered popular and convincing enough to develop its dedicated large scale challenge [Uhr+17]. As said above, in a navigation context, the scale invariant quality is not really interesting without a way to link the estimation to the real world.

[1] Sources: `https://commons.wikimedia.org/wiki/File:ChateauChambordArialView01.jpg` and `https://commons.wikimedia.org/wiki/File:France_Miniature_-_Ch%C3%A2teau_de_Chambord_(045)_(17034171565).jpg`, retrieved 07/24/2019

Figure 2.2: Ames room, a famous example of forced perspective, taken at *Cité des Sciences, Paris* [2]

Figure 2.3: Dakar 2019 photograph by Frank Fife / AFP

As such, because it also estimates movement during training, Zhou et al. [Zho+17] work can be achieved within a consumer drone context. It will be discussed and tested in chapter 5. However, it is already fair to say that these estimations are reliable on a test set similar to what it has been trained with, here KITTI [Gei+13].

Taking human perception as a reference, it has been shown [MBW04] that forcing perspective to the human was well known and studied, especially in art for dramatic effect. One of these techniques is Ames room. Figure 2.2 shows that the apparent perspective is contradicted by the person's size, the depth is thus different on the left and on the right while it looks the same. This illustrates the fact that the projection of a point $\boldsymbol{P}$ is not dependent to its distance to the camera, and highlights the lack of robustness of depth from context that is done by the human eye when looking at a single frame. We can extrapolate that a single frame depth network will suffer the same limitations.

One could argue that these counter examples of forced perspective are not really realistic: if they were to be found unintentionally, the human would have been robust to it. Figure 2.3 features a Dakar 2019 contestant, and an illusory cliff can be seen at first glance before realizing the picture is in fact a high-angle shot (that the human eye is not familiar with). This shows that confusing perspective can sometimes happen even with realistic imagery.

The conclusion of these examples, is that depth from context is not robust to new environments, and especially, depth error (even scale invariant) is not continuous with respect appearance. Because of multi-stable perception [Eag01], large errors can even be found in the very training set. As such, it means that a training set must be extremely thorough, because a slight perturbation of appearance, like a change of lighting or orientation might completely change the outcome. This would make a UAV depth from context dataset very challenging to construct.

Besides, a depth estimator trained on this hypothetic dataset might have to remember among all the possible perspective layouts in a particular image the most probable one with experience, which means heavy memory tasks might be at stake. We can suppose that a neural network dedicated to this task will probably have to be heavy and thus hard to embed on a mobile system.

[2] Source: `https://commons.wikimedia.org/wiki/File:Ames_room_forced_perspective.jpg`, retrieved 3/28/2019

2.2.2 Depth from motion advantages and limitations

Instead of depth from context, the depth from motion benefits from a possible training but mostly from geometrical relations between structure and movement. As it will be discussed on chapter 3, depth from motion only relies on how features are moved in the frame. In the absence of rotation, we know that infinitely distant objects won't move on the image and that features with large displacement will represent close objects.

As such, depth error is more continuous to light and structure conditions, and we predict that depth estimation will be more robust to unseen scenes. This suits both our needs in term of lightweight estimator, not needing to remember every possible realistic depth, and robustness, thus making dataset completeness requirements less vital.

Besides, this technique does not need to explicitly measure apparent displacement magnitude like it was the case for depth from context to solve the scale factor. If every depth from motion estimation is made with the assumption of a constant nominal speed, the apparent speed is fixed and the scale factor will only be a function of actual speed estimation from other sensors.

However, depth from motion is not robust to moving objects. The simplest example to illustrate this is an object that would be moving along with the camera. Object's relative movement to the camera will be null and its distance will appear infinite.

In the context of obstacle avoidance, this could be mitigated by the fact that this is less an obstacle since the relative movement is null. The time to collision, which is the time needed for the camera to reach the obstacle, should both the camera and the obstacle keep their relative motion constant, is infinite. However, knowing the actual depth value instead of the time to collision is important as maneuvers take typically more distance at high speed, making the best strategy dependent of the scene structure. This is a well known effect in road safety where the security distance must be a constant proportion of speed, regardless of actual relative motion [FB01]. Regarding a stopping distance increasing non linearly with speed, like in the example of appendix A, this should even be an increasing proportion of speed.

Considering other movement such as perpendicular motion relative to actual camera path, optical flow anomalies could help detect the object as moving and then discard it from the depth triangulation. However, depth of the moving object will be unknown and the same scale ambiguity problem than with the whole rigid will be encountered: we either need to know our relative motion to it, or have any idea of its typical depth based on its appearance, if e.g. the moving object is an adult pedestrian, which gets back to all the already discussed depth from context problems.

Being robust to moving objects is thus potentially challenging for depth from motion, while it's not the case for depth from context. Although we discuss possible solutions for this particular problem at the end of this document, we chose to only consider rigid scenes throughout the next chapters. The rationale is that solving the rigid scene subproblem would be already a leap forward, and that in the context of UAV navigation where the camera is often at a high altitude when moving fast, this can be enough for most cases.

2.2.3 Our strategy

From this quick analysis of our problem, we chose to focus on depth from motion. The main reasons can be summarized to

- The context of heterogeneous stabilized aerial videos makes a harder use-case for depth from context and appearance, and an easier case for depth from motion.
- Priority is given to robustness, which, as discussed above is problematic for depth from context, given how easily even a system as evolved as the human can be

Figure 2.4: Example of consecutive frames with a moving object in KITTI dataset [Gei+13]

fooled with unusual but realistic images.

- The estimator network needs to be lightweight for a chance to be embedded in the future, which seems more feasible with the lower level problem of feature movement recognition required by depth from motion

2.3 Obstacle avoidance oriented depth quality

2.3.1 Scale invariant estimation

Before trying to estimate the depth of a particular scene, we must be able to measure quality in the context of obstacle avoidance. More specifically, we must be able to take good trajectory correction based on the information given by a particular depth map and reasonable sensors.

For single frame networks, current depth quality measurements, originally introduced by Eigen et al. [EPF14], expect a relative depth map up to a scale factor. This is a scale invariant error measure, designed to evaluate the structure of the estimated depth rather than its actual values, particularly suited for depth completion tasks for example.

This is however not suitable for navigation context, as they completely ignore the scale factor uncertainty, and thus rely on estimating the scale factor as the ratio of the medians of network's output and ground-truth. This is then representative to an ideal use case where the median of an unknown depth map has to be available, which is clearly unrealistic.

Following our decision to estimate both depth and apparent movement from section 2.1, we propose a new quality measurement, which depends not only on depth but also on speed magnitude estimation by slightly modifying the already prevalent ones. This new measurement is not relative anymore, it computes actual depth errors, and is more representative of a real application case where velocity is available.

2.3.2 Obstacle avoidance requirements

The depth $\widetilde{\theta}$ from a particular estimator should give a reliable information in order to make a trajectory decision. As such, there is a particular range of depth values whose estimation needs to be precise.

	prior work [EPF14; Zho+17]	Our proposition
Predictions	Depth $\widetilde{\theta}$	Depth $\widetilde{\theta}$, Velocity $\widetilde{v}$
Ground Truth	Depth θ	Depth θ, Velocity v
Measure	$m = \delta\left(\theta, \widetilde{\theta} \times \frac{Me(\theta)}{Me(\widetilde{\theta})}\right)$	$m = \delta\left(\theta, \widetilde{\theta} \times \frac{\lvert v\rvert}{\lvert\widetilde{v}\rvert}\right)$

Table 2.1: $Me()$ is the median operator, and δ is a validation measure (e.g. L1 distance)

The obvious faulty case, is when an obstacle is thought to be farther than in reality, which makes the UAV chose a trajectory that will lead to a collision. As a result, too optimistic results in the validation set should be penalized in the range of maneuver of the drone. Thus we don't necessarily want depth to be of a constant precision until infinity, the range of maneuver will depend on the obstacle avoidance strategy and specifications. For our case of a relatively slow UAV (compared to e.g. a much faster military UAV) with a typical max speed of $20m.s^{-}1$ it might not be important to get precise depth above a maximum value, set here arbitrarily at $100m$, and the closer depth will be to this maximum value, the less precision will be required.

Also, in our particular use-case we must keep a trajectory close to the one initially targeted, too pessimistic results should also be penalized because it will lead to unnecessary modifications of trajectory, resulting in a safe but inefficient flight.

The relative importance of depth overestimation (too dangerous) compared to underestimation (too conservative) is not clearly defined, but a depth estimator should at least not be encouraged to be biased in one way or another if it was to be optimized for a particular measure.

2.4 Anatomy of an error function

An error function f takes a depth map estimation $\widetilde{\theta}$ and its ground truth θ, and outputs a positive scalar value. $f(\widetilde{\theta}, \theta) > 0$. Obviously, when estimation is perfect, $\widetilde{\theta} = \theta$, the error function is supposed to be 0.

An error measure $\mathcal{E}_f$ of this error function on a particular validation set V is then the mean of all the comparisons between estimation and ground truth.

$$\mathcal{E}_f = \frac{1}{|V|} \sum_{(\widetilde{\theta},\theta)\in V} f(\widetilde{\theta}, \theta)$$

If the validation set is thorough enough, we can write this equation as a mathematical expectation of the random variable $f(\widetilde{\theta}, \theta)$.

$$\mathcal{E}_f = \mathbb{E}(f(\widetilde{\theta}, \theta))$$

$$\mathcal{E}_f = \iint_{(\widetilde{\theta},\theta)\in\mathbb{R}^{+2}} f(\widetilde{\theta}, \theta) \times P(\theta) \times P(\widetilde{\theta}|\theta) d\widetilde{\theta} d\theta \tag{2.1}$$

Since the mean is computed for all $(\widetilde{\theta}, \theta)$ pairs, regardless of θ values, this measure is necessarily biased by the validation set distribution. If in a particular set the depth values in a certain range are under represented, the resulting measure won't reflect the actual depth at this range.

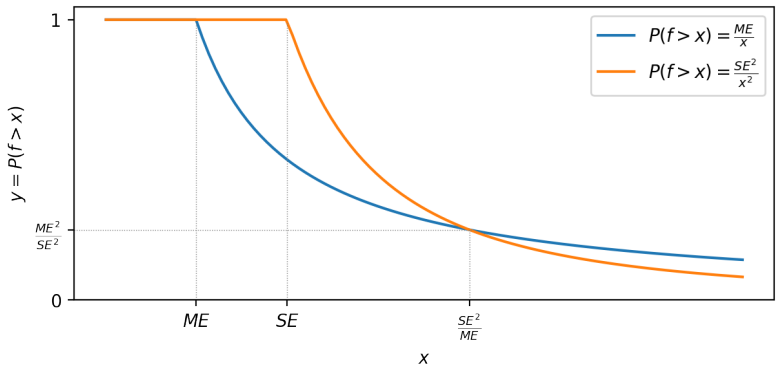


Figure 2.5: Maximum error probability guaranteed for standard and mean error. For $f \in \left[ME, \frac{SE^2}{ME}\right]$, mean error gives more information. Above, standard error is the most informative.

Standard error SE	Mean error ME
$\sqrt{\mathcal{E}_{f^2}} = \sqrt{\mathbb{E}(f(\widetilde{\theta}, \theta)^2)}$	$\mathcal{E}_f = \mathbb{E}\lvert f(\widetilde{\theta}, \theta\rvert$
$P(f(\widetilde{\theta}, \theta) \geq \alpha) \leq \frac{SE^2}{\alpha^2}$	$P(f(\widetilde{\theta}, \theta) \geq \alpha) \leq \frac{ME}{\alpha}$

Table 2.2: Comparison of standard and mean errors and corresponding maximum for the tail distribution of f

It is thus very important to design a validation set V that fits our needs and the typical depth values that we need to be precise on. For example, when considering a high speed obstacle avoidance strategy, very low depth values might not be very important, since no good strategy can be used in that context, even if the measurement is accurate.

2.4.1 Standard error vs mean error

For a particular error function f, there are generally two ways of computing statistics on it. We can compute its mean, which we will call the mean error (ME) but we can also compute the mean of its square value (MSE). Both of these techniques measure how far from 0 the error f usually is, but the mean square error gives more priority to outliers. However, the Bayesian error distribution is more restricted: with Markov's inequality, we can see we have more concentrated repartition of possible error values than with mean error. (see table 2.2). The root of mean square error (sometimes called $RMSE$), which we will call here standard error (SE) because of its proximity with standard deviation will always be higher than mean error and thus give less information on lower possible error values, but gives more guarantee on the possible outliers.

It can be seen figure 2.5 how these two ways of measuring an error are complementary, but when it comes to comparison, the preferred measure will depend on the obstacle avoidance strategy. A more conservative system will prefer to minimize SE (standard error) over ME (mean error). In most cases, the mean error is given, because the distribution tail problems that are solved by standard error are often negligible in finite validation sets, without infinite depth values. Unless specified otherwise, mostly to compare the proposed algorithms with literature, we will use the mean error ME when measuring an error function.

2.4.2 Usual error functions

This section will try to cover usual error functions that are typically measured when evaluating a depth map estimation, not necessarily in the context of obstacle avoidance. Our goal is to determine how informative an estimator is with a particular error evaluation function. More specifically, we want to get an idea of θ Bayesian distribution given an estimation $\widetilde{\theta}$ and an evaluation on a validation set V.

Absolute difference

The absolute difference is the simplest error function.

$$f(\widetilde{\theta}, \theta) = |\widetilde{\theta} - \theta|$$

$$\forall \widetilde{\theta} \in \mathbb{R}^+, f(\widetilde{\theta}, \theta) = \alpha \Rightarrow \theta = \widetilde{\theta} \pm \alpha$$

It can already be noted that these error functions don't rely on the scale when computing its mean, contradicting our desire of being more tolerant on high depth values.

Another simple problem raised by this is the dependence to the validation set's actual distribution. This is not necessarily problematic if the set's distribution is well known, but it should not be forgotten.

Relative difference

With the goal of penalizing errors in accordance to depth scale, the relative errors are normalized by dividing by the ground truth values.

$$f(\widetilde{\theta}, \theta) = \frac{|\widetilde{\theta} - \theta|}{\theta} = \left| 1 - \frac{\widetilde{\theta}}{\theta} \right|$$

$$\forall \widetilde{\theta} \in \mathbb{R}^+, f(\widetilde{\theta}, \theta) = \alpha \Rightarrow \theta = \frac{\widetilde{\theta}}{1 \pm \alpha}$$

The major issue is how penalizing this error function is for over-estimations. In fact, an error of 1 means that either the ground-truth is half the estimation or infinite. As such, an estimator chosen on this measure will be encouraged to under-estimate depth values.

Proposition 1. *Given an estimator e responsible for estimations $\{\widetilde{\theta}\}$, in a corresponding validation dataset $V = \{(\widetilde{\theta}, \theta)\}$, if the estimator is optimized for that validation set for a mean relative difference error, it will have more under-estimations than over-estimations.*

$$P(\widetilde{\theta} < \theta | (\widetilde{\theta}, \theta) \in V) > 0.5$$

Corollarily, if we have an evenly balanced estimator $\{\widetilde{\theta}\}$ such that $P(\widetilde{\theta} < \theta | (\widetilde{\theta}, \theta) \in V) = 0.5$, then we can find $\alpha < 1$ such that $\{\alpha\widetilde{\theta}\}$ is a better estimator according to mean relative difference MRE.

See appendix B for a proof.

Log difference

The two aforementioned error functions have an additional drawback because the difference (normalized or not) implies that compared values are spread on the whole real spectrum, as if the distribution $P(\theta|\widetilde{\theta})$ was a particular Gaussian with its mean on $\widetilde{\theta}$. The log error on the other hand makes this assumption on the log of depth values.

$$f(\widetilde{\theta},\theta) = \left|log(\widetilde{\theta}) - log(\theta)\right| = \left|log\left(\frac{\widetilde{\theta}}{\theta}\right)\right|$$

$$\forall\widetilde{\theta} \in \mathbb{R}^+, f(\widetilde{\theta},\theta) = \alpha \Rightarrow \theta = \widetilde{\theta} \times e^{\pm\alpha}$$

This error has the advantage of being symmetrical, and penalizes 0 estimations as heavily as $+\infty$, which better reflects our ambition to avoid under-estimation and unnecessary security maneuvers. Besides, since it is only a function of the ratio between estimation and ground truth, this measure is not tied to the validation set distribution the way absolute difference was.

Precision error

The precision error, contrary to other loss we discussed is a statistic loss that gives a proportion of outliers. Specifically, it cannot go outside of $[0,1]$. Besides, it only tells statistic for a specific error, here noted δ, and does not tell any information regarding the distribution of the outliers, which could be all very close to the threshold δ or very far without influencing this evaluation.

$$f(\widetilde{\theta},\theta,\delta) = 1 - \mathbb{1}_{\widetilde{\theta}\in[\frac{1}{\delta}\theta,\delta\theta]} = \begin{cases} 0 & \text{if } \widetilde{\theta} \in \left[\frac{1}{\delta}\theta,\delta\theta\right] \\ 1 & \text{otherwise} \end{cases}$$

It's easy to see that its expectation is in fact the proportion of outliers regarding the threshold logarithmic error $-log(\delta)$.

$$f(\widetilde{\theta},\theta,\delta) = \mathbb{1}_{\left|log\left(\frac{\widetilde{\theta}}{\theta}\right)\right| \geq -log(\delta)}$$

$$PE_\delta = P\left(max\left(\frac{\widetilde{\theta}}{\theta}, \frac{\theta}{\widetilde{\theta}}\right) \geq \delta\right)$$

As such, these often used measures are a good way to characterize the logarithmic error behavior, especially in the context of Bayesian inference. The delta values measured are typically 1.25, $1.25^2 = 1.56$ and $1.25^3 = 1.95$ [EPF14]. So their measure is a proportion of estimation with more than 25%, 56% and 95% of error.

It is worth noting that the opposite of precision error is often preferred, simply precision. Optimizing for this particular measure is then maximizing the precision

$$P_\delta = P\left(max\left(\frac{\widetilde{\theta}}{\theta}, \frac{\theta}{\widetilde{\theta}}\right) \leq \delta\right)$$

the optimal score being 1.

Error Name	Acronym	Equation $f(\widetilde{\theta}, \theta)$
Mean Absolute Error	MAE	$\mathbb{E}\lvert\widetilde{\theta} - \theta\rvert$
Mean Relative Error	MRE	$\mathbb{E}\frac{\lvert\widetilde{\theta}-\theta\rvert}{\theta}$
Mean Log Error	MLE	$\mathbb{E}\lvert\log(\widetilde{\theta}) - \log(\theta)\rvert$
Standard Absolute Error	SAE	$\sqrt{\mathbb{E}(\widetilde{\theta} - \theta)^2}$
Standard Log Error	SLE	$\sqrt{\mathbb{E}(\log(\widetilde{\theta}) - \log(\theta))^2}$
Precisions δ	P_δ	$P\left(\left\lvert log\left(\frac{\widetilde{\theta}}{\theta}\right)\right\rvert \leq log(\delta)\right)$

Table 2.3: Considered Losses Summary

2.5 Conclusion on error measures

These different error functions and their respective means on a particular validation set give a complementary view of different strengths and weaknesses. More specifically, the absolute difference is a good performance index on the most represented depth values of a validation set, relative difference (when performance is below 1) is a good indication on depth over-estimation.

Throughout this thesis, we will use the different error measures summarized table 2.3 to characterize a depth estimator, but the logarithm error will be used when tuning our algorithms, and especially the hyper-parameter search presented in chapter 4.

Chapter 3

Creating a depth map with a convolutional neural network

Contents

3.1 Geometric definitions . . . 26
3.1.1 Camera plane . . . 26
3.1.2 Pinhole camera model . . . 26
3.1.3 Moving camera . . . 27
3.1.4 Optical flow and disparity . . . 28
3.1.5 Focus of expansion . . . 29
3.1.6 Finding depth from optical flow . . . 30
3.2 Deep learning for optical flow . . . 30
3.2.1 FlowNet . . . 31
3.3 Optical flow for depth generation . . . 32
3.3.1 From disparity to depth . . . 32
3.3.2 From flow divergence to depth . . . 33
3.3.3 Conclusions on limitations . . . 34
3.4 End-to-end learning of depth generation . . . 34
3.4.1 Introducing StillBox, a depth synthetic dataset . . . 35
3.4.2 Dataset set augmentation . . . 37
3.5 DepthNet, a depth fully convolutional neural network . . . 37
3.5.1 The constant speed assumption . . . 37
3.5.2 On the optimal network output . . . 37
3.5.3 General structure . . . 38
3.6 Training details . . . 39
3.6.1 Influence of resolution on training . . . 39
3.6.2 Comparison with DeMoN and depth from optical flow . . . 44
3.7 UAV navigation use-case . . . 45
3.8 Conclusion of this chapter . . . 48

This chapter is dedicated to investigate the capacity of Fully Convolutional Neural Networks to perform regression tasks, or more specifically to compute the depth of a stabilized image. As discussed before, depth for stabilized image is a too specific problem to have a good dedicated literature. Our strategy is to put this problem in relation with more general ones, and try to adapt their solutions to fit this problem's specificity.

In a first part, we will define the framework of this task, with geometric definitions, and especially the link between optical flow and depth map.

Then, we will look at some Deep Learning solutions for a widely studied problem, the optical flow problem. This would lead us to a first working solution, but with sensible limitations, especially when considering navigation and obstacle avoidance.

We proposed a solution to overcome these, by asking the Network to infer depth directly instead of deducing it from optical flow. This solution implies the creation of a synthetic dedicated dataset.

Finally, we propose a more in depth review of a proposed network and its training process, along with results both in synthetic and real images.

This chapter has been the subject to the following publication

[Pin+17a]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "End-to-end depth from motion with stabilized monocular videos". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W3 (2017), pp. 67–74. DOI: 10.5194/isprs-annals-IV-2-W3-67-2017. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W3/67/2017/.

3.1 Geometric definitions

3.1.1 Camera plane

The camera plane describes the image pixels as a grid of 3D colors (Red, Green and Blue). When considering our image to be of dimensions $h \times w$, we can model the image $\boldsymbol{I}$ as a vector field defined on a discrete plane Ω

$$\begin{aligned} &\Omega = ([0, w-1] \times [0, h-1]) \cap \mathbb{Z}^2 \\ &card(\Omega) = |\Omega| = h \times w \end{aligned} \tag{3.1}$$

$$\forall \boldsymbol{p} = (u, v) \in \Omega, \boldsymbol{I}(\boldsymbol{p}) = \boldsymbol{c} = (c_1, c_2, c_3)$$

The problem of depth generation is then to assign for each point of Ω a particular depth value.

3.1.2 Pinhole camera model

The Pinhole camera model is a way of describing the mathematical relationship between 3D objects and their projection in the camera plane. This model is very appreciated for its absence of distortion and in fact is readily usable on a wide number of cameras, be it from a dedicated lens [Kin89] of from a software correction [Con19]. Thanks to this, systems with a good image quality can be assumed to either provide directly a rectilinear image as if it came from a pinhole, or at least a good calibration to make the virtual equivalent image available at the cost of a rectification algorithm.

If we consider a camera with its coordinate system centered at the optical center of its lens, a 3D point $\boldsymbol{P} = (x, y, z)$ will have the projected point $\boldsymbol{p} = (u, v)$ with the following relation:

$$\begin{cases} u &= u_0 + f_u \frac{x}{z} \\ v &= v_0 + f_v \frac{y}{z} \end{cases} \tag{3.2}$$

With u_0 and v_0 the coordinates of the optical center in the projection plane, and f_u and f_v the focal lengths. Note that with this thesis' convention, we then get $\theta(\boldsymbol{p}) = z$. Besides, $\boldsymbol{p}$ here is not necessarily on Ω: it can be out of bounds or not of integer coordinates.

Most of the time, f_u and f_v are very close and thus this model can be physically approximated to a pinhole camera where every light ray only pass through one point

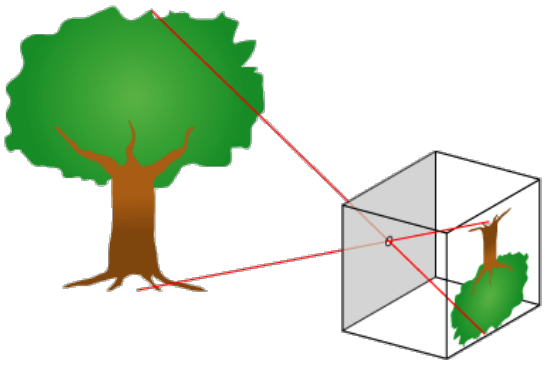


Figure 3.1: Illustration of a pinhole camera. the focal length is the depth of the box. [1]

(discarding any diffraction phenomenon), illustrated figure 3.1. To get a more concise notation, we can introduce the function Π, its pseudo-inverse Π_{-1}, and the matrix **K**

$$\begin{aligned} \Pi: \quad & \mathbb{R}^3 & \to \quad & \mathbb{R}^2 \\ & \boldsymbol{P} = (x, y, z) & \mapsto \quad & (\tfrac{x}{z}, \tfrac{y}{z}) \end{aligned} \tag{3.3}$$

$$\begin{aligned} \Pi_{-1}: \quad & \mathbb{R}^2 & \to \quad & \mathbb{R}^3 \\ & \boldsymbol{p} = (u, v) & \mapsto \quad & (u, v, 1) \end{aligned}$$

$$\mathbf{K} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that for an unoccluded 3D point $\boldsymbol{P} = (x, y, z)$, We have $\theta(\Pi(\mathbf{K}\boldsymbol{P})) = z$. We can then rewrite equation 3.2 as

$$\boldsymbol{p} = \Pi(\mathbf{K}\boldsymbol{P})$$

If we want to get the 3D point from its image on the projection plane, we get:

$$\boldsymbol{P} = \theta(\boldsymbol{p}) \times \mathbf{K}^{-1}\Pi_{-1}(\boldsymbol{p})$$

We can see how the depth is needed in order to figure the position of the 3D point. Besides, the Π operation is scale-invariant.

$$\forall \alpha \in \mathbb{R}^+, \Pi(\alpha\boldsymbol{P}) = \Pi(\boldsymbol{P})$$

This illustrates the fact that the projection of a point $\boldsymbol{P}$ does not depend on its distance (not to confuse with depth) to the camera, as said section 2.1.

3.1.3 Moving camera

If we suppose a moving camera, while our 3D point $\boldsymbol{P}$ stays motionless (the scene is supposed rigid), we can now compute the new projection $\boldsymbol{p}_2$ of $\boldsymbol{P}$.

The motion of our camera can be characterized by a rotation matrix **R** and a translation $\boldsymbol{t}$ from the second coordinate system to the first. It means that we consider the inverse of the actual displacement of the camera. That way, we can express a direct

[1] source `https://commons.wikimedia.org/wiki/File:Pinhole-camera.svg`, retrieved 07/24/2019

transformation for 3D points in the first frame coordinate. For example if the camera is going forward, the points will be closer and thus go backward, relatively to the frame coordinate. The new relative position of $\boldsymbol{P}$ is then

$$\boldsymbol{P}_2 = \boldsymbol{t} + \mathbf{R}\boldsymbol{P} = (x_2, y_2, z_2)$$

We can then deduce the position of $\boldsymbol{p}_2$ in our projection frame

$$\boldsymbol{p}_2 = \Pi(\mathbf{K}\boldsymbol{P}_2) \tag{3.4}$$
$$\boldsymbol{p}_2 = \Pi\left(\mathbf{K}(\boldsymbol{t} + \mathbf{R}\boldsymbol{P})\right) \tag{3.5}$$
$$\boldsymbol{p}_2 = \Pi\left(\mathbf{K}\mathbf{R}\mathbf{K}^{-1}\Pi_{-1}(\boldsymbol{p}) + \frac{\mathbf{K}\boldsymbol{t}}{\theta(\boldsymbol{p})}\right) \tag{3.6}$$

Again, we can see how the position $z = \theta(\boldsymbol{p})$ is needed in order to know the position of $\boldsymbol{p}_2$ from $\boldsymbol{p}$. However, when both projected points $\boldsymbol{p}$ and $\boldsymbol{p}_2$ are known, we can try to to figure out the depth value $\theta(\boldsymbol{p})$ thanks to this equation. Note that if the 3D point $\boldsymbol{P}$ is chosen so that its projection $\boldsymbol{p}$ has integer coordinates (and more specifically $\boldsymbol{p} \in \Omega$), this is not necessarily true for $\boldsymbol{p}_2$.

It is interesting to note how translation, given by $\boldsymbol{t}$, and rotation, given by $\mathbf{R}$ are separated in this equation. Besides, when considering a stabilized motion, i.e. with $\mathbf{R}$ equal to the identity matrix.

$$\boldsymbol{p}_2 = \Pi\left(\Pi_{-1}(\boldsymbol{p}) + \frac{\mathbf{K}\boldsymbol{t}}{\theta(\boldsymbol{t})}\right) \tag{3.7}$$

3.1.4 Optical flow and disparity

Definition 1. *The optical flow is the displacement of a point in the image plane between two frames. It's a vector field defined on Ω, associating a displacement vector for each point in the image. For a particular set of point correspondences $C = \{(\boldsymbol{p}, \boldsymbol{p}_2)\} \subset \Omega \times \mathbb{R}^2$ where each point $\boldsymbol{p}$ of Ω may only appear once in C, we have:*

$$\begin{array}{rccl} \boldsymbol{F}: & \Omega & \to & \mathbb{R}^2 \\ & \boldsymbol{p} & \mapsto & (F_u, F_v) = \boldsymbol{p}_2 - \boldsymbol{p} \end{array}$$

Definition 2. *Disparity is the norm of optical flow.*

$$\begin{array}{rccl} D: & \Omega & \to & \mathbb{R}^+ \\ & \boldsymbol{p} & \mapsto & \|\boldsymbol{F}(\boldsymbol{p})\| \end{array}$$

Disparity is a term mostly coined for stereo vision. It is a special case of stabilized pair where displacement is only lateral, with $\boldsymbol{t} = (t_x, t_y, 0)$. In that case, the 3D point $\theta(\boldsymbol{p})\Pi_{-1}(\boldsymbol{p}) + \mathbf{K}\boldsymbol{t}$ has the same depth value as $\boldsymbol{p}$ and the equation 3.7 becomes

$$\boldsymbol{p}_2 = \boldsymbol{p} + \frac{1}{z}(f_x t_x, f_y t_y)$$

As such, assuming $f_x = f_y = f$, optical flow is then $\boldsymbol{F}(\boldsymbol{p}) = \frac{f}{z}(t_x, t_y)$, and disparity is

$$D(\boldsymbol{p}) = \frac{f}{\theta(\boldsymbol{p})}\|\boldsymbol{t}\|$$

The value $\|\boldsymbol{t}\|$ is called the baseline in this case. When it is known, for example with two cameras on a fixed rig, it is easy to deduce the depth of a particular point. Because

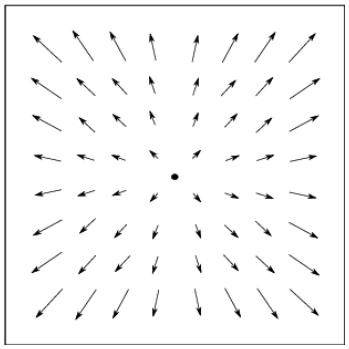


Figure 3.2: A flow field with constant positive divergence, even if the value at the center is a null vector. Figure taken from [HCC17]

of this simple relation for this particular case, the term disparity is abusively used to refer to the inverse depth.

As such, it might be useful to remind that in other circumstances, disparity is not the inverse depth up to a scale factor. This is important as both terms will be used in this work.

Optical flow divergence

If we suppose optical flow $\boldsymbol{F} = (F_u, F_v)$ to be spatially differentiable, we can compute the value

$$div(\boldsymbol{F}) = \nabla.\boldsymbol{F} = \frac{\partial F_u}{\partial u} + \frac{\partial F_v}{\partial v}$$

since $\boldsymbol{F}$ is only defined on Ω which is not continuous, we can approximate its divergence with finite difference convolution for example.

$$div(\boldsymbol{F}) = \mathbf{S}^T * F_u + \mathbf{S} * F_v$$

where $\mathbf{S} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$ and $*$ is the convolution operation.

It can be geometrically interpreted as the magnification of the image objects. Positive optical flow divergence stands for object appearing bigger in the second frame than in the first, even if they did not move, see figure 3.2. As such, measuring the divergence of optical flow is the same as measuring the zooming factor.

3.1.5 Focus of expansion

When considering a stabilized camera displacement $\boldsymbol{t}$ with $t_z \neq 0$, we introduce the Focus of Expansion (FOE) point, noted $\boldsymbol{\Phi_t}$, which is the projection of a virtual point with $\boldsymbol{t}$ as coordinates.

Definition 3. *Given a stabilized motion with translation $\boldsymbol{t} = (t_x, t_y, t_z)$ a pinhole camera with calibration matrix $\boldsymbol{K}$, the focus of expansion is defined by $\boldsymbol{\Phi_t} = \Pi(\boldsymbol{Kt})$*

This point is particularly interesting for stabilized motion, the resulting optical flow can be expressed as simply a function of $\boldsymbol{p}$, $\boldsymbol{\Phi_t}$ and $\theta(\boldsymbol{p})$ and t_z.

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{p}) &= \boldsymbol{p}_2 - \boldsymbol{p} && (3.8)\\ &= \Pi\left(\mathbf{K}\boldsymbol{t} + \theta(\boldsymbol{p})\Pi_{-1}(\boldsymbol{p})\right) - \boldsymbol{p} && (3.9)\\ &= \frac{1}{\theta(\boldsymbol{p}) + t_z}\left(t_z\boldsymbol{\Phi_t} + \theta(\boldsymbol{p})\boldsymbol{p}\right) - \boldsymbol{p} && (3.10)\\ &= \frac{t_z}{\theta(\boldsymbol{p}) + t_z}\left(\boldsymbol{\Phi_t} - \boldsymbol{p}\right) && (3.11) \end{aligned}$$

This equation proves how closely optical flow is linked to FOE, and why its named like that. We can see that the vector is strictly proportional to the vector that goes from $\boldsymbol{p}$ to $\boldsymbol{\Phi_t}$. When going forward (which means $t_z < 0$), every optical flow vector will be headed away from FOE.

3.1.6 Finding depth from optical flow

In light of these different links between depth, we now have a first angle of attack to solve our depth problem. By trying to computing optical flow, and assuming we know $\boldsymbol{t}$ with external sensors, e.g. accelerometers (IMU) and GPS, we can deduce the depth map of our first frame.

3.2 Deep learning for optical flow

Although very popular, optical flow is a very difficult computer vision problem and many issues still remain unsolved, especially for some artifacts like motion blur or change of exposition. Considering a deep learning solution, if older datasets can only serve for evaluation purpose due to their size like the Middlebury dataset [Bak+11], some bigger datasets can be used for training, be it from synthetic images with MPI Sintel [But+12] (with complete ground truth) or from real images with KITTI [Gei+13] with sparse ground-truth.

These datasets, although offering a training framework, were not sufficient to train a network to fully encompass the optical flow operation, but rather the projection function that could lead to feature maps to know if two pixels are referring to the same part in the 3D scene or not [ZK15; ZL16; Han+15]. That way, the optical flow is then found matching the most similar pixels.

$$\forall \boldsymbol{p} \in \Omega, \boldsymbol{F}(\boldsymbol{p}) = \underset{\boldsymbol{p'}\in\Omega}{argmin}(d(f(\boldsymbol{I}_1, \boldsymbol{W})(\boldsymbol{p}), f(\boldsymbol{I}_2, \boldsymbol{W})(\boldsymbol{p'})))$$

where f is the neural network outputting a feature map $f(\boldsymbol{I}, \boldsymbol{W})$ from an image $\boldsymbol{I}$ and a set of trained weights $\boldsymbol{W}$, and d is a distance function, possibly computed by another neural network, performing a classification task to infer how similar are two feature map pixels.

This sub-task might not need too much data to make a network converge, but lacks flexibility compared to end-to-end training algorithms powered by even bigger dataset, with synthetic purposely unrealistic images.

These datasets were proposed in order to get pretraining with a massive amount of data like Flying Chairs [Dos+15], from which Figure 3.3 presents two examples, or FlyingThings3D[N.M+16]. The main purpose is to propose a "pretraining" framework, and then perform a fine-tuning on a smaller realistic dataset. That way, the network can understand the global notion of optical flow with a pretraining and not only the feature

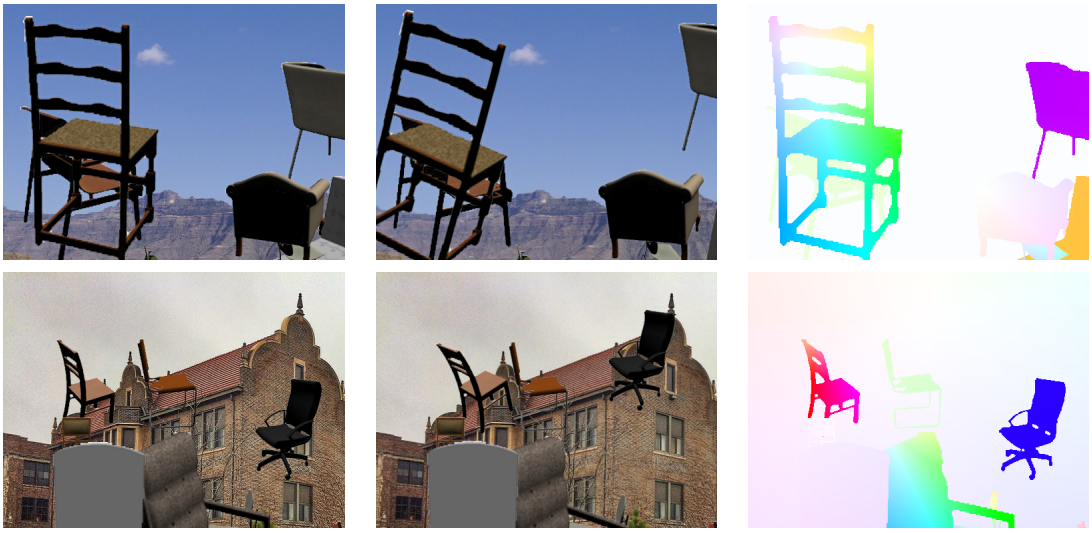


Figure 3.3: Samples of Flying Chairs for Optical Flow training, with more than 22K frame pairs with perfect ground-truth. Same color code as Middlebury dataset [Bak+11]

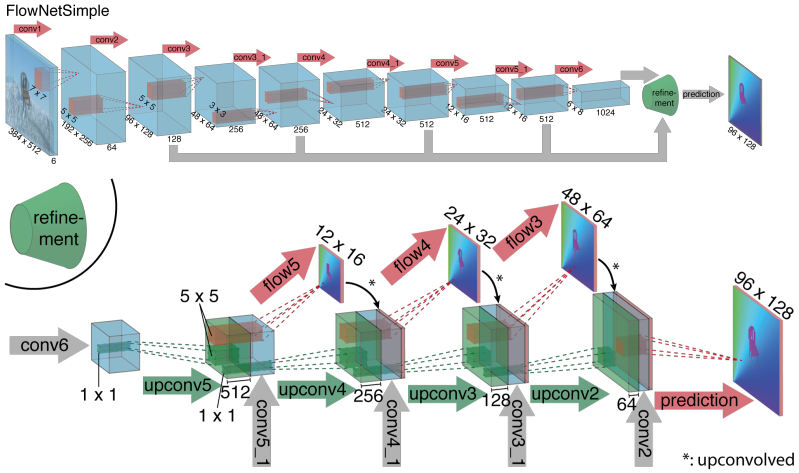


Figure 3.4: FlowNet general architecture for end-to-end optical flow generation, illustrations taken from [Dos+15]

matching, and then specialize to a certain type of images as a fine-tuning. This is somewhat related to Curriculum Learning [Ben+09] and gets the same conclusion about the network generalization.

3.2.1 FlowNet

One of the main network that particularly benefited from these very large datasets is FlowNet [Dos+15], whose architecture is inspired from semantic segmentation networks [LSD15] and presented figure 3.4. This architecture has been named with very different names such as tiramisu [Jég+17], feature pyramid [Lin+17], U-Net [RFB15], and it consists in a hourglass architecture, with skip connections with feature maps of the same size. As such, we get a "projection" part, that computes feature maps with high semantic level, and "refinement" part which concatenates up-sampled feature maps with corresponding earlier convolution outputs, with information more closely linked to pixels because it went through less strided convolutions.

This network is one of the first work that made possible a completely end-to-end train-

ing of optical flow, and only simple convolution and transposed convolution operations are done, with no geometry, smoothing or matching function involved anywhere. For its simplicity despite its good results, we chose to study closely its usage and architecture for our problem. One should note that the following study does not necessarily depend on the architecture, and could be used with more sophisticated networks [Sun+18; Ilg+18], although the simplicity of this architecture also guarantees a good flexibility when contemplating the possibility of modifying its initial purpose or architecture.

3.3 Optical flow for depth generation

In this section we discuss the relation between depth and optical flow $\boldsymbol{F}$ coupled with translation estimation $\boldsymbol{t}$. Most importantly, we discussed how uncertainties for optical flow and translation can lead to big errors in depth, even in the context of a stabilized camera with a perfectly known displacement.

This navigation context can help us prioritize errors. To get the depth values of the last image $\boldsymbol{I}_t$ we received from the stabilized camera, we need to get the optical flow of image $\boldsymbol{I}_t$ compared to anterior frames $\boldsymbol{I}_{t-\Delta t}$. As such, our convention of taking displacement $\boldsymbol{t}$ matches the actual camera displacement: forward movement means positive forward translation $t_z > 0$. Our sensible use-case is forward movement: FOE is inside the frame, and $t_z > 0$.

3.3.1 From disparity to depth

Let us consider the norm of $\boldsymbol{t}$, denoted t_m (m stands for magnitude). We have $t_m^2 = t_x^2 + t_y^2 + t_z^2$. We also consider the optical center $\boldsymbol{p}_0$, given by $\boldsymbol{p}_0 = (u_0, v_0)$. Finally, we assume $\theta >> |t_z|$, since the opposite would mean objects potentially going behind the camera.

From definition 3 of $\boldsymbol{\Phi_t}$, we can deduce that

$$t_m^2 = t_z^2 \left(1 + \frac{\|\boldsymbol{\Phi_t} - \boldsymbol{p}_0\|^2}{f^2}\right)$$
$$|t_z| = \frac{f t_m}{\sqrt{f^2 + \|\boldsymbol{\Phi_t} - \boldsymbol{p}_0\|^2}}$$

Besides, from Eq.3.8, we have

$$(\theta(\boldsymbol{p}) + t_z) D(\boldsymbol{p}) = |t_z| \|\boldsymbol{\Phi_t} - \boldsymbol{p}\|$$

$$\theta(\boldsymbol{p}) = |t_z| \left(\frac{\|\boldsymbol{\Phi_t} - \boldsymbol{p}\|}{D(\boldsymbol{p})} - sign(t_z)\right) \tag{3.12}$$

In the end, we can deduce depth θ from disparity D, focal length f, FOE $\boldsymbol{\Phi_t}$ and translation magnitude t_m

$$\theta(\boldsymbol{p}) = \frac{f t_m}{\sqrt{f^2 + \|\boldsymbol{\Phi_t} - \boldsymbol{p}_0\|^2}} \left(\frac{\|\boldsymbol{\Phi_t} - \boldsymbol{p}\|}{D(\boldsymbol{p})} - sign(t_z)\right) \tag{3.13}$$

This result is in a useful form for limit values. Lateral movement corresponds to $\|\boldsymbol{\Phi_t}\| \to +\infty$ and then

$$\lim_{\|\boldsymbol{\Phi_t}\| \to +\infty} \theta(\boldsymbol{p}) = \frac{f t_m}{D(\boldsymbol{p})}$$

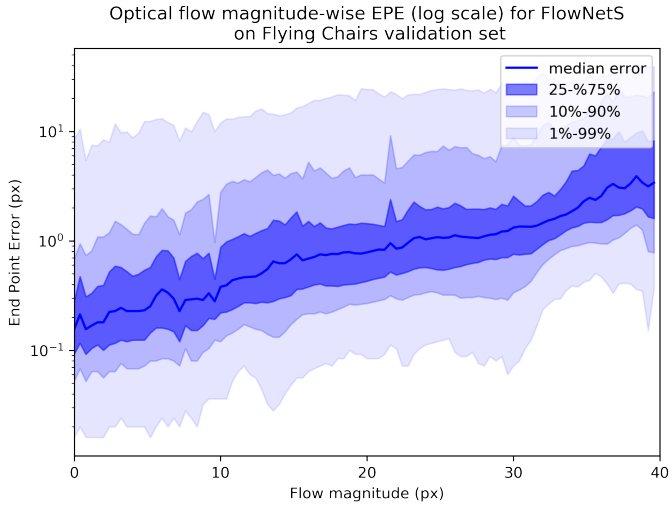


Figure 3.5: Percentiles of optical flow magnitude wise end point error (EPE) of our implementation of flownet, tested on 100 images of Flying Chairs dataset. Mean EPE on Flying Chairs is below 2 pixels.

On the other hand when $\boldsymbol{\Phi_t}$ is inside the frame boundaries, when $\boldsymbol{p} \to \boldsymbol{\Phi_t}$, knowing depth is a bounded positive value, we can deduce:

$$D(\boldsymbol{p}) \underset{\boldsymbol{p}\to\boldsymbol{\Phi_t}}{\propto} \|\boldsymbol{p} - \boldsymbol{\Phi_t}\|$$

Limit of disparity in this case is 0 and we use its inverse. As a consequence, small errors on disparity estimation will result in diverging values of depth near FOE while it corresponds to the direction the camera is moving to, which is clearly problematic for depth-based obstacle avoidance. Figure 3.5 illustrates how FlowNet errors behave with respect to target optical flow magnitude, and especially the fact that null ground truth optical flow can have great end point error, up to 10 pixels.

An other aspect of this equation shows that for a forward movement, $sign(t_z) = 1$, and thus disparity values can trigger a negative depth. This is expected as the dezooming from apparent backward movement (since we compute a reversed optical flow to get depth of last image) normally constraints the points to only get closer to the FOE while not going across it: an infinite depth makes the point $\boldsymbol{p}$ motionless, while a null depth moves the point to the FOE. As such, careless dense optical flow is almost guaranteed to output negative near the FOE in our sensible use-case.

An other issue can be the error from the estimation of the FOE from translation. Indeed, with a shifted $\boldsymbol{\Phi_t}$, depth in its true position will be infinite since there the disparity D will be 0. This problem can however be partly solved when finding $\boldsymbol{\Phi_t}$ analytically, since every optical flow vector is supposed to be aligned with $\boldsymbol{\Phi_t}$ (see equation 3.8).

$$\forall \boldsymbol{p} \in \Omega, \boldsymbol{F}(\boldsymbol{p}) \times (\boldsymbol{\Phi_t} - \boldsymbol{p}) = 0$$

Where $\times$ is the 2D determinant.

3.3.2 From flow divergence to depth

If we consider inverse depth $\frac{1}{\theta}$ to be spatially evolving slowly compared to optical flow, we can deduce it from divergence of $\boldsymbol{F}$:

$$\text{div}(\boldsymbol{F}) = \frac{t_z \text{div}(\boldsymbol{\Phi_t} - \boldsymbol{p})}{\theta(\boldsymbol{p}) + t_z} = \frac{-2t_z}{\theta(\boldsymbol{p}) + t_z}$$

Here we are in a complementary situation: for lateral movement, t_z is null, and so is divergence of $\boldsymbol{F}$, but for forward movement, this technique can be used around the FOE [SRSP16; SK07]. Obviously this method will fail with a non-constant depth (and thus a non-zero divergence), its use is more adapted for very close incoming obstacle and urgent stop maneuver, when t_z is of the same order of magnitude as z, and thus the obstacle likely to get reached within a few frames.

3.3.3 Conclusions on limitations

Even though optical flow is considered well predictable from networks like FlowNet [Dos+15], its relation to depth induces inevitable errors when the movement is random, and computing depth from disparity is much harder than for a classic stereo rig. Some earlier works already enlightened the difficulty in estimating depth solely with flow [SK07; Zin+10], especially when the camera is pointed toward movement. One can note that rotation compensation was already used with fish-eye camera in order to have a more direct link between flow and depth.

Methods based on optical flow divergence can help around the FOE area, but are far from perfect, especially when a fine long range depth estimation is needed.

3.4 End-to-end learning of depth generation

From last section findings, we decide to study the possibilities in learning depth solely from a neural network.

We then focus on RGB-D datasets that would allow supervised learning of depth. RGB pairs (preferably with the corresponding displacement) being the input, and D the desired output. Our choice today to learn depth from motion in existing RGB-D datasets is either unrestricted w.r.t. ego-motion [Fir16; Stu+12], or a simple stereo vision, equivalent to lateral movement [GLU12; SS02]. To our knowledge, no dataset proposes only translational movement.

An interesting work on a structure from motion network has been published and considers several IMU-enabled datasets [Umm+17]. The network, called DeMoN tries to get structure from a pair of images, but not necessarily stabilized. As such, several of these datasets could be used for our network, performing an online stabilization from IMU so that we would be back in our original use-case [XOT13; Stu+12; Aan+16]. This is definitely an interesting solution, but a careful read of their article however enlightens the necessity of large synthetic datasets for their training in addition to these realistic datasets, named BlendSwap and Scenes11 (which don't have a dedicated publication). For each part of the network, it follows the curriculum learning workflow, training on a large unrealistic dataset before fine-tuning on a smaller harder real one: the joint training using all datasets at the same time was probably motivated by their validation set that includes samples from different datasets, and thus needs to stay robust, at the cost of not being specialized for a particular situation.

Another problem when considering leveraging from this work is that these datasets are all very suited for the "multi view synthesis" problem, where the general movement of the camera is to rotate around a subject in order to scan it. This is then not very representative of an obstacle avoidance context, with camera going forward and not only sideways.

StillBox Dataset		
image size	number of scenes	total size (GB)
64x64	80K	19
128x128	16K	12
256x256	3.2K	8.5
512x512	3.2K	33

Table 3.1: Datasets sizes

Scenes parameters	
camera field of view	90^o
max render distance	$200m$
primitives number	20
texture ratio	0.5
size range of mesh objects (m)	$[0, 2]$
distance range of mesh objects (m)	$[0, 25]$
displacement between two consecutive frames	$10cm$
scene length (frames)	10
nominal shift	3
speed equivalent (for 30fps)	$9m.s^{-1}$

Table 3.2: StillBox dataset parameters

3.4.1 Introducing StillBox, a depth synthetic dataset

In light of available literature for stabilized RGB-D datasets, we propose a new dataset which aims at proposing a suited environment for our navigation context: stabilized videos or perfectly known orientations, random movements, and random scenes.

This dataset has been shared with the scientific community and is available to download at `http://stillbox.ensta.fr`.

We called this dataset **StillBox**.

For this dataset, we used the rendering software *Blender* [2] to generate an arbitrary number of random rigid scenes, composed of basic 3d primitives (cubes, spheres, cones and tores) randomly textured from an image set scrapped from *Flickr* [3] (see figure 3.6).

These objects are randomly placed and sized in the scene, so that they are mostly in front of the camera, with possible variations including objects behind camera, or even camera inside an object. Scenes in which camera goes through objects are discarded. To add difficulty we also applied uniform textures on a proportion of the primitives. Each object thus has a uniform probability (corresponding to texture ratio) of being textured from a color-ramp and not from a photograph, along with a random size and position uniformly sampled from specified ranges.

Walls are added at large distances as if the camera was inside a box (hence the name). The camera is moving at a fixed speed value, but to a random direction (uniform distribution), which is constant for each scene. It can be anything from forward/backward movement to lateral movement (which is then equivalent to stereo vision). Tables 3.1 and 3.2 show a summary of our scenes parameters. They can be changed at will, and are

[2] `https://www.blender.org/`, retrieved 07/24/2019

[3] `https://www.flickr.com/`, retrieved 07/24/2019

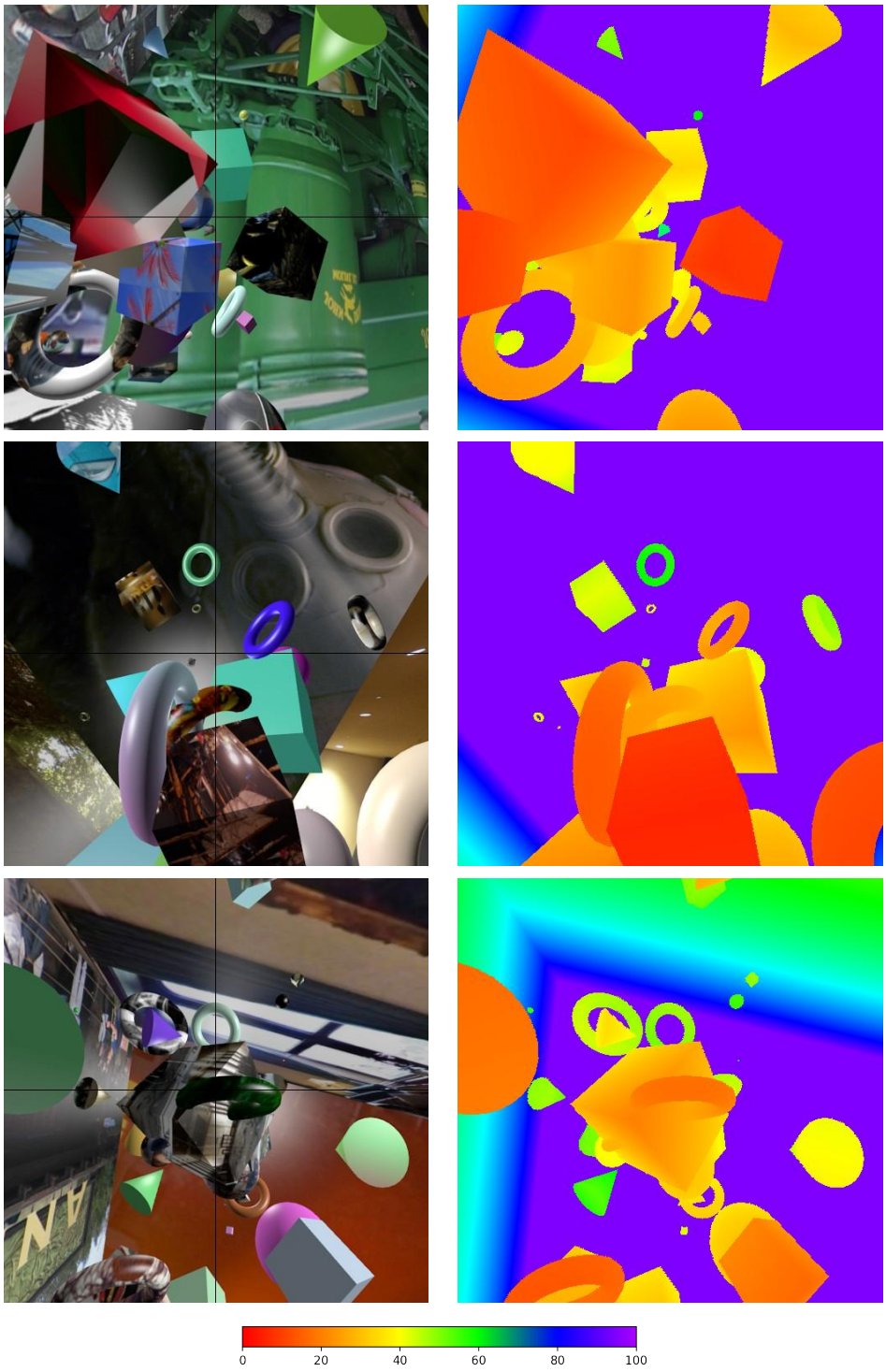


Figure 3.6: Some examples of our renderings with associated depth maps, from 0 to 100m. Visible reticle is not comprised in raw images

stored in a metadata JSON file to keep track of it. Our dataset is then composed of 4 sub-datasets with different resolutions, 64px dataset being the largest in term of number of samples, 512px being the heaviest in data.

The parameters described here present the version of this dataset we open sourced. However, thanks to the fact that we constructed it ourselves, we are able to generate very quickly a new dataset with custom camera properties such as focal length, image size or movement, potentially with rotation if we want to simulate a bad stabilization.

3.4.2 Dataset set augmentation

The way we store data in 10 images long videos, with each frame paired with its ground truth depth allows us to set *a posteriori* distances distribution with a variable temporal shift between two frames. A shift of 1, which means two consecutive frames, will have a lower temporal distance than a shift of 10 (the maximum shift in our dataset). If we use a baseline shift of 3 frames, we can e.g. assume a depth three times as great as for two consecutive frames.

In addition, we can also consider negative shift, which will only change displacement direction without changing speed value compared to opposite shift. This allows us, given a fixed dataset size, to get more evenly distributed depth values to learn, and also to de-correlate images from depth, preventing any over-fitting during training, that would result in a scene recognition algorithm and would perform poorly on a validation set.

3.5 DepthNet, a depth fully convolutional neural network

Now that we have a dataset that mimics UAV navigation, we can train a network to get depth from motion. This network is very inspired from FlowNetS presented in previous section 3.2.1.

3.5.1 The constant speed assumption

The network will accept two images $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ as input, and will output a depth map θ relative to $\boldsymbol{I}_2$. We want a network that assumes a constant displacement between frames. This will have two advantages:

- The way we augment the dataset, depth will be linked to apparent motion. For example if $\boldsymbol{I}_1 = \boldsymbol{I}_2$ the depth output by the network should be infinite.
- We make the assumption that since this constant speed assumptions makes depth very closely linked to disparity, a network performing well on optical flow will be a good candidate for depth.

3.5.2 On the optimal network output

As mentioned by authors of DeMoN [Umm+17], inverse depth might be a good value for our network to output, because its value is more linked to disparity. Besides, it gives less importance to far objects, which might be a good strategy to have good relative error such as MRE and MLE (see section 2.4.2)

The major problem with this approach is that it does not provide an "optimal" depth on which the network should be the most reliable: the closer an object is, the more contrasted output values the network should have, and thus the more weights in the network should be dedicated to this depth distribution. In our case, betting on close

Outputting method	Regular depth	Inverse depth	Log Depth
Loss function $\mathcal{L}$	$\lvert Out(\boldsymbol{p}) - \theta(\boldsymbol{p})\rvert$	$\left\lvert Out(\boldsymbol{p}) - \frac{1}{\theta(\boldsymbol{p})}\right\rvert$	$\lvert Out(\boldsymbol{p}) - log(1+\theta(\boldsymbol{p}))\rvert$
Inverse Post processing	$IPP(\theta(\boldsymbol{p})) = \theta(\boldsymbol{p})$	$IPP(\theta(\boldsymbol{p})) = \frac{1}{\theta(\boldsymbol{p})}$	$IPP(\theta(\boldsymbol{p})) = log(1+\theta(\boldsymbol{p}))$
Post processing	$\widetilde{\theta}(\boldsymbol{p}) = Out(\boldsymbol{p})$	$\widetilde{\theta}(\boldsymbol{p}) = \frac{1}{Out(\boldsymbol{p})}$	$\widetilde{\theta}(\boldsymbol{p}) = exp(Out(\boldsymbol{p})) - 1$

Table 3.3: Comparison of three possible depth representations for the network output

objects is probably not optimal in order to be able to avoid obstacles long before we reach them. We need to choose a more suiting hierarchy: our preferred depth quality measurement being the logarithmic error, we test three separate outputting methods, presented table 3.3: regular depth, inverse depth, logarithmic depth. Each method gets as a training loss the L1 difference between the outputting method and its corresponding ground truth. We thus need to design a post processing function to get the actual depth from the network output, and an inverse post processing function to get the value with which the network output will be compared for each loss function.

3.5.3 General structure

Our network is described figure 3.8. As mentioned above, this network architecture has been proven very efficient for flow and disparity computing while keeping a very simple supervised learning process. The main point of this experimentation is to show that end-to-end depth estimation can be beneficial regarding unknown translation. In the general effort of having a simple network that motivated our choice of inspiration from FlowNetS in the first place, all convolution are now 3×3 kernels, even the first ones that are 7×7 and 5×5 in FlowNetS, and activation functions are simple ReLU[KSH12] instead of LeakyReLU [Sch15]. Most importantly, our architecture differs from FlowNetS by its width, where each layer has half the feature maps. These simplifications makes DepthNet a relatively light network (at least 4 times lighter than FlowNetS) with only standard computation blocks for a facilitated potential mobile deployment. The motivation for a thinner network also relies in the fact that a stabilized footage offers a much simpler optical flow pattern, as shown in equation 3.8. It can be noted that we added Spatial Batch Normalization[IS15] in the convolution blocks to ease training (see figure 3.9), knowing that it can easily be replaced by simple addition and multiplication during the inference. Note that the "OutN" convolutions before the "1+ELU" activation functions neither have BatchNorm nor ReLU, and the "Up OutN" don't have BatchNorm.

The ELU trick

For final output, we use the activation function "1 + ELU" [CUH15]. It outputs only values above 0, which is to be expected, but unlike the ReLU, we avoid the null gradient when raw network output (before the activation function) is below zero. Besides, 1+ELU is still very close to a ReLU: it only differs for output values below 1, and could be replaced by a simple shifted ReLU ($x \mapsto \text{ReLU}(x+1)$) for inference if needed and applicable, ie with regular depth. See figure 3.7 for a graphic comparison. Functions are exactly the same above $y = 1$, meaning that in our dataset, the difference will almost always be negligible.

$$\begin{array}{rccl} \text{1 + ELU}: & \mathbb{R} & \to & \mathbb{R}^+ \\ & x & \mapsto & \begin{cases} 1+x & \text{if } x > 0 \\ e^x & \text{else} \end{cases} \end{array}$$

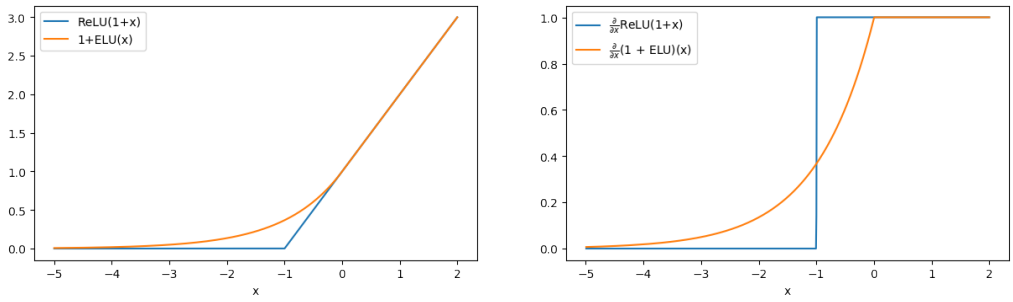


Figure 3.7: Comparison between "1+ELU" and "ReLU(1+x)", functions and derived functions

3.6 Training details

Like for FlowNetS with optical flow, we use a multi-scale criterion, with a L1 reconstruction error for each scale.

$$Loss = \sum_{s \in scales} \gamma_s \frac{1}{|\Omega|} \sum_{\boldsymbol{p} \in \Omega} |Out_s(\boldsymbol{p}) - IPP(\theta(\boldsymbol{p}))|$$

where

- Out_s is the output map at scale s of the network, upscaled bilinearly to match dimension of θ. Note that depending on the post processing function, this is not yet a depth map *per se*.
- IPP is the inverse post processing function. (See 3.5.2)
- γ_s is the weight of the scale, we arbitrarily chose higher weights for higher scales: $(\gamma_s)_{2 \leq s \leq 6} = [0.32, 0.08, 0.02, 0.01, 0.005]$ for s varying from 2 (highest scale) to 6 (lowest scale)
- θ is the depth ground-truth.

As said earlier, we apply data augmentation to the dataset using different shifts, along with classic methods such as flips and rotations. We also clamp depth to a maximum of 100m, and provide sample pairs without shift, assuming its depth is 100m everywhere. We use a stochastic gradient descent algorithm with a momentum of 0.9. The learning rate varies from 10^{-3} to 10^{-2}, depending on the post processing function.

We used the framework PyTorch [Pas+17], the full source code is available on *Github*[4], along with the StillBox dataset.

3.6.1 Influence of resolution on training

The special case of 64x64 images

Figure 3.10 shows results from 64px dataset. Like FlowNetS, results are down-sampled by a factor of 4 and up-sampled to 64x64 with bilinear interpolation, just like during training.

One can notice that although the network is still fully convolutional, feature map sizes go down to 1x1 and then behave exactly like a Fully Connected Layer, which can serve to figure out implicitly motion direction and spread this information across the outputs.

[4] `https://github.com/ClementPinard/DepthNet`, retrieved 07/24/2019

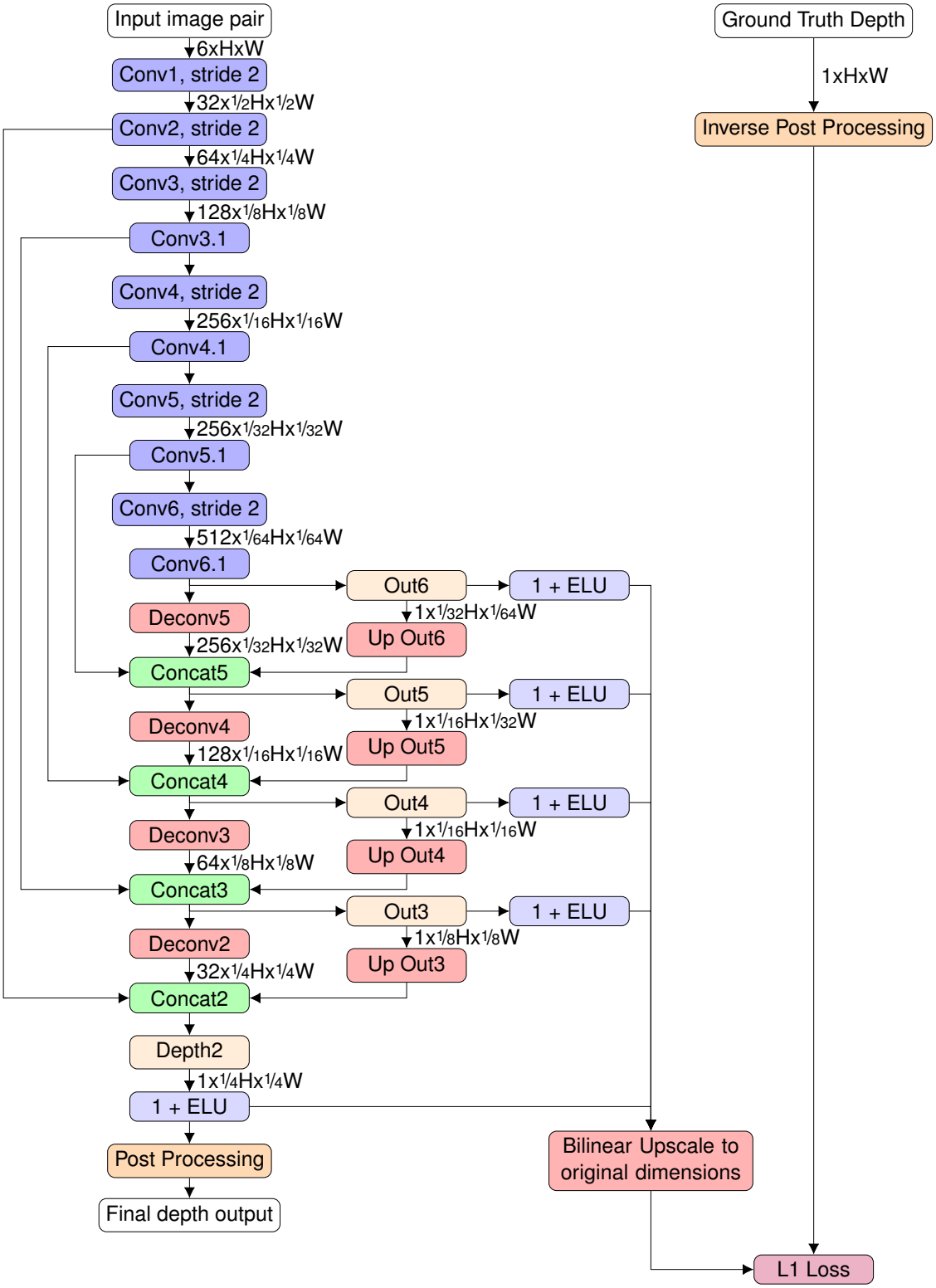


Figure 3.8: DepthNet structure parameters

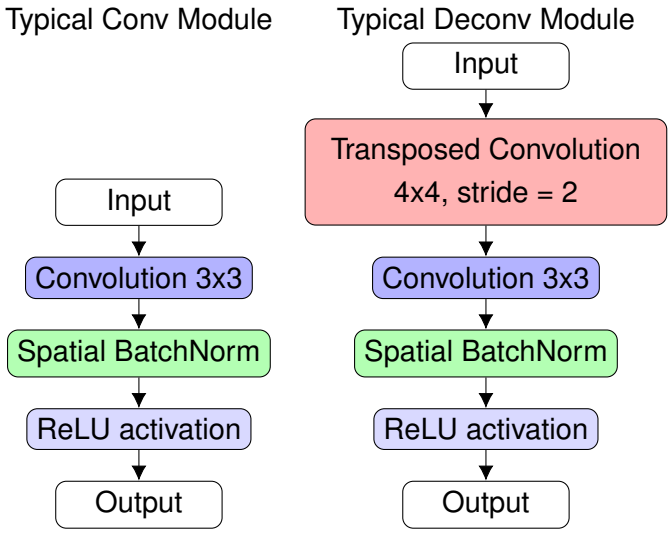


Figure 3.9: General Structure of convolution and deconvolution blocks of DepthNet.

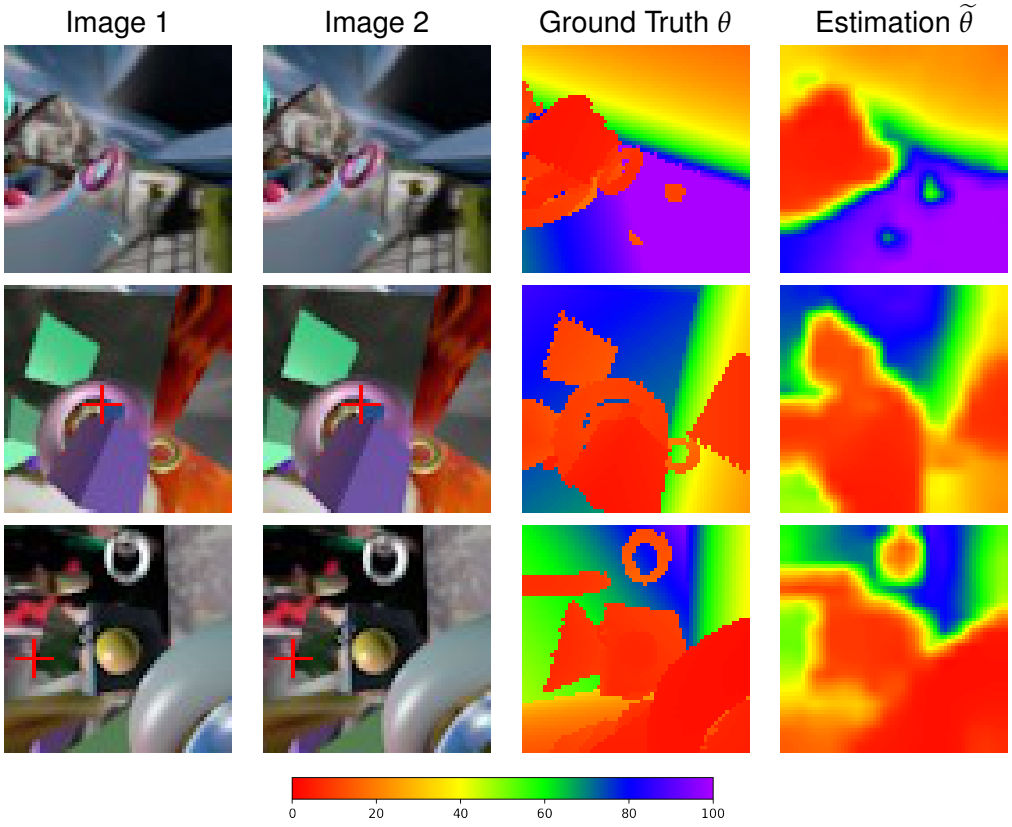


Figure 3.10: Qualitative results of DepthNet for 64x64 images. FOE is indicated by a red cross on Image1 and Image 2

Outputting method	MAE	MRE	MLE
Identity	**4.72**	**0.26**	0.32
Inverse	16.26	0.81	0.62
Log	6.91	0.37	**0.28**

Table 3.4: Quantitative results for 64x64 images.

Network	MEA	MRE	MLE
FlowNetS$_{64}$	5.72	0.363	0.276
DepthNet$_{64}$	4.72	0.261	0.323
DepthNet$_{64\rightarrow128}$	3.70	0.266	6.30
DepthNet$_{64\rightarrow128\rightarrow256}$	2.70	0.209	0.174
FlowNetS$_{64\rightarrow128\rightarrow256\rightarrow512}$	2.26	**0.167**	**0.142**
DepthNet$_{64\rightarrow128\rightarrow256\rightarrow512}$	**2.12**	0.168	**0.142**
DepthNet$_{64\rightarrow512}$	2.39	0.177	0.150
DepthNet$_{512}$	3.85	0.317	0.213

Table 3.5: quantitative results for depth inference networks. FlowNetS is modified with 1 channel outputs (instead of 2 for flow), trained from scratch for depth with Still Box

The second noticeable fact is that near FOE, the network has no problem inferring depth. It means that it uses both optical flow divergence and interpolates with neighbor depth values, where disparity is easier to measure.

This can be interpreted as 3D shapes identification, along with their magnification: pixels belonging to the same shape are deemed to have close and continuous depth values, resulting in a FOE-independent depth inference. Figure 3.11 shows the depthwise relative error. While quality is not optimal at very low depth (and thus high displacement), the network is very precise even at very high depth. A relative error of 0.2 at depth of 90 means that the equivalent error on pixel displacement is less than $\frac{0.2\times f\times 0.3}{90} = 0.02 pixels$. The error peak at 12m can be interpreted by the fact that most objects (and thus the scene's main source of diversity and estimation noise) are between 10 and 20 meters. It can be also reinforced by the fact that bilinear interpolation of our network's output (which is downscaled by a factor of 4) typically presents a constant slope at depth discontinuities, resulting in high error at object borders.

Table 3.4 shows quantitative results on a validation set with our three different post processing functions. It can appear surprising that in the end, inverse depth is clearly not the most efficient, especially when knowing that literature almost always tries to learn disparity (in the case of stereo vision) or inverse depth. Its poor results here can be explained by the mean disparity which is very low on our dataset and thus the L1 Error is too noisy. This result is thus particularly interesting for low disparity cases, where learning the inverse of disparity might be more efficient than learning it directly. As an example, we can cite the dataset CHairSDHorn, published with FlowNet2 [Ilg+16] which features small displacements.

For our two other losses, results are somewhat comparable. For sake of simplicity, we keep the identity post processing.

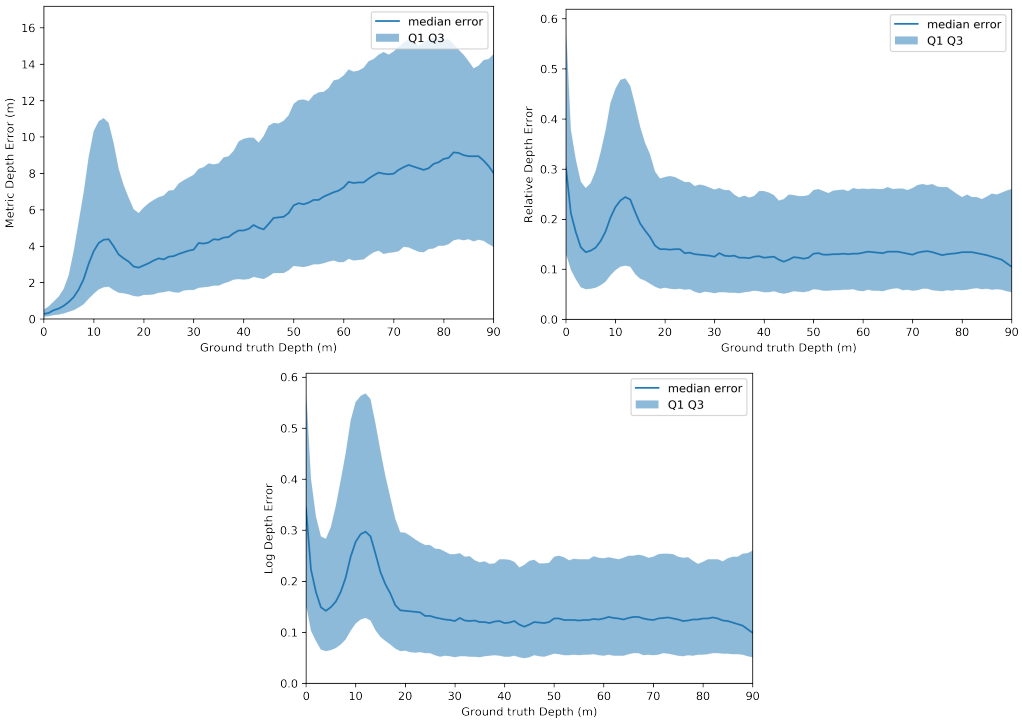


Figure 3.11: Median depthwise metric, relative and logarithmic errors of DepthNet on 64x64 images with Identity post processing. Shaded areas are delimited by first and third quartiles.

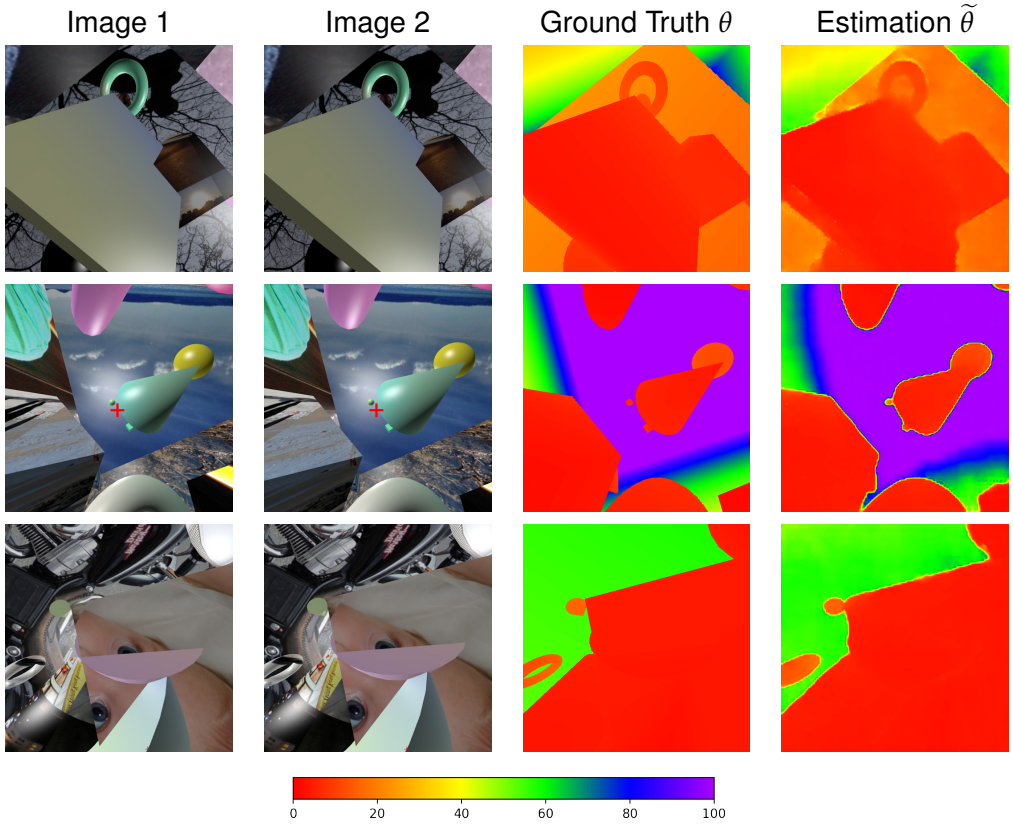


Figure 3.12: Qualitative results of finetuned DepthNet$_{64\to128\to256\to512}$ for 512x512 images. FOE is indicated by a red cross on Image 1 and Image 2

Network	size	980Ti		Quadro M1000M		TX1	
		1	8	1	8	1	8
DeMoN	48.0	11	4.8	N/A	N/A	N/A	N/A
FlowNetS_{64}	39.4	225	153	86	41	27	14
DepthNet_{64}	**7.33**	364	245	213	88	70	40
FlowNetS_{512}	39.4	69	8.8	15	N/A	2.8	N/A
DepthNet_{128}	**7.33**	294	118	205	74	51	15
DepthNet_{256}	**7.33**	178	36	144	29	39	3.2
DepthNet_{512}	**7.33**	68	8.8	58	8.2	9.2	N/A

Card model	GFLOPS (FP32)	Memory (GB)
GTX 980Ti	5,632	6
Quadro M1000M	1,390	2
Tegra X1	512	8

Table 3.6: Size (millions of parameters) and Inference speeds (fps) on different devices. Batch sizes are 1 and 8 (when applicable). A batch size of 8 means 8 depth maps are computed at the same time. DeMoN was tested only on a 980Ti. On the second table, a quick comparison between tested GPUs

From 64px to 512px Depth inference

One could think that a fully convolutional network such as ours can not solve depth extraction for pictures greater than 64x64. The main concern is that for a fully convolutional network, each pixel is applied the same operation. For disparity, this makes sense because the problem is essentially similarity from different picture shifts. However, for depth inference when FOE is not diverging (forward movement is non negligible), result from equation 3.13 apparently shows that once the FOE is known, we then get different operations to do depending on distance from it and from the optical center $\boldsymbol{p}_0$. The only possible strategy for a fully convolutional network would be to compute the position in the frame as well and to apply the compensating scaling to the output.

This problem then seems very difficult, if not impossible for a network as simple as ours, and if we run the training directly on 512x512 images, the network fails to converge to better results than with 128x128 images while better resolution would help getting more precision. However, if we take the converged network and apply a fine-tuning on it with 512x512 images, we get much better results. Figure 3.12 shows training results and shows that our deemed-impossible problem seems to be easily solved with multi-scale fine-tuning. As Table 3.5 shows, best results are obtained with multiple fine-tuning, with intermediate scales 64, 128, 256, and finally 512 pixels. Subscript values indicate fine-tuning processes. FlowNetS is performing comparably to DepthNet while being 5 times heavier and most of the time much slower, as shown Table 3.6.

3.6.2 Comparison with DeMoN and depth from optical flow

To show that a network dedicated to output depth map on stabilized videos can outperform more traditional methods, we compare our network to two methods:

- A mixture of FlowNetS and depth from optical flow (not to be confused with FlowNetS

Method	MAE	MRE	MLE
DepthNet	**2.67**	**0.162**	**0.132**
FlowNetS $\rightarrow$ depth	6.462	0.203	0.172
DeMoN	18.85	1.57	0.726

Table 3.7: Quantitative comparison between the three methods. FlowNetS $\rightarrow$ depth row is deducing depth from optical flow and displacement (perfectly known)

trained for depth used section 3.5) . FOE is supposed perfectly known. FlowNetS is finetuned for optical flow on still box.

- DeMoN network, which is supposed to work on any movement. But as discussed above, DeMoN might be more specialized on lateral movement, and not forward movement. Because of its heaviness which makes it unsuitable for light embedded integration, we did not take the time to fine-tune DeMoN on still box. The main point of comparing with DeMoN is to show that even though its inner architecture is based on optical flow, depth does not diverge near FOE. This is most probably thanks to the refinement network that regularizes the optical flow map and outputs a plausible depth map instead of deducing it geometrically from optical flow.

For this comparison, and because DeMoN which we don't retrain is supposed to work on a specific set of intrinsic parameters, we constructed a new still box dataset for fine tuning and validation that matches DeMoN's expected intrinsics. In order to help DeMoN, which might confuse translation and rotation while we know there is none, we feed the iterative network the right rotation (null) instead of the estimated value.

Figure 3.13 compares the three methods on a scene with lateral movement, and a scene with forward movement. We can see how near FOE, depth diverges for depth from optical flow.

Table 3.7 displays quantitative quality of these three methods. As both depth from FlowNet and DeMoN can output negative value that would result in an infinite log error, we clamped the depth to be between 0.1 and 100.

Finally, figure 3.14 shows the error with respect to distance to FOE. As shown, the depth from optical flow method, although with a better average score than DeMoN has poor values near FOE. This is an important result as it shows that even for optical flow based network like DeMoN, a refinement step makes it possible to output a better quality depth map near FOE. This shows that a depth from optical flow is salvageable with enough refinement.

3.7 UAV navigation use-case

Figure 3.15 shows qualitative results from our validation set, and from real conditions drone footage, on which we were careful to avoid camera rotation. These results did not benefit from any fine-tuning from real footage, indicating that our Still Box Dataset, although not realistic in its scenes structures and rendering, appears to be already sufficient for learning to produce decent depth maps in real conditions, the same way FlowNetS trained Flying Chairs [Dos+15] dataset did get decent results on more realistic optical flow datasets, like MPI Sintel [But+12] and KITTI [Gei+13].

We trained depth estimation from a moving camera, assuming its velocity is always the same. When running during flight, such a system can easily deduce the real depth map from the drone displacement magnitude t, knowing that the training speed was V_0

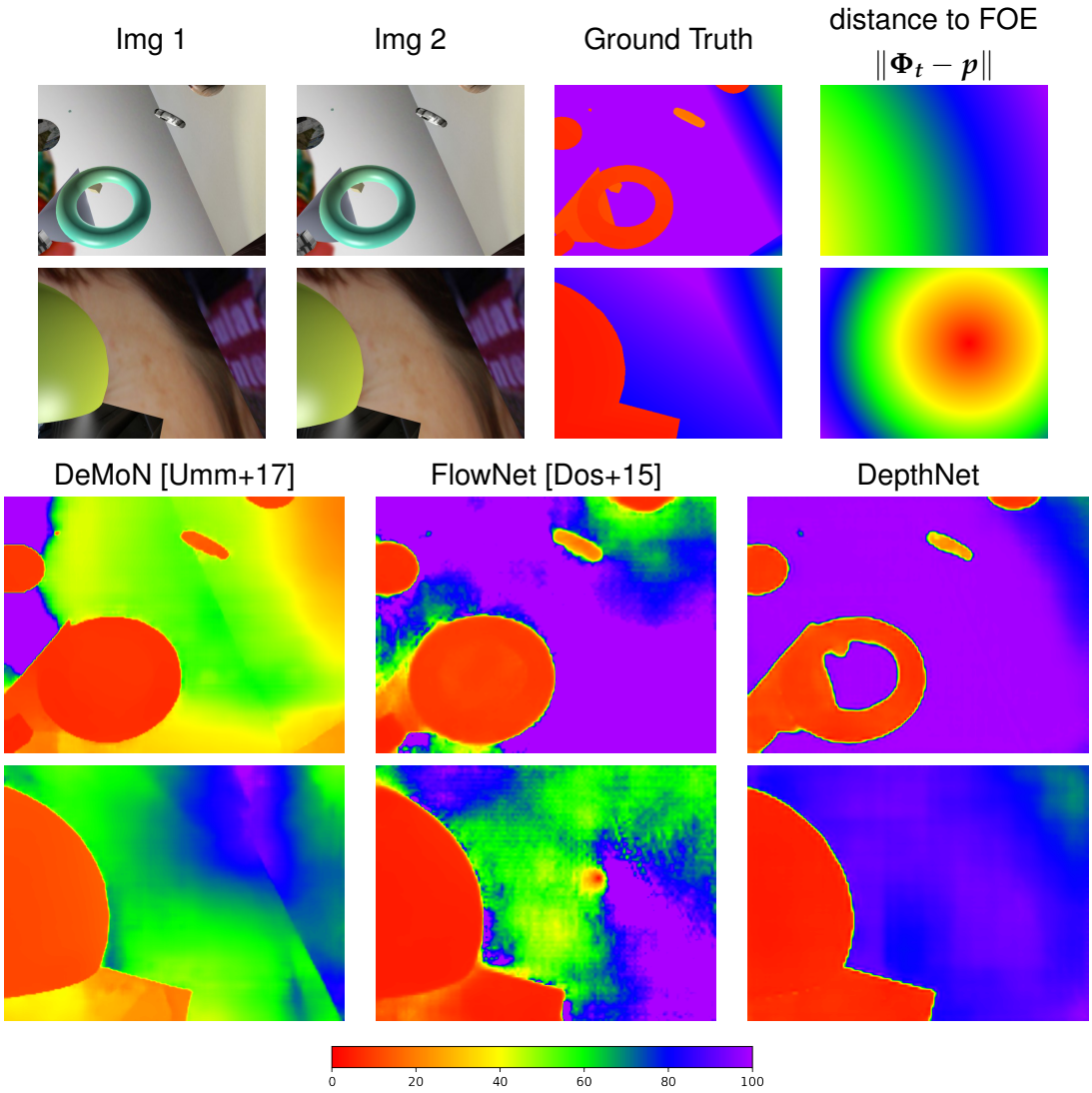


Figure 3.13: Qualitative comparison between DeMoN [Umm+17], FlowNet [Dos+15] and DepthNet.

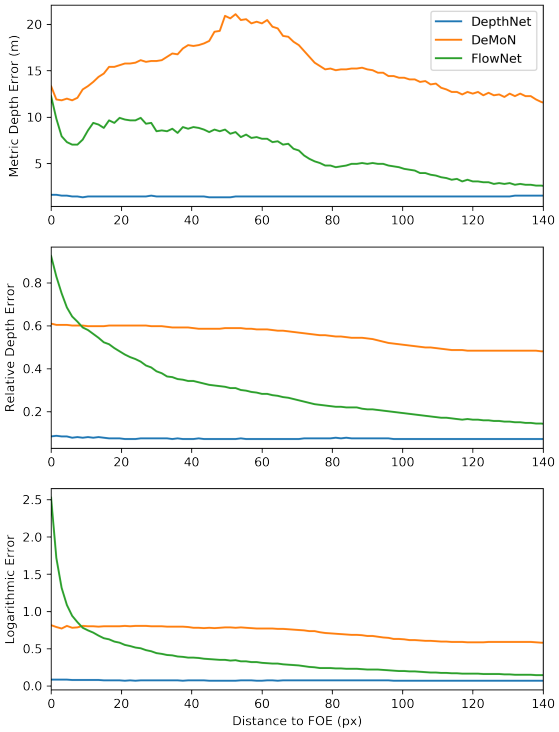


Figure 3.14: Depth errors with respect to distance to FOE on our test set. Contrary to FlowNet, DepthNet and DeMoN do not depend on it

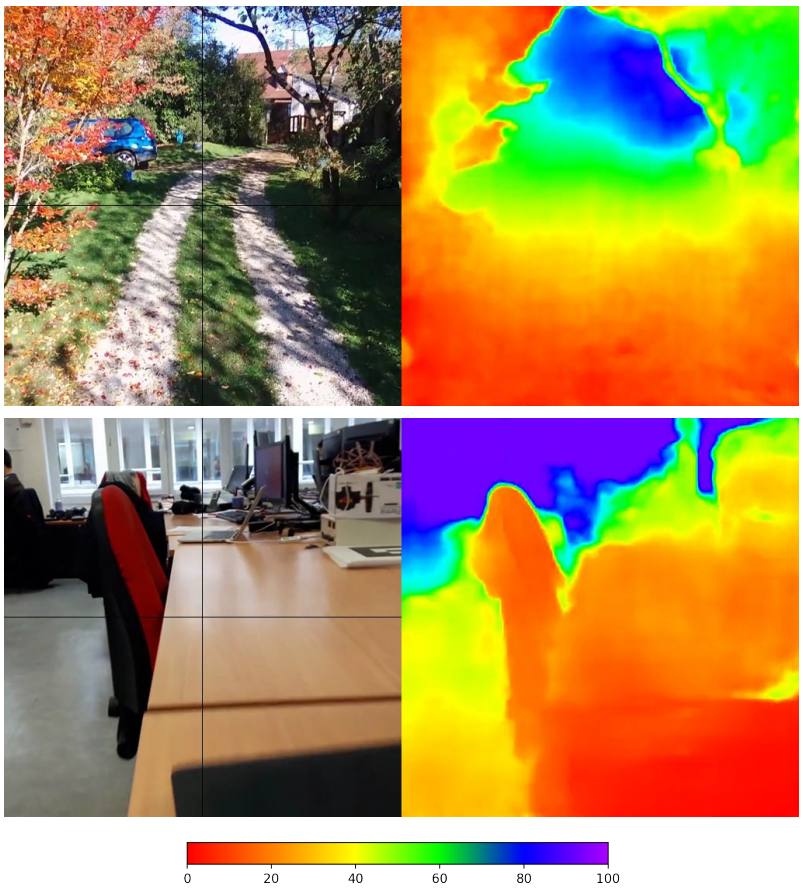


Figure 3.15: some results on real images input. Top is from a Bebop drone footage, bottom is from a gimbal stabilized smartphone video.

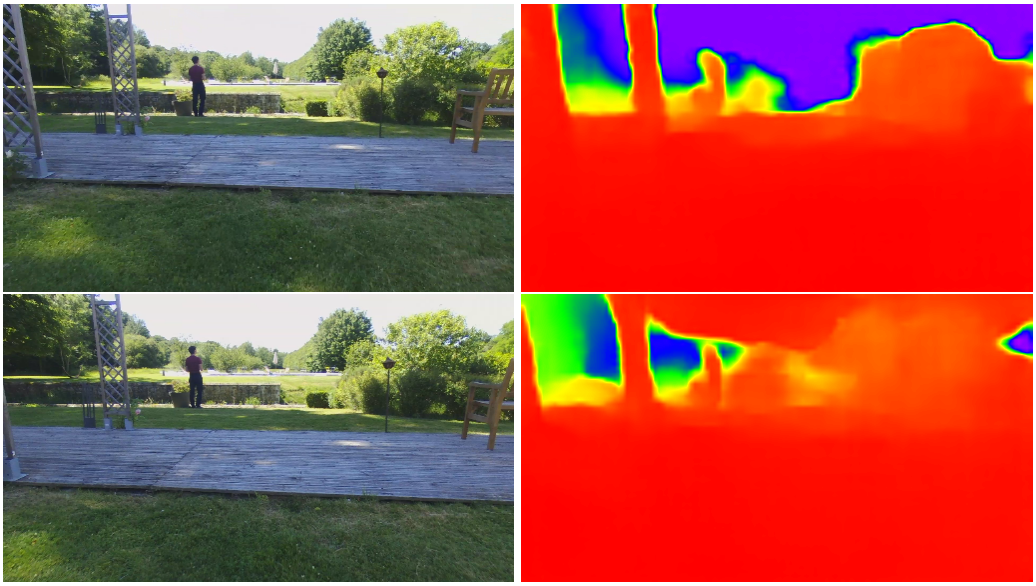


Figure 3.16: Example of depth flickering for the sky. The two frames are $\approx 83ms$ apart.

(here $9m.s^{-1}$)

$$depth(t) = \frac{V_t}{V_0} DepthNet(frame_t, frame_{t-1}) \tag{3.14}$$

One of the drawbacks of this learning method is that the f value (which is focal length divided by sensor size per pixel) of our camera must be the same as the one used in training. Our dataset creation framework however allows us to change this value very easily for training. One must also be sure to have pinhole equivalent frames like during training.

However, since the network is fully convolutional, the inference will work as long as the focal length is the same. More specifically, this is supposed to be robust to optical center errors.

Figure 3.16 shows a typical failure case, where the sky is thought to be a very close object. This shows that although already decent on real videos, out network is probably over-fitted on typical StillBox images, even if it was designed with diversity in mind. More specifically in this case, we suspect a small defect on frame stabilization, which the network never had to be robust with when training on StillBox.

3.8 Conclusion of this chapter

We proposed a novel way of computing dense depth maps from monocular image sequences, along with a very comprehensive dataset for stabilized footage analysis. This algorithm can then be used for depth-based sense and avoid algorithm in a very flexible way, in order to cover all kinds of path planning, from collision avoidance to long range obstacle bypassing.

The proposed network outperforms more general depth from motion algorithms, and is yet very simple with only convolutions, transposed convolutions and ReLU activations.

We now have a very powerful pretraining for depth, the same way Flying chairs [Dos+15] was for Optical Flow. The next chapter will cover the ways of fine-tuning this network in order to perform well on real videos, and especially how to leverage videos without ground-truth depth thanks to photometric reprojection based training.

Chapter 4

Training Depth estimation with auto-supervision

Contents

4.1 Motivations . . . 50
4.1.1 Training depth networks without ground-truth . . . 50
4.1.2 Outline of this chapter . . . 51
4.2 Core concept for reprojection based optimization . . . 51
4.2.1 The image as a continuous 2D function . . . 51
4.2.2 Optical flow and reprojection based optimization . . . 53
4.3 Retrieving a depth map using reprojection based optimization . . . 54
4.3.1 Training a neural network to output depth with reprojection based optimization . . . 55
4.3.2 Conclusions of reprojection based optimization . . . 56
4.4 Regularization with smooth loss and diffusion . . . 56
4.4.1 Optical flow smoothing . . . 56
4.4.2 Applying smoothing on a depth map . . . 57
4.4.3 Translating physical assumptions into equations . . . 58
4.4.4 Smooth loss presentation . . . 59
4.4.5 Diffusing the output of a neural network . . . 60
4.4.6 Minimizing gradient with $\mathcal{L}_{s\nabla}$ . . . 62
4.4.7 Minimizing Laplacian with $\mathcal{L}_{s\Delta}$ . . . 63
4.4.8 Diffusion of a vector map with divergence minimization . . . 64
4.4.9 Edge-aware smooth loss . . . 65
4.4.10 Final edge-aware forms of considered smooth losses . . . 67
4.5 Dealing with occlusions . . . 68
4.5.1 Inverse warp based optimization *vs* occlusions . . . 68
4.5.2 Filtering occlusion areas from optimization . . . 70
4.5.3 Self filtered depth map inference . . . 70
4.5.4 Backward/forward consistency . . . 71
4.5.5 Depth map consistency . . . 71
4.5.6 Occlusions detections with direct warp instead of inverse warp . . . 72
4.5.7 Conclusions on occlusion considerations . . . 75
4.6 Determination of optimal hyper-parameters with toy problems . . . 77
4.6.1 Scenes presentation . . . 77

4.6.2 Tests on smooth loss regularization 81
4.7 Photometric losses 86
4.7.1 Multi-scale photometric loss 86
4.7.2 Minimal loss sampling 89
4.7.3 Photometric distance 90
4.8 Conclusions on literature review for unsupervised depth training 93

4.1 Motivations

This chapter aims at studying the possibilities of learning depth without ground truth. This is one of the logical further steps after having shown how a neural network can learn robust depth inference from a supervised training.

Indeed, our network could probably be enhanced by training on real images instead of our unrealistic still box dataset. To do so, one could use supervised learning to train a system for depth from vision on an offline dataset featuring explicit depth measurement, such as KITTI [Gei+13]. However, even setting up such recording devices can be costly and time demanding, which can limit the amount of data the system can be trained on.

One could also try to set up a synthetic but realistic dataset using state of the art game engines and assets [Sha+17]. Unfortunately, this solution could potentially lack diversity, as assets such as 3D scenes are not easy to setup and require human assumptions on what makes a rendering realistic.

The ground-truth-less learning solution, although more difficult in terms of designing a loss function, would have the huge advantage of not needing any sort of preprocessing. As a consequence, extending a dataset of training images would only require to capture them and directly add them to training database.

4.1.1 Training depth networks without ground-truth

Most recent works on autosupervised training networks for computing depth maps use differentiable bilinear warping techniques, which we will talk about in first section, first introduced in [JSZ+15]. The main idea is trying to match two frames using a depth map and a displacement. The new loss function to be minimized is the photometric error between the target frame and the projected one. Depth is then indirectly optimized. Although sensitive to errors coming from occlusions, non Lambertian surfaces and moving objects, this optimization shows great potential, especially when considering how little calibrated data is needed.

For instance [GMB17; Gar+16; XGF16] use stereo views and try to reconstruct one frame from the other. This particular use case for depth training allows to always consider the same displacement and rigid scenes since both images are captured at the same time and their relative poses are always the same. However, in addition to having a fixed displacement, with its already discussed depth range limitations, it constrains the training set to perfectly rectified stereo rigs, which are not as easy to set up as a monocular camera.

When trying to estimate both depth and movement, [Zho+17; YS18; MWA18; Ran+18] also achieved decent results on completely unconstrained ego-motion video. One can note that some methods [Zho+17] are assuming rigid scenes although the training set does not always conform to this assumption. The other ones try to do without this assumption by computing a residual optical flow to resolve the uncertainty from moving objects.

[Vij+17] explicitly considered non rigid scenes by trying to estimate multiple objects movements in the scene, to begin with the motion of the camera itself, which allowed them to deduce a flow map, along with the depth map.

4.1.2 Outline of this chapter

This chapter is dedicated to present as thoroughly as possible the main ideas between frame reprojection, and the existing related techniques in literature. This study will help us evaluate several extensions to the initial reprojection optimization before taking it to a full size deep learning workflow.

1. Core concept will be presented section 4.2, with inverse warp, pixel wise differentiation and optimization.

2. We will then discuss section 4.3 the depth optimization problem as a set of frame reprojection photometric errors to minimize in a particular scene, along with the SFMLearner[Zho+17] use-case.

3. On section 4.4, Smooth constraints, inspired from optical flow regularization will be here studied on inverse depth. A general smooth loss expression will be considered, along with several popular smooth losses which are in fact variations of the general form.

4. The occlusion problem will be addressed section 4.5. We call into question existing solutions in literature and propose an analytical alternative.

5. We will construct section 4.6 a toy problem designed to be challenging for these presented techniques, despite its size of only two scenes.

6. Finally, we will present section 4.7 several extensions to the initial photometric loss used for reprojection optimization

Moreover, a collection of notebooks to recreate this chapter figures are available online [1].

4.2 Core concept for reprojection based optimization

The key feature of unsupervised depth training is the integration of images in the gradient propagation workflow. As a consequence, for any point $\boldsymbol{p} \in \mathbb{R}^2$, we must be able to compute the gradients of $\boldsymbol{I}(\boldsymbol{p})$ with respect to $\boldsymbol{p}$ coordinates.

4.2.1 The image as a continuous 2D function

As in chapter 3, we consider an image $\boldsymbol{I}$ as a function that maps from integer values in Ω, defined eq 3.1 to channel luminance $[0, 255]^3$.

$$\Omega = ([0, W-1] \times [0, H-1]) \cap \mathbb{Z}^2$$
$$cardinal(\Omega) = |\Omega| = H \times W$$

[1] `https://gitlab.ensta.fr/pinard/thesis-notebooks` and `https://gitlab.ensta.fr/pinard/direct-warper`, retrieved 07/24/2019

$$\forall \boldsymbol{p} = (u, v) \in \Omega, \boldsymbol{I}(\boldsymbol{p}) = \boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Our current problem is to extend our image in the whole 2d plan $\mathbb{R}^2$. More specifically, we want to be able to compute for any point $\boldsymbol{p} = (u, v) \in \mathbb{R}^2$ the following values:

$$\boldsymbol{I}(\boldsymbol{p})$$
$$\nabla \boldsymbol{I}(\boldsymbol{p}) = \nabla_p \boldsymbol{I}(\boldsymbol{p}) = \begin{bmatrix} \frac{\partial \boldsymbol{I}}{\partial u} \\ \frac{\partial \boldsymbol{I}}{\partial v} \end{bmatrix}$$

That way, we will be able to tell the variation of the function $\boldsymbol{I}$ according to $\boldsymbol{p}$ and use it in an gradient retropropagation workflow. Here, we present the bilinear interpolation, which is the one used in Spatial Transformer Networks (STN) [JSZ+15].

First, if we consider the image size, the image function $\boldsymbol{I}$ can be extended to $\mathbb{Z}^2$, by setting its value arbitrarily outside of size limitations. Depending on conventions, it can be:

1. $\boldsymbol{I}(\boldsymbol{p}) = 0$, this is used when sampling an isolated point outside of Ω, in order for gradient to be null.
2. $\boldsymbol{I}(\boldsymbol{p}) = \boldsymbol{I}(\boldsymbol{q}), \boldsymbol{q} = \arg\min_{\boldsymbol{q} \in \Omega}(\|\boldsymbol{p} - \boldsymbol{q}\|)$, this is used when computing image gradient at image boundaries.

Interpolation can then extend the function to $\mathbb{R}^2$ by finding possible values between integer positions. Several interpolations techniques can be used, such as nearest or bicubic.

The bilinear interpolation on which we focus on tries to find the quadratic function $f_{i,j} : [0,1]^2 \to \mathbb{R}^3$ between 4 adjacent points as shown in figure 4.1, that solves the equations related to these points, where value is known.

$$\begin{array}{rccl} \boldsymbol{f}_{\boldsymbol{p}=(i,j)\in\Omega} : & [0,1]^2 & \to & \mathbb{R}^3 \\ & (u,v) & \mapsto & \boldsymbol{I}(\boldsymbol{p}) + u\partial_x \boldsymbol{I} + v\partial_y \boldsymbol{I} + uv\partial_{xy} \boldsymbol{I} \end{array}$$

Where

$$\begin{aligned} \partial_x \boldsymbol{I} &= \boldsymbol{I}(i+1, j) - \boldsymbol{I}(\boldsymbol{p}) \\ \partial_y \boldsymbol{I} &= \boldsymbol{I}(i, j+1) - \boldsymbol{I}(\boldsymbol{p}) \\ \partial_{xy} \boldsymbol{I} &= \boldsymbol{I}(\boldsymbol{p}) + \boldsymbol{I}(i+1, j+1) - \boldsymbol{I}(i+1, j) - \boldsymbol{I}(i, j+1) \end{aligned}$$

The extended image function is then the piecewise function composed of all different quadratic solutions to the systems stated above.

$$\begin{array}{rccl} \boldsymbol{I} : & \mathbb{R}^2 & \to & \mathbb{R}^3 \\ & (u,v) & \mapsto & f_{\lfloor u \rfloor, \lfloor v \rfloor}(frac(u), frac(v)) \end{array}$$

where $frac$ is the fractional part defined as $x - \lfloor x \rfloor$ for $x \in \mathbb{R}$.

One should note that this function is continuous everywhere and almost differentiable everywhere, but it's not a $\mathcal{C}^1$ function, gradient generally has discontinuity when approaching $\boldsymbol{p}_{lim} \in (\mathbb{R} \times \mathbb{Z}) \cup (\mathbb{Z} \times \mathbb{R})$. Besides, it can be shown that wherever defined, Laplacian

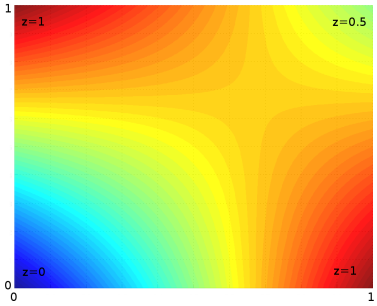


Figure 4.1: Illustration of bilinear interpolation on the unit square. The four corners have fixed z values as indicated in the figure, the values in between are interpolated, and the interpolated values are represented by the color. [2]

is 0 everywhere. As discussed later, for these values to be computed on Ω, and especially when trying to optimize them, we will use the scale space representation [Lin96], which makes the picture $\mathcal{C}^{+\infty}$ everywhere, but has a blurred representation of $\boldsymbol{I}$.

From now on, for the sake of compact notation, we will consider any 2D function on Ω to be a differentiable function, and for any image $\boldsymbol{I}$, the notation $\boldsymbol{I}(\boldsymbol{p})$ with $\boldsymbol{p} = (u, v) \in \mathbb{R}^2$ will implicitly refer to the bilinear interpolation extension of this image.

4.2.2 Optical flow and reprojection based optimization

Thanks to its differentiability, an optimization by gradient descent can be applied to the bilinear interpolation of an image. For a particular color value $\boldsymbol{c}$ in $\mathbb{R}^3$, one can try to find the optimal $\boldsymbol{p}$ coordinates where $\boldsymbol{p} = \arg\min_{\boldsymbol{p}} \{\|\boldsymbol{I}(\boldsymbol{p}) - \boldsymbol{c}\|\}$.

An extension to this simple optimization is to consider an array of points to retrieve from one image $\boldsymbol{I}_t$ to another $\boldsymbol{I}_{t+1}$. Indeed, each point to retrieve is defined as a color that is assumed to be the same in the other image. This optimization then tries to compute optical flow $\boldsymbol{F}$.

$$\forall \boldsymbol{p} \in \Omega, \boldsymbol{I}_{t+1}(\boldsymbol{p} + \boldsymbol{F}(\boldsymbol{p})) = \boldsymbol{I}_t(\boldsymbol{p})$$

$$\forall \boldsymbol{p} \in \Omega, \boldsymbol{F}(\boldsymbol{p}) = \arg\min_{\boldsymbol{q} \in \mathbb{R}^2} \|\boldsymbol{I}_t(\boldsymbol{p}) - \boldsymbol{I}_{t+1}(\boldsymbol{p} + \boldsymbol{q})\| \tag{4.1}$$

Note that optical flow $\boldsymbol{F}$, similarly to images is a 2D function defined on Ω, thus all aforementioned notations can be used, including interpolation. In practice, the optimization tries to minimize the following photometric loss, defined by $\mathcal{L}_p$.

$$\mathcal{L}_p = \sum_{\boldsymbol{p} \in \Omega} d\left(\boldsymbol{I}_t(\boldsymbol{p}), \boldsymbol{I}_{t+1}\left(\boldsymbol{p} + \boldsymbol{F}(\boldsymbol{p})\right)\right)$$

where d is a distance function such as L1 distance.

By using gradient descent, we then get for each coordinate $\boldsymbol{F} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$ of our optical flow

$$\forall \boldsymbol{p} = (i, j) \in \Omega, \partial_t \boldsymbol{F}(\boldsymbol{p}) = \begin{bmatrix} \partial_t F_1(\boldsymbol{p}) \\ \partial_t F_2(\boldsymbol{p}) \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathcal{L}_p}{\partial F_1}(\boldsymbol{p}) \\ \frac{\partial \mathcal{L}_p}{\partial F_2}(\boldsymbol{p}) \end{bmatrix}$$

This equation can be more generally written as:

[2] Source: `https://commons.wikimedia.org/wiki/File:Bilininterp.png`, retrieved 07/24/2019

$$\partial_t \boldsymbol{F} = -\nabla_{\boldsymbol{F}} \mathcal{L}_p$$

where ∇_T is the gradient operation with respect to the parameter vector $\boldsymbol{T}$. In this notation, $\boldsymbol{F}$ is considered a 1D vector of size $|\Omega| \times 2$

By using Euler's method we then update $\boldsymbol{F}$ as follow:

$$\boldsymbol{F} \leftarrow \boldsymbol{F} - \gamma \nabla_{\boldsymbol{F}} \mathcal{L}_p$$

where γ is the optimization step value.

The optical flow problem is actually the root of reprojection based optimization. Our depth problem, is in fact a composition of a function that outputs an optical flow and a photometric error.

Note on inverse warp

Throughout this subsection, we implicitly introduced the operation of inverse warp. Indeed, when optimizing optical flow from $\boldsymbol{I}_t$ to $\boldsymbol{I}_{t+1}$, we constructed an image similar to $\boldsymbol{I}_t$ while sampling colors from $\boldsymbol{I}_{t+1}$. This operation is called inverse warp because even if the optimized optical flow map indicates the displacement of $\boldsymbol{I}_t$ pixels, the constructed image moves $\boldsymbol{I}_{t+1}$ pixels and not $\boldsymbol{I}_t$ pixels, and the applied optical flow $\boldsymbol{F}'$ from $\boldsymbol{I}_{t+1}$ to $\boldsymbol{I}_t$ is the inverse of $\boldsymbol{F}$.

$$\forall \boldsymbol{p} \in \Omega, \begin{cases} \boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{F}(\boldsymbol{p}) \\ \boldsymbol{p} = \boldsymbol{p}' + \boldsymbol{F}'(\boldsymbol{p}') \end{cases}$$

The difference with direct warp, where $\boldsymbol{I}_t$ pixels would have been moved so that constructed image would resemble $\boldsymbol{I}_{t+1}$ will be discussed on section4.5.

4.3 Retrieving a depth map using reprojection based optimization

Retrieving a depth map can be considered as a subproblem of our initial optical flow problem. Indeed, from an estimated depth $\widetilde{\theta}_t$ and pose estimation $\boldsymbol{T}_{t\to t+1} = (\mathbf{R}, \boldsymbol{t})$ (possibly known from other means, e.g. dedicated sensors), we can deduce an optical flow map. Using equation 3.4 from chapter 3, and assuming a rigid scene, we get for any point coordinate $\boldsymbol{p}$ in $\boldsymbol{I}_t$:

$$\boldsymbol{F}(\boldsymbol{p}) = \boldsymbol{p} - \Pi \left(\mathbf{KRK}^{-1} \Pi_{-1}(\boldsymbol{p}) + \frac{\mathbf{K}\boldsymbol{t}}{\widetilde{\theta}(\boldsymbol{p})} \right) \tag{4.2}$$

Practically, this subproblem consists in modeling the reprojection with less parameters: instead of optimizing the full optical flow, which is $|\Omega| \times 2$ independent elements, we simply optimize a depth map of $|\Omega|$ elements and a 6 degrees-of-freedom vector of displacement.

An important aspect of this problem to consider is the fact that estimated depth $\widetilde{\theta}_t$ is only a function of I_t while pose $\boldsymbol{T}_{t\to t+1}$ is function of both I_t and I_{t+1}. This is particularly interesting when considering another frame pair still involving I_t. For instance, when considering I_{t+2} with its associated pose relative to I_t, $\boldsymbol{T}_{t\to t+2}$, the new reprojection optimization is still modifying $\widetilde{\theta}_t$ along with $\boldsymbol{T}_{t\to t+1}$. Only 6 new parameters are added to the problem.

More generally, when considering one key frame I_t and a set of N reference frames I_k, one can retrieve depth with joint reprojection optimization, as it only requires $W \times H + N \times 6$

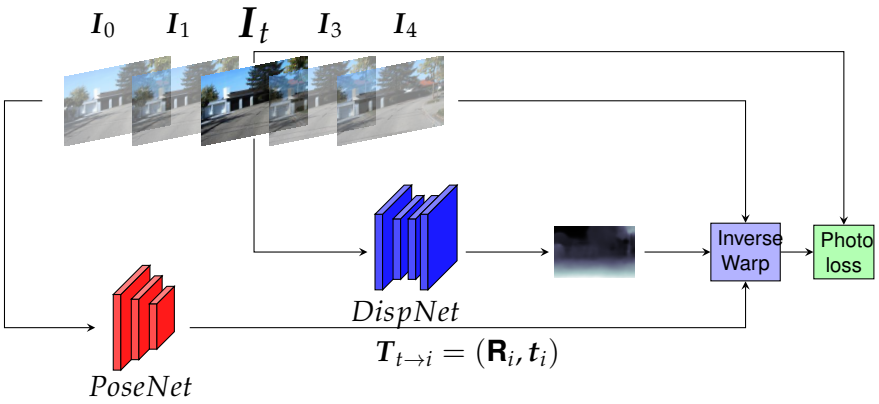


Figure 4.2: General workflow architecture of SFMLearner[Zho+17]. (simplified, without regularization)

parameters. This technique is the underlying concept behind dense SLAM techniques such as DTAM [NLD11].

4.3.1 Training a neural network to output depth with reprojection based optimization

The next logical step from this depth retrieval is then to train a neural network to output depth with this technique, also called auto-supervision. The supervision is not made on the target depth map anymore, since the network trains on a lot of different samples. Indeed instead of computing a loss relative to the difference of our network's output and an estimated depth map from reprojection based optimization, the photometric loss function will serve as a direct value to optimize for the network, which will eventually result in an indirect optimization of depth values from its output.

SFMLearner from Zhou et al., a quick presentation

To illustrate this training workflow, we partly introduce a very influential and yet simple work on auto supervision from Zhou et al. [Zho+17]. All other works of this field (including this one) are actually built on this fundamental basis.

As mentioned in the introduction of this chapter, this work relies on a neural network to produce a depth map (more specifically, the network called DispNet outputs an inverse depth map), but also on another network to produce a pose (called PoseNet). Both networks are simple CNNs. PoseNet is inspired from VGG[SZ14], while DispNet is inpired from FlowNet[Dos+15], just like our DepthNet. This work may not be the most recent, but will serve as a basis for comparison, as further improvement built on this work [Cas+19; Ran+18; YS18] mostly try to address the non-rigid scene problem which we chose not to focus on. The general workflow is presented figure 4.2. The depth is estimated at a target frame called I_t, located at the middle of a sequence. Other frames are called reference frames I_i with $i \in [\![0, N-1[\![$.

This work is very interesting for us, because the supervision need is very low, we don't even need to know the displacement which is trained to be estimated. Using this technique might help us to quickly get a first prototype for a real world DepthNet fine-tuning workflow with very few changes.

4.3.2 Conclusions of reprojection based optimization

Our review showed that it was possible to train a neural network to output depth only with a bilinear sampling operation which is differentiable, information about camera movement and geometric calibration. No depth groundtruth is involved in the training process.

We also have a way to construct a training algorithm that does not require camera movement to get significant results for both depth and pose estimation. Indeed, the work of Zhou et al. is very similar to what we want to achieve, and require a few changes for it to fit our needs regarding DepthNet :

- Replace DispNet with DepthNet
- feed two images to DepthNet, with rotation compensation, this is possible with PoseNet.
- normalize PoseNet translation estimation magnitude to get DepthNet nominal displacement

These changes will be discussed during the next chapter. Instead, we are going to study this work as thoroughly as possible with the hope that fixing potential problems in this context will improve the basis for a better unsupervised DepthNet training.

The next sections will discuss the context of training a network to output depth within a sequence of frames, with a known pose, and especially the two main independent limitations of the reprojection based optimization used alone, and how we might be able to overcome them:

- Regularizing a depth map to be physically plausible (section 4.4)
- Filtering invalid reprojection errors due to occlusions (section 4.5)

4.4 Regularization with smooth loss and diffusion

In addition to photometric loss $\mathcal{L}_p$, SFMLearner's algorithm also introduces a smooth loss. This loss tries to regularize the depth network's output by averaging Laplacian absolute values of its output.

$$\mathcal{L}_s = \frac{1}{|\Omega|} \sum_{\boldsymbol{p} \in \Omega} |\Delta DispNet(\boldsymbol{I}_t)(\boldsymbol{p})|$$

This regularization problem is actually very common in classic image processing operations such as denoising or optical flow estimation.

After studying smooth regularization in context of the more well known optical flow estimation problem, this section will focus on presenting a review on possible smooth losses to apply on a network output to improve the resulting depth and making it as physically plausible as possible.

4.4.1 Optical flow smoothing

We discussed the color consistency optimization in its simplest form that assumes that finding the global minimum of eq 4.1 is the same as finding optical flow. Obviously this assumptions does not hold in general because the mapping from position to color is highly non-injective, and multiple points can have similar colors. This is the case for example in a texture-less area, where the color is similar on a large area of points.

To solve this problem, several assumptions are made on typical realistic optical flow. For example, Horn and Schunck [HS81] extended the reprojection optimization with a smooth constraint, so that points close to each other have similar optical flow values. This assumption is related to the fact that when looking at moving objects, optical flow is continuous with respect to surface.

It can be noted that continuity on a discrete 2D map is not physically meaningful. In practice, reducing the discrete Laplacian ensures value proximity between nearby points. This is known as diffusion, inspired from the heat equation.

$$\partial_t \boldsymbol{F} = \Delta \boldsymbol{F} = \begin{bmatrix} \Delta \boldsymbol{F}_1 \\ \Delta \boldsymbol{F}_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial^2}{\partial u^2}\boldsymbol{F}_1 + \frac{\partial^2}{\partial v^2}\boldsymbol{F}_1 \\ \frac{\partial^2}{\partial u^2}\boldsymbol{F}_2 + \frac{\partial^2}{\partial v^2}\boldsymbol{F}_2 \end{bmatrix}$$

where Δ is the Laplacian operator, which is then approximated by updating its value using Euler's method and (u, v) are real values inside $[0, H] \times [0, W]$. In practice, the derivative is constructed from difference values within integer positions inside Ω, using convolution kernels, such as Sobel for first order derivate and discrete Laplacian for second order, in accordance with the scale space method [Lin96].

$$\Delta \boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * \boldsymbol{F}$$

$$\boldsymbol{F} \leftarrow \boldsymbol{F} + \lambda' \Delta \boldsymbol{F}$$

where λ' is the step value.

The new optimization workflow is then

$$\partial_t \boldsymbol{F} = -\nabla_{\boldsymbol{F}} \mathcal{L}_p + \lambda \Delta \boldsymbol{F} \tag{4.3}$$

$$\boldsymbol{F} \leftarrow \boldsymbol{F} - \gamma \nabla_{\boldsymbol{F}} \mathcal{L}_p + \lambda \gamma \Delta \boldsymbol{F}$$

where λ is the smoothing rate and γ the optimization step.

4.4.2 Applying smoothing on a depth map

Similarly to optical flow, we can try to regularize a depth map. The first idea might be to simply smooth the resulting optical flow from depth and motion.

However, we can take this regularization further by making several physical assumptions directly on the depth map and try to apply them as a diffusion operation. It is important to formulate them precisely to determine how a depth map can be regularized, and what constitutes a realistic depth. Instead of deciding how we want our depth maps to be like, we must first think about it the other way round: "Given a perfect depth map, what regularization would make it stable?"

- For naturally occurring scenes, depth discontinuity is possible only when image colors are also discontinuous. This is not necessarily true for the opposite, when for example the camera is looking at a textured plane. This is the case when looking at a road with markings for example.
- We suppose 3D points to describe a continuous differentiable surface. As a consequence, every surface point when zoomed enough from any point of view would look like a plane. A plane surface should be possible to converge to, even if it is not normal to the camera and goes to an infinite value, this case can be illustrated when looking at the horizon for example. In other words, the smoothing regularization on a depth map of a plane surface should be ineffective.

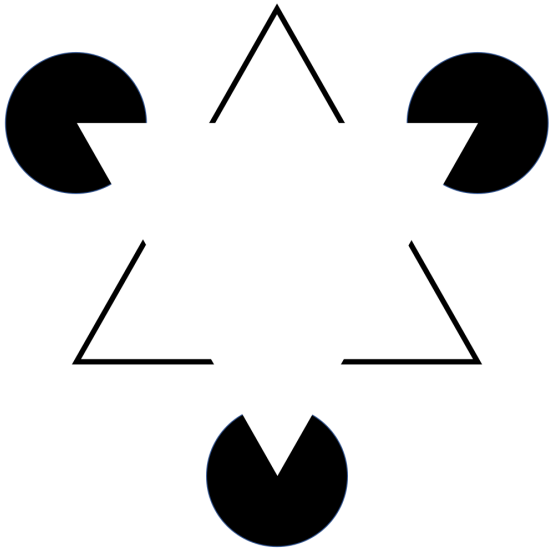


Figure 4.3: In this example called Kanizsa Triangle [Kan55], humans instinctively see a triangle above everything else and thus depth discontinuity in areas with no luminance gradient.

- We can also mention the problem with a completely uniform background. Since there is no texture, the photometric loss does not create any incentive to change its value there since there is no error. One of the goal of regularization will be to keep this background at the right value. In other word, any regularization happening on a textured surface cannot spread to a texture-less area if there is a clear boundary between the two surfaces.

It is worth mentioning that the first assumption for image discontinuity is not always true, as the human mind itself assumes surface (and thus depth) discontinuities in well known optical illusions called subjective contours [Cor72]. This not only indicates that our learning workflow is not the same as the natural one experienced by humans, but also that under very specific light conditions, this could happen, making our whole optimization fail. While being probably worth studying in both neurological and computer vision contexts in further works, we believe these examples to be marginal in nature and choose to ignore them here.

In this section, we are interested in regularizing our optimization problem by smoothing a 2D map, the same way optical flow needed to be smoothed in Horn and Schunck [HS81]. The first thing we need to do is to transpose the physical assumption of our surfaces into the depth map.

Regardless of the method used for smoothing, we consider a 2D map to be smooth at some point when its Laplacian is zero, because it's locally planar. Consequently, this is also a map stable to diffusion.

4.4.3 Translating physical assumptions into equations

For our optimization to be physically accurate, We must find a map where we would actually want its Laplacian to be zero. For example, as shown figure 4.4, when looking at the horizon, depth gradient $\frac{\partial \theta}{\partial y}$ is not of constant gradient. However, inverse depth ξ is piecewise affine.

Proposition 2. *The inverse depth map $\xi = \frac{1}{\theta}$ of a perfect plane has a null Laplacian and is then stable with respect to smoothing.*

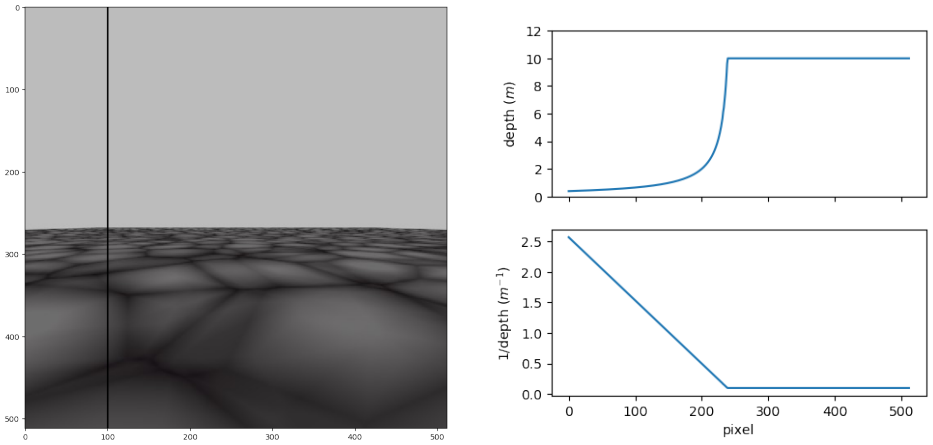


Figure 4.4: Example of a scene with an horizontal plane. Right is the depth of the vertical line depicted on the rendered image.

See appendix C for proof. From now on, our study will be focused on optimizing depth map, with a smooth constraint on inverse depth, called ξ.

4.4.4 Smooth loss presentation

In an attempt to generalize this "smooth constraint", this section will study loss based smoothing, as it is widely used e.g. in image denoising. It is an optimization problem where we design a smooth loss $\mathcal{L}_s$ and add it to the other loss we try to minimize.

For a particular 2D map ξ defined on a surface Ω, we will study two smooth loss forms, trying to minimize gradients:

$$\mathcal{L}_{s\nabla} = \frac{1}{2} \iint_{\Omega} c\left(\|\nabla\xi\|^2\right) dS \tag{4.4}$$

or trying to minimize Laplacians:

$$\mathcal{L}_{s\Delta} = \frac{1}{2} \iint_{\Omega} c\left((\Delta\xi)^2\right) dS \tag{4.5}$$

These equations can be used to retrieve well known denoising loss functions.

Total Variation, also called TV Loss [ROF92] is a special case of equation 4.4 where $c = x \mapsto 2\sqrt{x}$.

$$\mathcal{L}_{TV} = \iint_{\Omega} \|\nabla\xi\| dS$$

Besides, we will link the general expression of these losses with diffusion by studying their gradient.

Definition 4. *For a particular loss function $\mathcal{L}$, if there is a scalar or vector field T such that*

$$\nabla_T \mathcal{L} = -\Delta T$$

the loss wi ll be called diffusion loss with respect to T, because if T was the training parameter for the gradient descent optimization, we would get the diffusion operation

$$\partial_t T = -\nabla_T \mathcal{L} = \Delta T$$

This is the case when considering $\mathcal{L}_{s\nabla}$ and $c = x \mapsto x$, as shown by Perona and Malik [PM90], because we have at the interior of Ω

$$\nabla_\xi \mathcal{L}_{s\nabla} = -\Delta\xi$$

(see appendix D.1)

As such, we now identify the loss $\mathcal{L}_{s\nabla}$ with $c = x \mapsto x$ as the diffusion loss.

$$\mathcal{L}_{diff} = \frac{1}{2} \iint_\Omega \|\nabla\xi\|^2 dS \tag{4.6}$$

4.4.5 Diffusing the output of a neural network

One must be careful when talking about diffusion in the context of a function output, here a neural network. In the case of a regular diffusion such as with Horn and Shunck [HS81], the trainable parameters are the map we want to smooth itself.

As such, if we apply a smooth loss $\mathcal{L}_s$ on ξ and ignore the other losses, the optimization step is as simple as:

$$\begin{aligned} \forall \boldsymbol{p} \in \Omega, \partial_t \xi(\boldsymbol{p}) &= -\nabla_{\xi(\boldsymbol{p})} \mathcal{L}_s \\ \xi(\boldsymbol{p}) &\leftarrow \xi(\boldsymbol{p}) - \gamma \nabla_{\xi(\boldsymbol{p})} \mathcal{L}_s \end{aligned} \tag{4.7}$$

finding the value $\nabla_{\xi(\boldsymbol{p})} \mathcal{L}_s$ will completely characterize the smoothing process on ξ.

In the case of an output of a function (here, a neural network), the general optimization step becomes for a vector of trainable parameters $\boldsymbol{W}$ (for example, the weights of a neural network) and a function f such that $f(\boldsymbol{W}) = \xi$:

$$\begin{aligned} \partial_t \boldsymbol{W} &= -\nabla_{\boldsymbol{W}} \mathcal{L}_s \\ &= -\mathbf{J}_f(\boldsymbol{W})^\top \nabla_\xi \mathcal{L}_s \end{aligned}$$

Where $\mathbf{J}_f(\boldsymbol{W})$ is the Jacobian of f evaluated in $\boldsymbol{W}$ and $\mathbf{M} \mapsto \mathbf{M}^\top$ the transposition operation. It should be noted that for $\mathbf{J}_f(\boldsymbol{W})$ to be a matrix, we must consider ξ as a vector of values, $\xi \in \mathbb{R}^{|\Omega|}$ (the vector dimension is the number of pixels).

This equation can be also written like this:

$$\forall i, \partial_t \boldsymbol{W}_i = \sum_{\boldsymbol{p} \in \Omega} \frac{\partial f}{\partial \boldsymbol{W}_i}(\boldsymbol{p}) \times \frac{\partial \mathcal{L}_s(\xi)}{\partial \xi(\boldsymbol{p})}$$

As a consequence, we can have the evolution of ξ

$$\partial_t \xi = \mathbf{J}_f(\boldsymbol{W}) \partial_t \boldsymbol{W} = -\mathbf{J}_f(\boldsymbol{W}) \mathbf{J}_f(\boldsymbol{W})^\top \nabla_\xi \mathcal{L}_s \tag{4.8}$$

It appears that when f is identity (and thus $\boldsymbol{W} = \xi$), $\mathbf{J}_f$ is also identity and when $\mathcal{L}_s$ is the diffusion loss with respect to ξ, can retrieve the behaviour presented in definition 4. In practice, the Jacobian of a neural network is very computationally expensive to get, so there is no guarantee about $\xi = f(\boldsymbol{W})$ evolution. The only thing we know is given a step little enough, $\mathcal{L}_s$ will be lowered, but we don't even know the maximum size of that step.

Of course, this study generalizes to any kind of loss whose gradient with respect to a function output can be computed. Although widely studied on the regular context (when Jacobian is identity matrix), loss functions may have robustness problems for an unknown Jacobian. Appendix D.2 shows in a simple example how a too large optimization step

size might make a 2D map diverge, and in appendix D.3 how a simple function such as inverse will make the regularization step size unknown. This is particularly interesting for our use-case, since DepthNet is supposed to output depth directly, so we will end up trying to smooth its inverse. Note that the ELU function[CUH15] we used (see 3.5.3) might help mitigating gradients magnitude of $\nabla\xi$ when estimated depth $\widetilde{\theta}$ is close to 0 (and thus ξ is high).

Adding robustness to smooth loss

From now on, for a particular smooth loss function $\mathcal{L}_s$ with respect to a scalar or vector map T, its gradient $\nabla_T\mathcal{L}_s$ will be discussed but in the general case, additional steps might be added. Namely, we can decompose the optimization process into three separate steps:

1. consider the $T = f(\boldsymbol{W})$ map we would like to smooth that is the output of a neural network, like e.g. ξ or $\nabla\xi$, as a set of trainable parameters.

2. perform a smoothing operation (such as diffusion) directly on it.

3. design the loss function as a distance between the smoothed map T_{smooth} and T we actually want to smooth with the parameters $\boldsymbol{W}$. $\mathcal{L}'_s = d(T = f(\boldsymbol{W}), T_{smooth})$.

This method might use any kind of smoothing method, as long as it is applied directly on T.

Two step joint optimization

An interesting idea to investigate in the future would be to further apply this idea of decoupling smoothness and photometric optimizations. This idea first introduced by Steinbrucker [SPC09] is also applied within DTAM algorithm [NLD11], and aims at finding the solution of equation 4.9, for a fixed estimated inverse depth $\xi = \frac{1}{\widetilde{\theta}}$. d is a distance function, e.g. L2 in the case of DTAM, and γ is an arbitrary positive hyper-parameter. Following e.g. DTAM method using Legendre-Fenchel transform and primal-dual optimization [CP11], a global minimum could be found in real time (with a 2011 high end customer hardware), and it can potentially be reused here, alternatively with optimization of the same equation 4.9 but with fixed α.

$$\mathcal{L}(\widetilde{\theta}, \alpha) = \mathcal{L}_p(\widetilde{\theta}) + \lambda\mathcal{L}_s(\alpha) + \gamma d\left(\alpha, \xi = \frac{1}{\widetilde{\theta}}\right) \tag{4.9}$$

The global optimization can then become:

1. estimate $\widetilde{\theta} = f(\boldsymbol{W})$ from trainable weights $\boldsymbol{W}$.

2. optimize $\mathcal{L}(\widetilde{\theta}, \alpha)$ with $\widetilde{\theta}$ fixed, find the global solution $\widetilde{\alpha} = \arg\min_{\alpha} \mathcal{L}(\widetilde{\theta}, \alpha)$ with an off-the-shelf algorithm.

3. compute $\nabla_{\boldsymbol{W}}\mathcal{L}(\widetilde{\theta}, \widetilde{\alpha})$ and update $\boldsymbol{W}$.

4. update the γ value if needed, ideally it would lower throughout the training, enforcing a stricter similarity between α and ξ.

4.4.6 Minimizing gradient with $\mathcal{L}_{s\nabla}$

As it is used by most published works, we can briefly analyze variations of the smooth loss function mentioned eq 4.4

$$\mathcal{L}_{s\nabla} = \frac{1}{2} \iint_{\Omega} c\left(\|\nabla\xi\|^2\right) dS$$

As said above, but in a more general way, we know that simple gradient descent applied on ξ of this cost function results in the following inside the boundaries.

$$\partial_t \xi = -\nabla_\xi \mathcal{L}_{s\nabla} = \text{div}\left(c'(\|\nabla\xi\|^2)\nabla\xi\right) \tag{4.10}$$

For example, the total variation loss (called TV loss) [ROF92] mentioned above is a widely used constraint in partial derivative equations (PDE) based inpainting algorithms [SC02], and uses the same idea, but with gradient norm.

$$\mathcal{L}_{TV} = \iint_{\Omega} \|\nabla\xi\| dS$$

From identifying $c = x \mapsto 2\sqrt{x}$, we can deduce

$$\nabla_\xi \mathcal{L}_{TV} = -2\text{div}\left(\frac{\nabla\xi}{\|\nabla\xi\|}\right)$$

If instead, we have $c = x \mapsto x$, we end up with this already discussed equation:

$$\nabla_\xi \mathcal{L}_{diff} = -\Delta\xi \tag{4.11}$$

We then retrieve the gradient that would result in a diffusion if ξ was the trainable parameter.

Global solution of diffusion loss $\mathcal{L}_{diff}$

Although not the global solution, from equation 4.10 it seems that a constant slope (and thus a null Laplacian) would be a stable position for this optimization, regardless of c function. This is not the case because the integration is made on a finite map, and the exact term on the boundary of Ω is:

$$\partial_t \xi = -\nabla_\xi \mathcal{L}_{diff} = \text{div}\left(c'(\|\nabla\xi\|^2)\nabla\xi\right) - c'(\|\xi\|^2)(\nabla\xi.\boldsymbol{n}) = \Delta\xi - \nabla\xi.\boldsymbol{n}$$

where $\boldsymbol{n}$ is the unit vector normal to the boundary. If we consider the 1D example and a map where $\Delta\xi = \frac{d^2\xi}{dx^2} = 0$ everywhere, we get:

$$\partial_t \xi(0) = -\frac{d\xi}{dx}(0)$$

Following the convention of computing gradient in a picture boundary with values inside the interior of that set ($\frac{d\xi}{dx}(0) = \lim_{x\to 0+} \frac{d\xi}{dx}(x)$), we can see that increasing $\xi(0)$ while keeping $\partial_t\xi = 0$ inside of Ω (since $\Delta(\xi)$ is null everywhere) will then decrease the gradient $\frac{d\xi}{dx}(0)$ and vice-versa. $\partial_t \frac{d\xi}{dx}(0)$ is then of the same sign as $-\frac{d\xi}{dx}$. In other words, $\left|\frac{d\xi}{dx}\right|$ will be lowered, and gradient will converge to 0. The same applies to $\partial\Omega$, and can be extended to the 2D case:

$$\forall \boldsymbol{p} \in \partial\Omega, \xi(\boldsymbol{p}).\boldsymbol{n} \xrightarrow[t+\infty]{} 0$$

With propagation of this gradient, we can see how this will make the whole map converge to a constant plane.

Although not developed here, an interesting solution to this problem would be to not apply the optimization on $\delta\Omega$. That way, the points on boundaries would be unaffected by the optimization making the rest of the image stable with a non-null gradient. This is potentially hazardous as points in boundaries often are noisy because it lacks neighbouring pixels informations for the main photometric loss to be reliable, making smooth regularization essential for them. In our 1D use-case, if we use this method, the only solution for the smooth loss would be a linear interpolation between the two boundary values, regardless of starting values in the interior of Ω

4.4.7 Minimizing Laplacian with $\mathcal{L}_{s\Delta}$

In the case we want a null Laplacian but not necessarily a constant map, some recent works proposed to minimize the sum of all absolute values of Laplacians of a 2D map [Zho+17; WB18]. For ease of notation, we will call this loss TVV for total variation of variation, in reference of total variation Loss [ROF92]. This equation can be retrieved from equation 4.5, with $c = x \mapsto 2\sqrt{x}$

$$\mathcal{L}_{TVV} = \iint_{\Omega} |\Delta\xi| dS$$

However, every Laplacian value is dependent on the adjacent values. More generally, from Ostrogradski 2D theorem, we can state that

$$\iint_{\Omega} \text{div}(\boldsymbol{A}) dS = \oint_{\partial\Omega} \boldsymbol{A}.\boldsymbol{n} dl$$

where $\boldsymbol{A}$ is any 2D function, Ω is a closed surface, and $\partial\Omega$ its boundaries, and $\boldsymbol{n}$ is the normal vector to the boundary tangent. By identifying $\boldsymbol{A} = \nabla\xi$, we get

$$\forall\Omega' \subseteq \Omega, \iint_{\Omega'} \Delta\xi dS = \oint_{\partial\Omega'} \nabla\xi.\boldsymbol{n} dl$$

Here, since we would use the absolute values, the surfaces Ω' to consider are the ones where Laplacian values keep the same sign.

With TVV loss, the only points in the inverse depth map ξ where an optimization is actually done are the array bounds and the inflexion points, where Laplacian changes of sign.

This optimization, even though it has empirically proved to converge to blurry version of the initial image after several optimization steps is far from regular.

Let's instead look at the general smooth loss $\mathcal{L}_{s\Delta}$ that tries to lower Laplacians.

$$\mathcal{L}_{s\Delta} = \frac{1}{2} \iint_{\Omega} c\left((\Delta\xi)^2\right) dS$$

We can show that the resulting ξ-wise gradient loss is

$$\nabla_{\xi}\mathcal{L}_{s\Delta} = -\Delta(c'((\Delta\xi)^2)\Delta\xi) \tag{4.12}$$

(see appendix D.4.1 for a demonstration)

we can retrieve our first example of TVV Loss when $c = x \mapsto 2\sqrt{x}$. We then get

$$\nabla_{\xi}\mathcal{L}_{TVV} = 2\Delta\left(\frac{\Delta\xi}{|\Delta\xi|}\right)$$

since $\frac{\Delta\xi}{|\Delta\xi|}$ is either 1 or -1, its differentiation is always 0 or infinite. In practice, since discrete differentiation is approximated with finite differences, we actually get a finite value when $\Delta\xi$ changes of sign.

$$\forall \boldsymbol{p} \in \Omega, \nabla_\xi \mathcal{L}_{TVV} = \begin{cases} 1 & \text{if } \Delta\xi < 0 \text{ and changes of sign} \\ -1 & \text{if } \Delta\xi > 0 \text{ and changes of sign} \\ 0 & \text{otherwise} \end{cases}$$

When $c = x \mapsto x$, we have:

$$\nabla_\xi \mathcal{L}_s = \Delta^2 \xi \tag{4.13}$$

This equation is similar to the equation that qualifies a diffusion loss (definition 4) but is not of the form $\nabla_T \mathcal{L} = -\Delta T$

However, knowing that ξ is in fact a function and not the trainable parameters, rather than studying Laplacian minimization of it, we can see this loss as minimizing the divergence of its gradient $\nabla\xi$ and get insight from the gradient of loss with respect to $\nabla\xi$, $\nabla_{\boldsymbol{A}} \mathcal{L}_{s\Delta}$ with $\boldsymbol{A} = \nabla\xi$.

4.4.8 Diffusion of a vector map with divergence minimization

Let us consider the 2D vector map $\boldsymbol{A}$. We want to minimize its divergence $\text{div}(\boldsymbol{A})$. The motivation for this is that when $\boldsymbol{A} = \nabla\xi$, we get the equivalence $\text{div}(\boldsymbol{A}) = \Delta\xi$.

We can then modify equation 4.10 to work with divergence of a vector instead of gradient of a scalar:

Let us consider this modified loss function, from 4.5, by replacing $\Delta\xi$ with $\text{div}\boldsymbol{A}$:

$$\mathcal{L}_{s\Delta} = \iint_\Omega c\left(\text{div}(\boldsymbol{A})^2\right) dS$$

From this equation, assuming the trainable parameters are components of $\boldsymbol{A}$, we get in $\overline{\Omega}$ (the interior of Ω):

$$\nabla_{\boldsymbol{A}} \mathcal{L}_{s\Delta} = -\nabla\left(c'\left(\text{div}(\boldsymbol{A})^2\right)\text{div}(\boldsymbol{A})\right) \tag{4.14}$$

(see appendixD.4.2 for a demonstration)

we can retrieve our first example of TVV Loss when $c = x \mapsto 2\sqrt{x}$. We then get

$$\nabla_{\boldsymbol{A}} \mathcal{L}_{s\Delta} = 2\nabla\left(\frac{\text{div}(\boldsymbol{A})}{|\text{div}(\boldsymbol{A})|}\right)$$

When $c = x \mapsto x$, we have

$$\nabla_{\boldsymbol{A}} \mathcal{L}_{s\Delta} = -\nabla\left(\text{div}\boldsymbol{A}\right) \tag{4.15}$$

And especially, when at first $\boldsymbol{A}$ is of the form $\nabla\xi$, we get the following equation, no matter how $\boldsymbol{A}$ is derived from, even when it is diffused independently of ξ, for example in the fashion of two steps optimization (section 4.4.5). See appendix D.4.3 for a demonstration.

$$\nabla_{\boldsymbol{A}} \mathcal{L}_{s\Delta} = -\Delta\boldsymbol{A} \tag{4.16}$$

The minimization of square values of Laplacian, as described by equation 4.5 with $c = x \mapsto x$ can then be called gradient diffusion loss $\mathcal{L}_{gdiff}$. Because we have

$$\mathcal{L}_{gdiff} = \iint_{\Omega} (\Delta\xi)^2 dS$$

$$\nabla_{\nabla\xi}\mathcal{L}_{gdiff} = -\Delta(\nabla\xi)$$

Figure 4.5 presents a comparison for the four mentioned techniques in a 1D example: TV, diffusion, TVV and gradient diffusion. We test a smooth loss optimization with a sinusoid and and abs function with added noise. Stochastic gradient descent (SGD) is used, with indicated learning rates and momentums in each graph title. As shown and discussed section 4.4.6, TV loss and Diffusion Loss seem to converge to a constant slope, but also a constant value. On the other hand, the first TVV appears to smooth the function enough to denoise it, but fails to converge to a simple affine function. Finally, gradient diffusion loss converges to an affine function, but needs ten times more iterations to achieve a reasonable result, with a higher momentum of 9.99. This function which makes intervene the fourth derivative of ξ (see equation 4.13) is one of the reasons the convergence is so slow.

4.4.9 Edge-aware smooth loss

When trying to smooth a 2D map relative to a particular image, for example optical flow $\boldsymbol{F}$, depth $\widetilde{\theta}$ or inverse depth map ξ of image $\boldsymbol{I}$, although useful for texture-less area, smoothing everything equally might be not ideal when considering discontinuities (and then the Laplacian is theoretically infinite). For example occlusion areas in a depth map. The map is then "over smoothed" [Gar+16].

As tested by many previous works [NE86; WB18; MWA18; Ran+18; YS18], a simple way to overcome the smearing of edges in our map, is to temper the smoothing when the image has edges, i.e. when it has gradients with high norm.

To establish an efficient strategy, we can look at the image denoising techniques, and where we can use it differently for inverse depth map texture-aware smoothing.

Taking back our smooth losses equations 4.4 and 4.5, we can replace c function with more complicated ones, especially function with decreasing derivate. As such, smoothing will be less enforced for high gradients. For example Perona and Malik [PM90] proposed the following derivate function $g = c'$ (among others) for the image I:

$$g : \|\nabla I\|^2 \mapsto exp\left(-\left(\frac{\|\nabla I\|}{\kappa}\right)^2\right) \tag{4.17}$$

This technique is called anisotropic diffusion, contrary to the regular diffusion (also called isotropic), which has the same diffusion strengh everywhere.

Besides, in their work, they presented a mathematical analysis of the function g.

- They show that with any positive function g, the smoothed image never diverges outside original image extrema (given a small enough optimization step), this is the maximum principle.
- They also show that if g is differentiable, given the sign of Φ' where $\Phi = x \mapsto xg(x)$, the edges will be smoothed (if $\Phi' > 0$), or enhanced (if $\Phi' < 0$). The image diffused that way converges to a piecewise constant map where gradient is either null or infinite.

From equation 4.17 we can compute the function Φ and its differentiation Φ'.

$$\forall x \in \mathbb{R}^+ \Phi'(x) = g(x) + xg'(x) = g(x)\left(1 - 2\left(\frac{x}{\kappa}\right)^2\right)$$

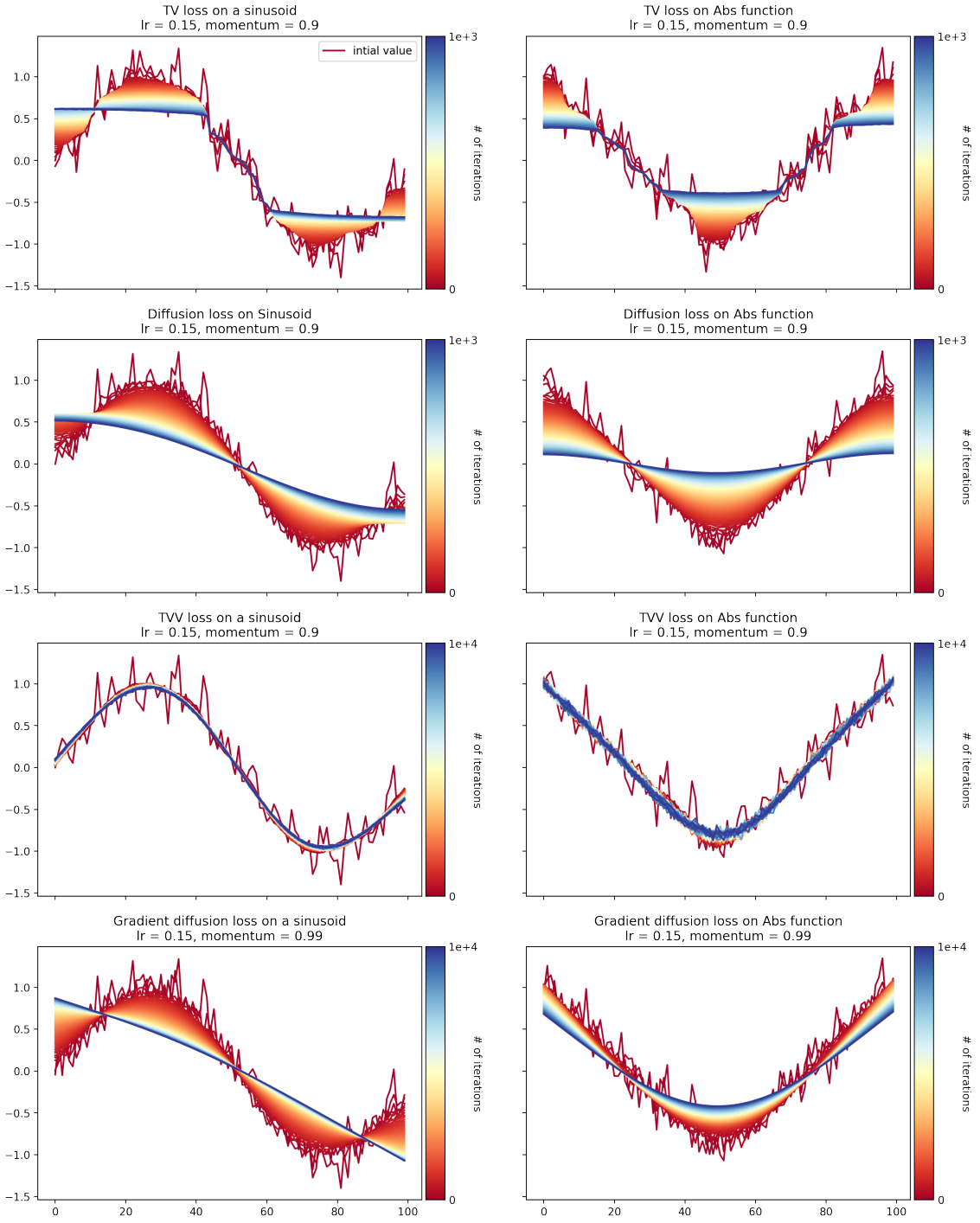


Figure 4.5: comparison of four smooth losses TV, TVV, diffusion, and gradient diffusion. Gaussian noise with $\sigma = 0.2$ has been added to input: a sinusoid and an abs function. SGD is used, with indicated learning rates and momentums in each graph title. Color indicates the number of iterations

Figure 4.6: Perona-Malik anisotropic diffusion of a picture of a flower, $\kappa = 0.3$

In this particular function g, edges with a gradient of norm above $\frac{\kappa}{\sqrt{2}}$ get enhanced, and mitigated otherwise. Figure 4.6 shows the picture of a flower (with values in $[0,1]$) that got diffused with a κ of 0.3. We can see that the picture tends to converges to a 3 colors image.

Our case is different here, since we don't have only the map ξ to work with: we also have the image $\boldsymbol{I}$ from which ξ is extracted. We don't want to preserve the edges of ξ (or $\nabla\xi$), but we want ξ edges (or $\nabla\xi$ edges) to match image $\boldsymbol{I}$ edges. As discussed section 4.4.2, we want depth discontinuities to coincide with image discontinuities.

Nevertheless, we can be inspired by Perona and Malik proposition to adapt our loss functions so that $c' = x \mapsto h(\nabla I)x$, where h is a decreasing function. For example:

$$\nabla_\xi \mathcal{L}_{diff}(\xi) = \text{div}\left(g\left(\|\nabla \boldsymbol{I}\|^2\right)\nabla\xi\right) = \text{div}\left(exp\left(-\frac{\|\nabla \boldsymbol{I}\|^2}{\kappa^2}\right)\nabla\xi\right)$$

Similarly, with diffusion of $\nabla\xi$, we would get

$$\nabla_{\nabla\xi} \mathcal{L}_{gdiff}(\nabla\xi) = \nabla\left(g\left(\|\nabla \boldsymbol{I}\|^2\right)\nabla.\boldsymbol{A}\right) = \nabla\left(exp\left(-\frac{\|\nabla \boldsymbol{I}\|^2}{\kappa^2}\right)\Delta\xi\right)$$

Here, we mentioned $\nabla\boldsymbol{I}$ which is in fact three different vectors (since image is composed of three channels). To compute the norm, we implicitly take the mean of the norms of the three gradient vectors.

One should note that if the maximum principle can still be respected for ξ diffusion, when diffusing gradient, this is only the case with respect to $\nabla\xi$: the slope is bounded from the initial depth map, but not the actual values. It can have some problems, for example when the diffusion makes ξ go under 0 which is not physically possible. So it may be necessary to add regularization to prevent that. The simplest way is to manually bound the output of the network so that $\xi = f(\boldsymbol{W}) > \epsilon$ with ϵ a fixed strictly positive value.

The equivalent losses $\mathcal{L}_{diff}$ and $\mathcal{L}_{gdiff}$ to these equations can then be computed by simply integrating these g functions. It is actually fairly easy since ∇I is not a function of ξ, so $g(\|\nabla I\|^2)$ is constant with respect to ξ.

4.4.10 Final edge-aware forms of considered smooth losses

For loss functions $\mathcal{L}_{s\nabla}$ (equation 4.4) and $\mathcal{L}_{s\Delta}$ (equation 4.5, we can study a particular form of c which includes influence of $\nabla\boldsymbol{I}$. TV and TVV losses are modified to match literature, as it is used with DTAM [NLD11] and Godard et al. [GMB17].

$$\mathcal{L}_{diff}(\xi) = \iint_{\Omega} exp\left(-\left(\frac{\|\nabla \boldsymbol{I}\|}{\kappa}\right)^2\right) \|\nabla\xi\|^2 dS \tag{4.18}$$

$$\mathcal{L}_{TV}(\xi) = \iint_{\Omega} exp\left(-\frac{\|\nabla \boldsymbol{I}\|}{\kappa}\right) \|\nabla\xi\| dS \tag{4.19}$$

$$\mathcal{L}_{gdiff}(\nabla\xi) = \iint_{\Omega} exp\left(-\left(\frac{\|\nabla \boldsymbol{I}\|}{\kappa}\right)^2\right) (\Delta\xi)^2 dS \tag{4.20}$$

$$\mathcal{L}_{TVV}(\nabla\xi) = \iint_{\Omega} exp\left(-\frac{\|\nabla \boldsymbol{I}\|}{\kappa}\right) |\Delta\xi| dS \tag{4.21}$$

It's easy to see how a high κ value makes it equivalent to our first simple loss functions definitions.

Figure 4.7 shows a comparison of anisotropic diffusion and anisotropic gradient diffusion on a set of 1D examples with a simple toy image (dashed blue curve) whose gradient is 0 everywhere except in pixel coordinates 20 and 80 where it is above the κ value. These results indicate that as we foresaw, diffusing $\nabla\xi$ is more stable relative to image edges: last example shows a typical texture plane surface, and regular diffusion creates an edge from nowhere to make the depth piecewise constant. We can also see that our first example shows a typical transgression of the maximum principle: in order to be piecewise linear, ξ is rised above the sinusoid maximum. However, in the same time, it appears that the portion on which this happens seems to be a good linear regression of our function between the two discontinuities. Lastly, it can be noted that regular diffusion converges much faster than gradient diffusion.

As a conclusion for this section, we have a set of different losses that we can apply to a specific 2D map that might be the output of a network. In addition to the actual function c and the weight we can apply to this loss, we can also change a κ parameters to change behavior on images areas with texture (i.e. with a high gradient $\nabla \boldsymbol{I}$). We also have a robust counterpart of these losses designed to avoid divergence, which might be indicated for particularly non-linear neural network function. The inverse input of a network being one of them.

4.5 Dealing with occlusions

Occlusions occur when an object moves behind an other. From one frame to another, the object is then not visible anymore. This section studies how this phenomenon compromises the reprojection principle which tries to retrieve pixels from one frame to another, and what filtering can be applied to be perform a robust training.

4.5.1 Inverse warp based optimization *vs* occlusions

Taking back the first inverse warp photometric optimization problem, another aspect to take into account is the occluded areas. As seen figure 4.8, some pixels can be occluded from one frame to another. In this example, the torus should not be retrieved in the first image but can be found in the last image. Similarly, the cube should not be retrieved in the last frame, but can be found in the first frame. As a consequence, the task of retrieving them in both frames is impossible and the optimization will eventually converge to a false solution.

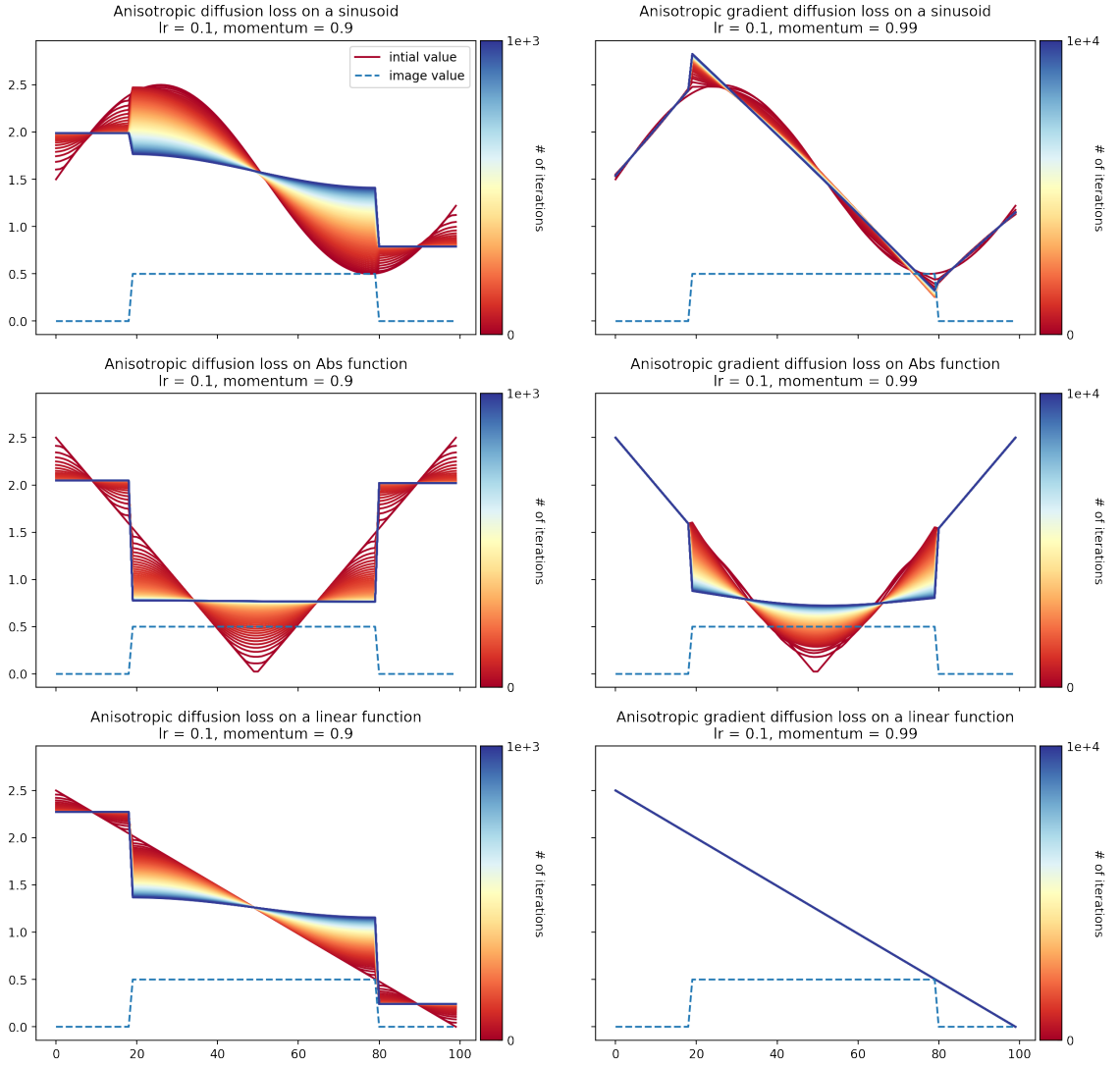


Figure 4.7: 1D Anisotropic diffusion and gradient diffusion of several functions. Dashed line indicates the image function with null gradient, except on $x = 20$ and $x = 80$. Plain line colors indicate number of iterations

$\boldsymbol{I}_{t-1}$ $\boldsymbol{I}_t$ $\boldsymbol{I}_{t+1}$

Figure 4.8: Example of occluded elements in a synthetic scene when trying to estimate depth of $\boldsymbol{I}_t$.

Let's consider our problem with $\boldsymbol{I_t}$, $\boldsymbol{I_{t_2}}$ and $T_{t\to t_2}$, we try to reconstruct $\widetilde{\theta}_t$ indirectly by reconstructing $\boldsymbol{I_t}$ with mentioned parameters and camera intrinsics $\mathbf{K}$. If we consider the function IW (for inverse warp) that performs this image reprojection, we can define it as:

$$\begin{array}{rlll} IW: & \mathbb{R}^{|\Omega|}\times\mathbb{R}^{3\times|\Omega|}\times\mathbb{R}^6\times\mathbb{R}^{3\times3} & \to & \mathbb{R}^{3\times|\Omega|} \\ & \widetilde{\theta}_t, \boldsymbol{I}_{t_2}, T_{t\to t_2}, \mathbf{K} & \mapsto & \widetilde{\boldsymbol{I}}_{t_2\to t} \end{array}$$

The photometric loss is then:

$$\mathcal{L}_p = \sum_{\boldsymbol{p}\in\Omega} d(\boldsymbol{I}_t, IW(\widetilde{\theta}_t, \boldsymbol{I}_{t_2}, T_{t\to t_2}, \mathbf{K}))(p) = \sum_{\boldsymbol{p}\in\Omega} d(\boldsymbol{I}_t, \widetilde{\boldsymbol{I}}_{t_2\to t})(\boldsymbol{p})$$

For example, if $\boldsymbol{I}_{t+1}$ is the image $\boldsymbol{I}_{t_2}$ in figure 4.8, the optimization process will try to reconstruct the cube in image $\boldsymbol{I}_t$ with pixels from $\boldsymbol{I}_{t_2}$ although it does not appear in the image.

Another way to view this issue is to see that a perfect depth map would produce an imperfect reconstruction, resulting in a non-zero loss. The optimization will then modify the depth map to better satisfy the photometric loss. Figure 4.12 shows an example of imperfect inverse warp caused by a perfect depth map, with a "ghost" of foreground on the occluded background.

4.5.2 Filtering occlusion areas from optimization

One solution for solving this problem is to try to detect the occlusion and then filter them from the photometric error to get $\mathcal{L}_{pf}$ (for photometric filtered).

$$\mathcal{L}_{pf}(t_2) = \sum_{\boldsymbol{p}\in\Omega} \begin{cases} d(\boldsymbol{I}_t, \widetilde{\boldsymbol{I}}_{t_2\to t})(\boldsymbol{p}) & \text{if } \boldsymbol{p} \text{ not occluded in } I_{t_2} \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

This technique, although not guaranteed to converge to the ground truth, at least makes the actual solution stable with the optimization. The smooth loss regularization is supposed to help in these areas where photometric loss is undefined.

As said section 4.3, the depth retrieval problem can benefit from multiple pairs of frames at the same time (one of them being invariably the frame of which we want to estimate the depth). As such, the occluded areas, relative to a particular frame pair won't be the same if the relative movement is different. Consequently, a simple solution to the occlusion problem in two frames stated before is here to retrieve the pixel in multiple frames, before and after, as illustrated figure 4.8. Although not guaranteed, the pixel has greater chances to be found in another frame, especially when the movement is consistent throughout the frame sequence: relative to I_t, occlusion areas for the pairs $I_t \to I_{t+n}$ and $I_t \to I_{t-n}$ should be completely different since the associated relative displacement is deemed almost opposite. Assuming occluded areas could be identified between two frames, the depth map $\widetilde{\theta}_t$ can be estimated using redundancy in multiple optical flow maps.

$$\mathcal{L}_{pf} = \sum_{t_k} \mathcal{L}_{pf}(t_k) \tag{4.23}$$

The next sections will discuss the different methods for detecting the occlusion areas.

4.5.3 Self filtered depth map inference

Several works [Jan+18; Zho+17] proposed a dedicated network to figure out the occluded scenes, or more generally, the areas on which the photometric error won't be low even

if the optical flow map is perfect, it can be interpreted as a confidence network for photometric loss. The main idea is to multiply the photometric loss by a confidence value $\widehat{E}(\boldsymbol{I}_t, \boldsymbol{I}_{t_2})$ outputted by a network. During the training, the network is then encouraged to have low values where photometric error is high. To avoid the degenerate solution with 0 confidence everywhere that would result in a 0 photometric loss, the network is also encouraged to output high confidence values with e.g. a binary cross entropy H_1 with ground-truth set to 1.

$$\mathcal{L} = \widehat{E}\mathcal{L}_p + \alpha H_1(\widehat{E})$$

The confidence network tries to find a compromise between these two antagonistic losses, and is deemed to find it where photometric error are not easily lowered, be it from occlusion, moving object or graphical artifacts such as non Lambertian surfaces or change of exposition.

This is a promising technique, somewhat related to Bayesian inference, where the network also predicts uncertainty, but it adds lots of new hyperparameters to the training workflow, especially the architecture of this new network and the weight to put to α.

4.5.4 Backward/forward consistency

Instead of asking a neural network to decide what constitutes an occluded area, we can solve this problem analytically. In this case, we extract the occlusion areas not from the image input like with self filtering, but rather from the pose and estimated depth. The occlusion areas will then be tied to the depth estimation, possibly erroneous but it can be geometrically deduced and not estimated. To better understand how we can analytically find these occlusion areas, we can look at the more general case of optical flow.

Optical flow consistency

When considering optical flow optimization, one solution for this occluded area problem is the forward/backward consistency check. This technique not only tries to compute forward flow $\boldsymbol{F}^+$ from I_t to I_{t_2}, but also the backward flow $\boldsymbol{F}^-$ from I_{t_2} to I_t. The general idea behind this redundancy is that these optical flow maps are expected to be consistent *up to the occluded areas*. Areas where optical flow is not consistent within forward/backward is excluded from the optimization.

$$\forall \boldsymbol{p} \in \Omega, \boldsymbol{F}^-(\boldsymbol{p} + \boldsymbol{F}^+(\boldsymbol{p})) + \boldsymbol{F}^+(\boldsymbol{p}) = \boldsymbol{0} \tag{4.24}$$

In other words, a point warped from $\boldsymbol{I}_t$ to $\boldsymbol{I}_{t+1}$ with $\boldsymbol{F}^+$ should be warped back to its original place with $\boldsymbol{F}^-$.

Since the photometric optimization problem is based on optical flow, we can apply this filtering method to optical flow maps that results from depth and pose inference. However, it can be seen that two different optical flow maps will be needed, $\boldsymbol{F}^+$ and $\boldsymbol{F}^-$. That means that we would need two different depth maps to deduce these optical flow maps, and we also would require them to be perfectly consistent.

This technique has been used for example in the case of GeoNet [YS18] and by Ranjan et al. [Ran+18].

4.5.5 Depth map consistency

Occlusion zones can also be determined analytically directly from depth and displacement. We can adapt the backward/forward consistency method used for optical flow explained above to have a depth consistency method.

The idea is to adapt the inverse warp function to reconstruct depth maps instead of images. Instead of inverse warping I_{t_2} into $\widetilde{I}_{t_2 \to t}$ to reconstruct I_t, we can try to inverse warp $\widetilde{\theta}_{t_2}$ into $\widetilde{\theta}_{t_2 \to t}$ and only optimize the difference between $\boldsymbol{I}_t$ and $\widetilde{\boldsymbol{I}}_{t_2 \to t}$ where $\widetilde{\theta}_t$ and $\widetilde{\theta}_{t_2 \to t}$ are close. This solution however also requires $\widetilde{\theta}_t$ and $\widetilde{\theta}_{t_2}$ to be both estimated, and these estimations to be consistent.

When training depth maps from scratch, and thus having them random and independent at first, how can we distinguish inconsistencies related to a bad depth estimation from those related to occlusion? We are not certain the neural network won't fall into a degenerate solution

One solution could be to carefully engineer the training workflow. Since this problem typically appears at the end of the convergence, this regularization can be applied with a very low weight at first, and then when the network converges, we can assume sufficient reliability in the depth estimation so that inconsistency is mainly caused by the occlusions.

Unsurprisingly, these techniques of optical flow and depth forward-backward consistency with inverse warp encourage trial and error, and are prone to over-fitting with respect to the validation set. As our main goal is to be robust to drone videos heterogeneity, we need a more general regularization, that does not need to make assumptions on the dataset.

4.5.6 Occlusions detections with direct warp instead of inverse warp

Our main problem with inverse warp consistency check is that it needs two different depth maps. What we want is a way to estimate $\widetilde{\theta}_{t_2}$ only with $\widetilde{\theta}_t$ and $\boldsymbol{T}_{t \to t_2}$. By looking at reprojection equation (4.2), we can see that for each point $\boldsymbol{p}$ in Ω a corresponding point in $\mathbb{R}^2$ is found. The problem of inverse warp was initially to get the color of a point in $\mathbb{R}^2$ using interpolation from points in Ω.

This is now the opposite problem: by a set of point of $\mathbb{R}^2$ with corresponding colors, how can we know the values of points in Ω? This problem can be called depth based image rendering (DIBR) [Sun+10]. Mathematically, this can be understood as finding the inverse of equation 3.4, from chapter 3: for a point $\boldsymbol{p} \in \Omega$, find $\boldsymbol{p_2} \in \mathbb{R}^2$ such that

$$\boldsymbol{p} = \Pi \left(\mathbf{K} \boldsymbol{T}_{t \to t_2} (\widetilde{\theta}_t(\boldsymbol{p}_2) \mathbf{K}^{-1} \Pi_{-1}(\boldsymbol{p}_2) \right) \tag{4.25}$$

Unfortunately, this equation may have several solutions in case of occlusions, because the function Π is not injective. In that case, thanks to the known depth of these points, we can just take the closest point. For ease of notation, let's define the function C (for point cloud) as:

$$\begin{array}{rccc} C: & \mathbb{R}^2 & \to & \mathbb{R}^3 \\ & \boldsymbol{p_2} & \mapsto & \boldsymbol{T}_{t \to t_2} (\widetilde{\theta}_t(\boldsymbol{p}_2) \mathbf{K}^{-1} \Pi_{-1}(\boldsymbol{p}_2) \end{array}$$

Let's also define for each point $\boldsymbol{p} \in \Omega$ the set of possible projected points $\mathcal{P}(\boldsymbol{p})$, and the set of depth values $\mathcal{D}(\boldsymbol{p})$:

$$\begin{aligned} \mathcal{P}(\boldsymbol{p}) &= \{ \boldsymbol{p_2} \text{ s.t. } \boldsymbol{p} = \Pi \left(\mathbf{K} C(\boldsymbol{p}_2) \right) \} \\ \mathcal{D}(\boldsymbol{p}) &= \{ C(\boldsymbol{p_2}).\boldsymbol{u}_z, \boldsymbol{p}_2 \in \mathcal{P}(\boldsymbol{p}) \} \end{aligned}$$

where $\boldsymbol{u}_z = [0, 0, 1]$ is the forward unit vector. We can then define the direct projection of $\widetilde{\theta}_t$ as:

$$\forall \boldsymbol{p} \in \Omega, \widetilde{\theta}_{t \to t_2}(\boldsymbol{p}) = \begin{cases} +\infty & \text{if } \mathcal{D}(\boldsymbol{p}) = \emptyset \\ \min \mathcal{D}(\boldsymbol{p}) & \text{otherwise} \end{cases}$$

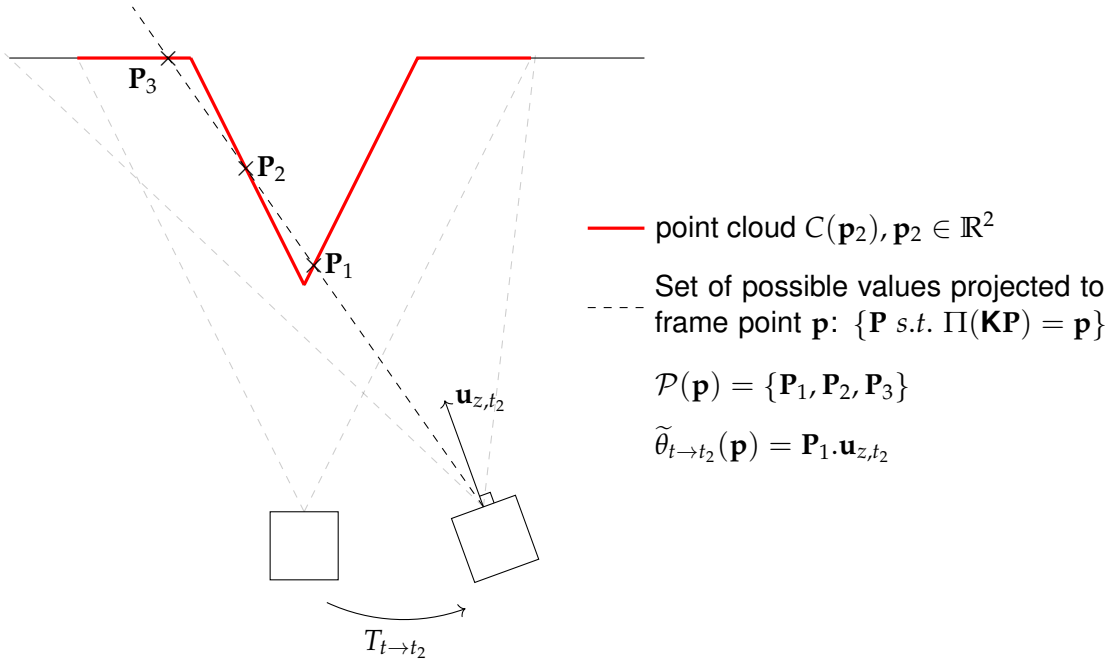


Figure 4.9: Illustration of the direct projection workflow. The pixel p characterizes an occluded view with multiple possible depth values

Figure 4.9 shows an example where the projected depth has multiple possible values for a particular pixel p.

Assuming we can perform direct warping, the idea is then to warp $\widetilde{\theta}_t$ into $\widetilde{\theta}_{t\to t_2}$ and then either warp $\widetilde{\theta}_{t\to t_2}$ back to $\widetilde{\theta}_{t\to t_2\to t}$ or identify pixel that were painted with a 2D map of integer indices Id.

The index map Id is a way to retrieve what pixel was used in the point cloud constructed from $\widetilde{\theta}_t$ and the transformation $T_{t\to t_2}$ to be projected in each pixel of $\widetilde{\theta}_{t\to t_2}$.

$$\forall \boldsymbol{p} \in \Omega, Id(\boldsymbol{p}) = \begin{cases} \textbf{NaN} & \text{if } \mathcal{P}(\boldsymbol{p}) = \emptyset \\ \arg\min_{\boldsymbol{p}_2 \in \mathcal{P}(\boldsymbol{p})} \left(C(\boldsymbol{p}_2).\boldsymbol{u}_z\right) & \text{otherwise} \end{cases}$$

We can then summarize these two methods:

1. Warp back:

$$\widetilde{\theta}_t \xrightarrow{T_{t\to t_2}} \widetilde{\theta}_{t\to t_2} \xrightarrow{T_{t_2\to t}} \widetilde{\theta}_{t\to t_2\to t}$$

   If the warp operation is accurate enough, we can safely assume that differences between $\widetilde{\theta}_t$ and $\widetilde{\theta}_{t\to t_2\to t}$ are occlusion areas.

$$\forall \boldsymbol{p} \in \Omega, occ(\boldsymbol{p}) = \begin{cases} \textbf{False} \text{ if } \widetilde{\theta}_{t\to t_2\to t}(\boldsymbol{p}) = \widetilde{\theta}_t(\boldsymbol{p}) \\ \textbf{True} \text{ otherwise} \end{cases}$$

2. Index back:

$$\widetilde{\theta}_t \xrightarrow{T_{t\to t_2}} \begin{matrix} \widetilde{\theta}_{t\to t_2} \\ Id \end{matrix}$$

   We can identify which point was painted and which one was not from the index map Id that we created along $\widetilde{\theta}_{t_2}$.

$$\forall \boldsymbol{p} \in \Omega, occ(u, v) = \begin{cases} \textbf{False} \text{ if } \boldsymbol{p} \in Id(\Omega) \\ \textbf{True} \text{ otherwise} \end{cases} \tag{4.26}$$

Direct warp Module Presentation

As a proof of concept, we propose a consistency check using direct warping instead of inverse warping. While the inverse warp will be used for photometric error minimization, occlusion areas will be detected thanks to this consistency check using direct warping. We will then be able to dismiss these areas from being optimized with photometric error. Besides, no differentiation needs to be done with direct warping, allowing us to use a very simple pipeline.

Here, instead of finding the exact set of possible depth for each point, which would imply using complicated geometry techniques such as ray-tracing [Whi05], we chose to do the opposite: from a set of points in $\mathbb{R}^2$, we discretely paint pixels in Ω on the output that are close to points actual projections. This technique is known as rasterization: it is much simpler but also less precise and not easily differentiable, although the differentiability aspect is not our priority since we only want to know occlusion maps.

To model direct warping, we assumed the initial depth map θ_t to be not continuous as we did for I_{t_2} for inverse warping, but a "square cloud". The difference with a point cloud is that each point has a "width". The width is computed so that the projection of the 3D square into the initial frame is exactly 1 pixel large. This algorithm is a simplified version of McMillan's image-space Gaussian reconstruction [MJ97].

For frame coordinates we take the convention that for an integer coordinate, the point is at the center of the square pixel. As a consequence, when we have a float frame coordinate point (u, v), the corresponding pixel to paint is at $(round(u), round(v))$

The rationale for square cloud is that when we have a forward movement, and thus a zoom in our depth map, disocclusions don't appear, because along with the distance between the squares, their width is also increased.

The algorithm for our projection module is presented algorithm 1.

Algorithm 1 Compute $\widetilde{\theta}_{t\to t_2}$

set $\widetilde{\theta}_{t\to t_2}$ to infinity everywhere
set Id to **NaN** everywhere
for all $\boldsymbol{p} \in \Omega$ **do**
  $Width = \widetilde{\theta}_t(\boldsymbol{p})$
  $\boldsymbol{P'} = \mathbf{K} T_{t\to t_2} (\widetilde{\theta}_t(\boldsymbol{p}) \mathbf{K}^{-1} \Pi_{-1}(\boldsymbol{p}))$
  $X, Y, Z = \boldsymbol{P'}$
  $u_0 = \frac{X - 0.5 \times Width}{Z}$
  $v_0 = \frac{Y - 0.5 \times Width}{Z}$
  $d = \frac{Width}{Z}$
  $v = round(v_0)$
  do
    $u = round(u_0)$
    do
      if $(u, v) \in \Omega$ **and** $Z < \widetilde{\theta}_{t\to t_2}$ **then**
        $\widetilde{\theta}_{t\to t_2} \leftarrow Z$
        $Id(u, v) \leftarrow \boldsymbol{p}$
      end if
      $u \leftarrow u + 1$
    while $u \leq u_0 + d$
    $v \leftarrow v + 1$
  while $v \leq v_0 + d$
end for

This pipeline is very simple as it always assumes the squares to be perfectly normal to the camera axis. As a consequence, this algorithm is not perfectly robust to rotations. In the case of rotations in our scene large enough for side effects not to be negligible, we could easily extend this algorithm to include square inclinations, but because our scenes usually have very low displacements relative to each other, we choose to keep it simple. The algorithm is not developed in this report, but we extended it to also warp the corresponding image pixel for visualization purpose, using the index map Id. That way we can see which pixel is mapped where. As mentioned above, unlike inverse warping, this operation is not spatially differentiable. Some works have been done to to make rasterization and even ray tracing differentiable [Li+18; LB14], but we don't need that functionality here, as we just want to identify the occlusion areas.

Occlusion detection module

Once we got direct warping developed, we know the occlusion and valid areas for $\widetilde{\theta}_{t\to t_2}$ with respect to $T_{t_2\to t}$. To get the equivalent in θ_t we can first look at the method based on index map (see equation 4.26).

This method we call "index back" will have a sparsity problem: overlapped pixels will not be indexed and thus considered occluded while a continuous rasterization would have made them appear. As a consequence, the filtered gradient map from photometric error is very sparse.

However, our main goal is to get rid of false negative for our occlusion filter: pixel that are falsely considered non-occluded should be avoided, at the cost of potential false positive if they are evenly distributed.

Because of its simplicity, our pipeline might not be robust to inclined plane. Indeed, unwanted disocclusion can appear, some background parts normally occluded will be visible, indexed and optimized while they should not be. As mentioned above, this problem is partly solved by the assumption of little displacement, but can appear when considering very thin local maxima.

The indexed map could probably be improved by filtering with e.g. morphological filters or median filters, but we can also chose to do the same direct warp operation with $\widetilde{\theta}_{t\to t+1}$ to avoid foreground sparsity. This is particularly interesting when considering false negative regarding occluded background. With a foreground continuous enough, false negative will be sparse, while true positive won't be. It means we can filter out these values with a simple dilation operation applied on the occlusion mask. The valid map is simply the coordinates where $\widetilde{\theta}_{t\to t_2\to t}$ is within a range of $\widetilde{\theta}_t$.

$$\alpha > 1, \forall (u,v) \in \Omega, occ(u,v) = \begin{cases} \textbf{\textit{False}} \text{ if } \widetilde{\theta}_{t\to t_2\to t} \in [\frac{1}{\alpha}\widetilde{\theta}_t, \alpha\widetilde{\theta}_t] \\ \textbf{\textit{True}} \text{ otherwise} \end{cases}$$

Figure 4.10 shows a comparison with "index back" and "warp back + erosion of valid pixels by 2 pixels". This example features a depth map with a foreground at his center. The considered displacement is The depth map has a noise of $\sigma = 0.1$ and false negative appear beneath the foreground during warp (first frame).

4.5.7 Conclusions on occlusion considerations

This section presented the fact that in addition to not being injective which is partly solved with smooth regularization, the inverse warp function is not perfectly representative with respect to depth and movement. True minimum of the resulting photometric loss function is not indicating a true depth as soon as the scene features occlusions. Filtering invalid

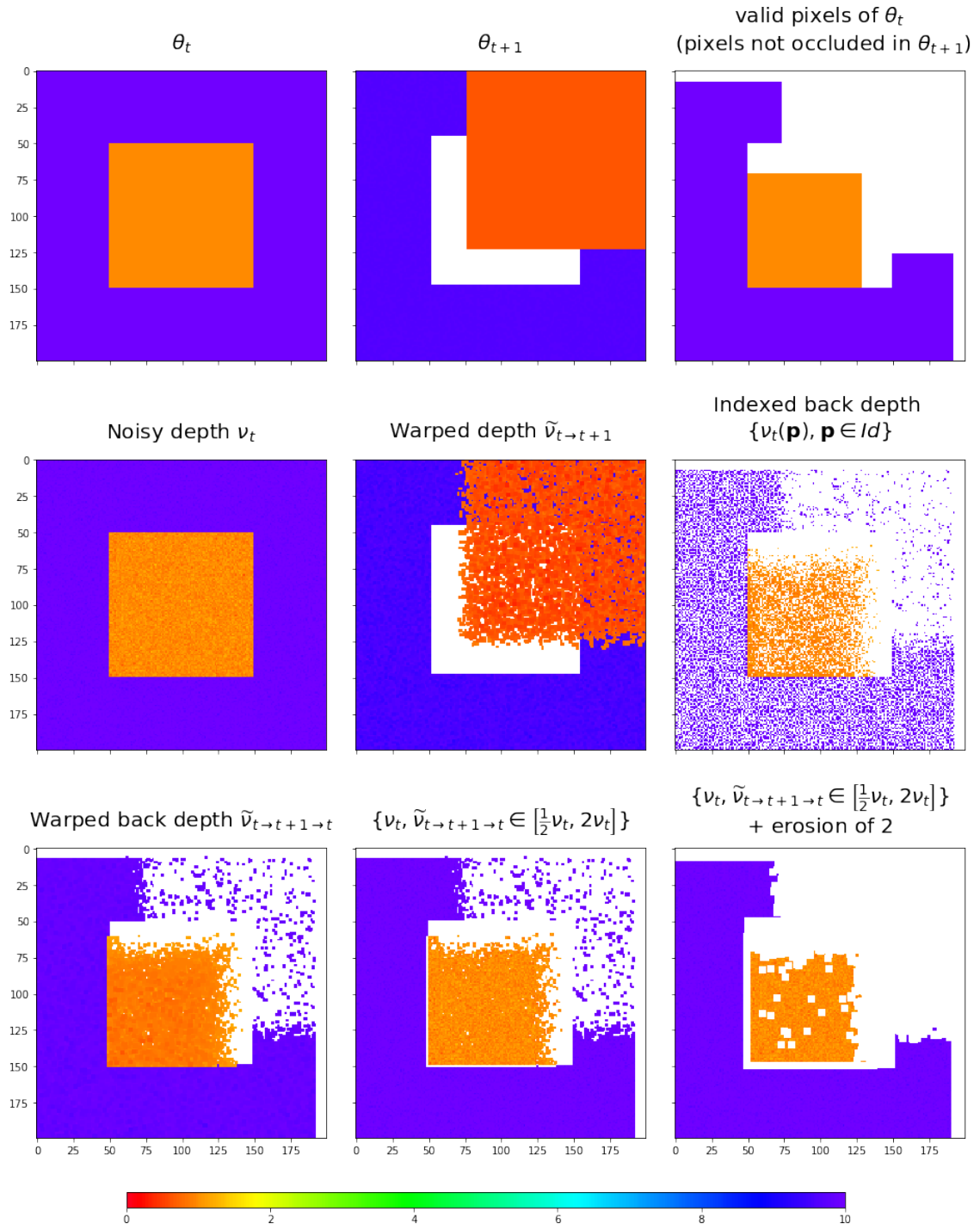


Figure 4.10: Illustration of occlusion detection module. **NaN** values are white colored. Displacement is $\frac{\sqrt{2}}{2}[-1, 1, 1]$: the camera goes down left and forward, field of view is $90°$. First row: presentation of θ_t and theoretical results on this perfectly smooth depth map. Notice the foreground is also occluded because it goes out of field of view. Second row: presentation of on ν, a noisy equivalent of θ_t, with a Gaussian noise with std of 0.1 $\nu_t = \theta_t + \mathcal{N}(0, 0.1)$. Result of direct warping (center) and filtered depth map with index back method. Last row: warp back of warped depth $\widetilde{\nu}_{t\to t+1}$, and filtered depth map using the warp back method, followed by an erosion of 2 pixels.

inverse warp areas is an attempt at making the photometric loss indeed minimum at the perfect value.

We now have along photometric error two studied tools to train a network to retrieve the depth of a picture within a sequence of frames. The next step in elaborating an unsupervised training workflow is to determine optimal values.

4.6 Determination of optimal hyper-parameters with toy problems

We now want to make a comprehensive study of our potential options for an unsupervised training before deploying it on a big drone dataset with our DepthNet network.

In order to study the different works on unsupervised depth learning, we decided to evaluate it on a very simple version of the problem. The key here is to evaluate the training quality and how it compares with the ideal supervised case, and not the network quality regarding over-fitting. As such, we reduce the problem to over-fit a small number of different depth maps: depth values are the very trainable parameters we apply the gradient descent on, which is very similar to DTAM [NLD11]. In this toy problem, we make abstraction of the neural network: we simply want a training algorithm that can converge to a reasonable and stable depth map in the hope that it will be a good candidate for the full workflow with a neural network that tries to infer depth from images.

Indeed, one of the main drawbacks of this photometric based training is the lack of stability. It has been shown thanks to a test set with ground truth that photometric loss was not a good quality measure, and that depth quality did worsen when the training was too long [Zho+17]. This indicates that training on a totally ground truth-less algorithm with only photometric loss is impossible since it would require a validation set for early stopping (as presented in [Pre12] among other methods to prevent overfitting) in order for the network to stop training further and compromising its quality.

It must be noted that this is different from over-fitting: the quality also worsens for training frames. It will appear that even in our toy problems that is designed to be over-fitted, a lack of good regularization can make our depth maps diverge, while still lowering the photometric loss.

4.6.1 Scenes presentation

From physical assumptions made section 4.4.2, we want to design scenes that will test the robustness of an algorithm to them.

Namely, we identified problems that an auto-supervised training algorithm should be able to solve.

- A scene with two clear grounds, where the foreground has some texture discontinuity while being perfectly smooth.
- A scene with depth discontinuity that triggers occluded area from one view point to another.
- a scene with an infinite plane, not normal to camera plane
- a scene with a texture-less background, on which the regularization should not be applied.

To feature these problems, we focus on two very simple scenes, shown Figure 4.11:

1. a scene composed of one static foreground object and a background. This scene will be further referred to as "Flower", since the foreground is textured with a daisy.

2. a scene of an infinite horizontal plane. This scene will be referred to as "Horizon".

The scenes have three frames and a known displacement between the frames. The goal of this experiment is to find the depth map of the center frame only using reprojection optimization. Thanks to the uniformity of displacement, for the flower scene, the occluded areas shown figure 4.12 are not the same, which makes it theoretically possible for every pixel to be retrieved in at least one of the 2 frames.

A workflow not able to converge in our examples might not be robust to train on a larger dataset such as KITTI [Gei+13], Still Box or our unlabeled drone sequence without the necessity of a validation dataset with depth ground truth. This statement is not necessarily true as stochasticity and sufficiently heterogeneous examples during training can lead to implicit regularization. As a consequence, the optimal learning schedule can be different from one dataset to another.

A first result on optimization can be seen figure 4.13 where only the photometric loss is optimized. This failed convergence serves as a baseline for our next regularization losses.

Metrics

To evaluate optimization tricks on our toy problems, we have to design a set of meaningful metrics for a potential extension to a wider problem. More specifically, we must optimize the error for both our scenes.

As a consequence, we must look at a joint metric that heavily penalizes performance discrepancy. A good way to do so is to find a normalized metric that is 1 when the estimated depth map $\widetilde{\theta}$ is perfect, 0 if it's worse than our starting point (with everything set to background depth). That way, by the geometric mean of the scene metrics, we get a performance index that gives priority to consistency.

If we look at a standard performance evaluation such as logarithmic error LE (see chapter 3), for a particular scene $\mathcal{S}$ with associated depth target $\theta^{\mathcal{S}}$ and initial depth value $\widetilde{\theta}_0^{\mathcal{S}}$, the normalized metric is then

$$LE(\theta, \widetilde{\theta}) = \left\| \log(\theta) - \log(\widetilde{\theta}) \right\|$$

$$\mathcal{M}_{\mathcal{S}}(\widetilde{\theta}^{\mathcal{S}}) = \begin{cases} 0 & \text{if } LE(\theta^{\mathcal{S}}, \widetilde{\theta}^{\mathcal{S}}) > LE_0 \\ 1 - \frac{LE(\theta^{\mathcal{S}}, \widetilde{\theta}^{\mathcal{S}})}{LE_0} & \text{otherwise} \end{cases}$$

with LE_0 the initial error: $LE_0 = LE(\widetilde{\theta}_0^{\mathcal{S}}, \theta^{\mathcal{S}})$. This value depends on the starting depth map.

The general quality metric is then for a set of k scenes $\mathbb{S} = \{\mathcal{S}_0, \mathcal{S}_1, \cdots, \mathcal{S}_k\}$

$$\mathcal{M} = \sqrt[|\mathbb{S}|]{\prod_{\mathcal{S} \in \mathbb{S}} \mathcal{M}_{\mathcal{S}}}$$

In our case k is only 2, but should we find a new suited scene representative of a characteristic problem not found in the first two, the formula would still hold.

From now on, unless specified otherwise, every quality measure will be based on $\mathcal{M}$

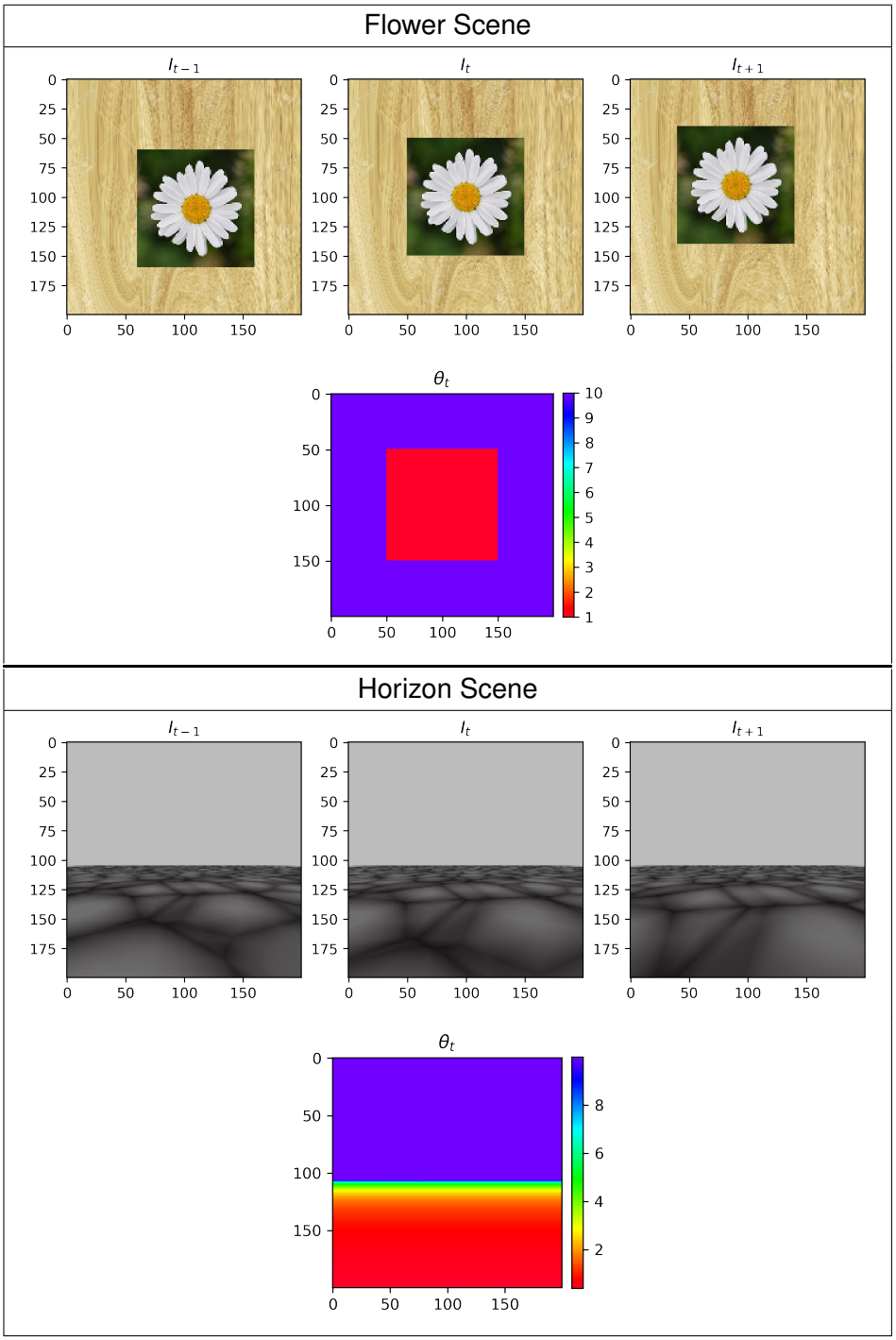


Figure 4.11: Our two toy problems "Flower" (top) and "Horizon" (bottom). For each scene, first row are the three frames of the scene, and second row is the ground truth depth of the center frame. Images are 200×200 pixels. Scenes have no rotations. Flower scene has a displacement of $\frac{1}{10}[1, 1, 0]$ (camera goes right and down, so the image goes left and up), Horizon scene has a displacement of $\frac{1}{10}[0, 0, 2]$ (camera goes forward). Camera is a standard centered pinhole with 90° of field of view. For optimization depth starting points are equal to background, i.e. 10m

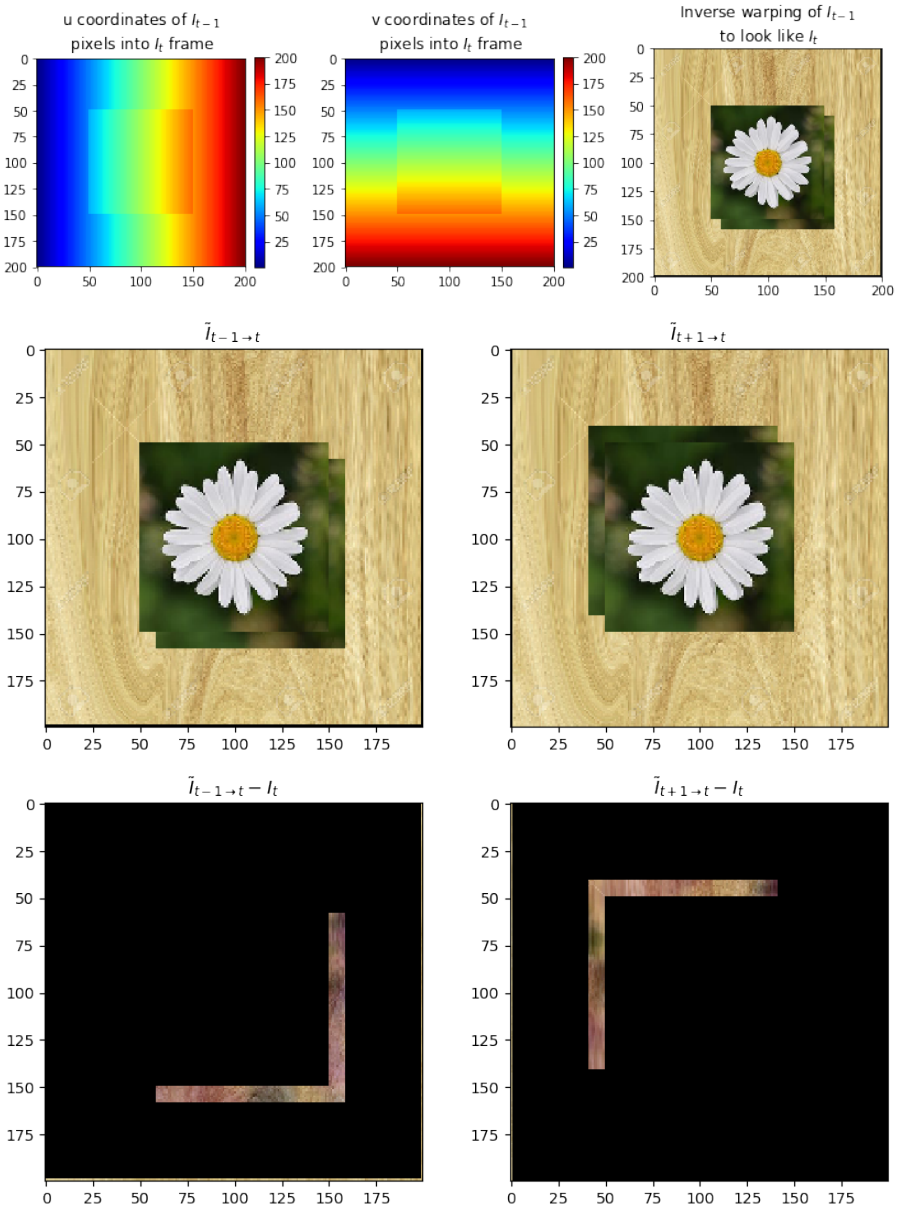


Figure 4.12: First row: Illustration of optical flow construction, and how a perfect depth produces imperfect warping. A "ghost" of the foreground can be seen on occluded areas, which the reprojection based optimization will wrongly try to solve. Second row, our two imperfectly warped frames from I_{t-1} and I_{t+1} to I_t. Last row, the differences between warped images and I_t are the occlusion areas. We can see that the non-zero areas don't cover the same pixels.

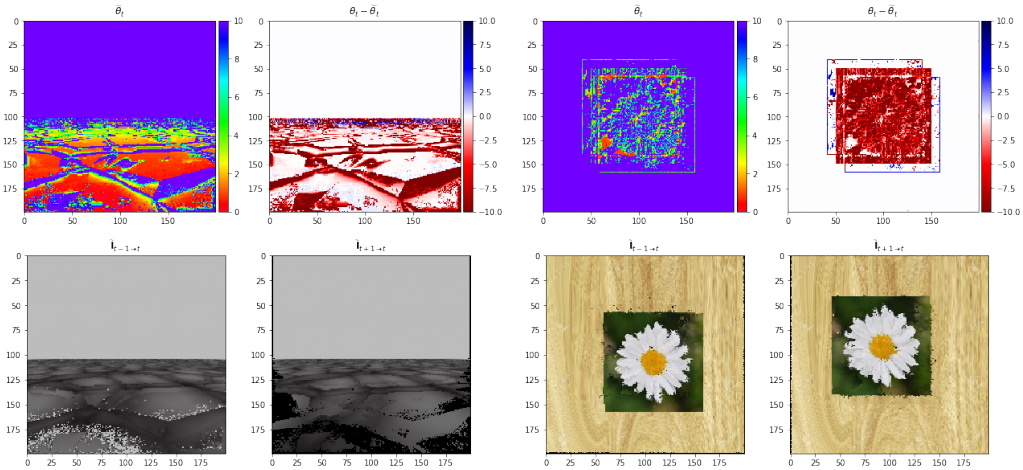


Figure 4.13: optimization results when only photometric loss is applied for depth optimization. Mean logarithmic error is worse at the end than at the beginning for both cases. For each scene, from left to right and top to bottom: depth result, difference with ground truth, final inverse warp of I_{t-1} and final inverse warp of I_{t+1}

4.6.2 Tests on smooth loss regularization

From section 4.4.3, we know that the value we want to apply diffusion on is not our trainable parameters θ, but the inverse depth map $\xi = \frac{1}{\theta}$.

Smooth loss functions presented section 4.4.10 will be tested. For each smooth loss (TV, TVV, diffusion and gradient diffusion), we will try to find the best loss hyper parameters for the measure $\mathcal{M}$, and we will be able to compare them.

Robust smooth losses

As we advised section 4.4.5, we will also try "robust" versions of the considered loss function. In our tests (sec 4.6), they will be denoted by the prefix "robust".

The robust smoothing will consist in a simple gradient descent applied on a clone of ξ. It needs two parameters to compute the diffusion of our output, the optimization step, just as a regular optimization, and the number of iterations. In the following tests, to avoid unnecessary overhead of loss computation, we chose a step value of $\gamma = 0.1$ and 10 iterations. For distance function, mean square error will be used.

Besides, the training parameters are depth values, while the smooth loss functions are applied on inverse depth ξ. This makes a trivial use-case where applying a "robust" version of a smooth losses, might be useful. The derivative of the inverse function is highly non linear (see appendix D.3). As such, all our tests with smooth losses will be done using a regular potentially unstable version, and their robust counterpart.

Hyper-parameter search

The main goal of this experiment is to find a loss that is robust to our two supposedly complementary scenes. The two main hyper-parameters that can be searched are the influence of textureness over smooth loss dampening κ and the relative weight λ with which the smooth loss is applied, compared to the photometric loss.

Namely, given a smooth loss $\mathcal{L}_s(\xi, \kappa)$, we apply the loss

$$\mathcal{L}(\theta) = \mathcal{L}_p + \lambda \mathcal{L}_s(\xi, \kappa)$$

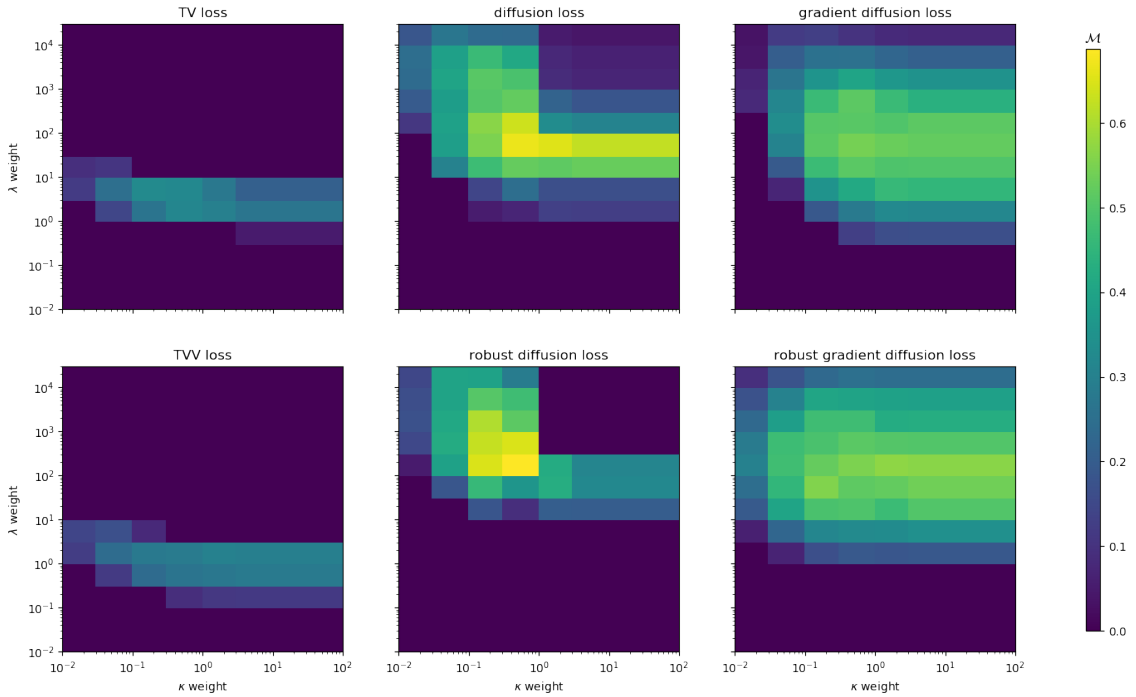


Figure 4.14: Hyper-parameter search for λ and κ for our two scenes. An ideal (λ, κ) value should maximize the metric $\mathcal{M}$

It should be noted that for any loss function $\mathcal{L}_s$, either λ or κ being 0 is equivalent to no smooth loss applied.

Figures 4.14 and 4.15 present the results of a logarithmic grid search for parameters λ and κ. Figures 4.16 and 4.17, present optimization results on our toy problems after selecting λ and κ maximizing measure $\mathcal{M}$ for each loss.

Surprisingly, the diffusion methods (robust or regular) tend to the best results, even for our infinite plane in Horizon.

This can be mitigated by three observations:

- As shown figure 4.15, performance on Horizon Scene is very erratic for diffusion loss, for an unknown similar scene, there is a chance that the chosen hyperparameters (λ, κ) will perform very poorly. This is mostly due to not respecting the uniform background requirement, because the sky gets diffused with the foreground depth.

- It can be observed figure 4.17 that for Horizon scene, diffusion losses actually worsen over time after reaching a very good depth map, while gradient diffusion losses seem to keep going down until the end of the optimization. This shows that gradient diffusion losses may be more robust to extended training workflows which may happen when no ground-truth is available at validation time.

- When looking at parameter change robustness figures 4.14 and 4.15, gradient diffusion optimum appears to be more evenly distributed, without a strong peak at optimum. As a consequence, considering a different set of scenes would probably mean a set of optimal hyper-parameters somewhat different, gradient diffusion will probably outperform regular diffusion

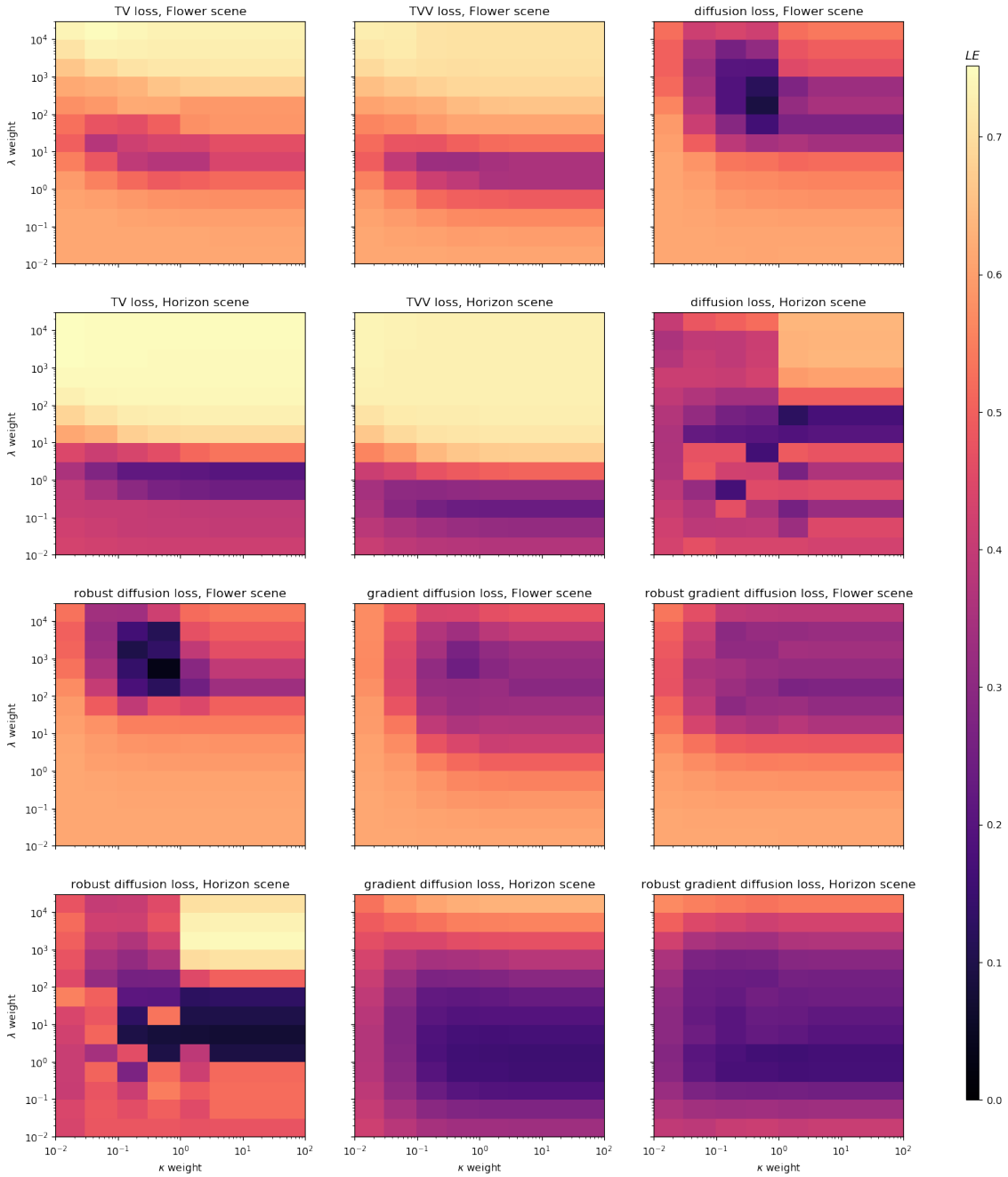


Figure 4.15: Logarithmic error (LE) for our two scenes, using different values of λ and κ.

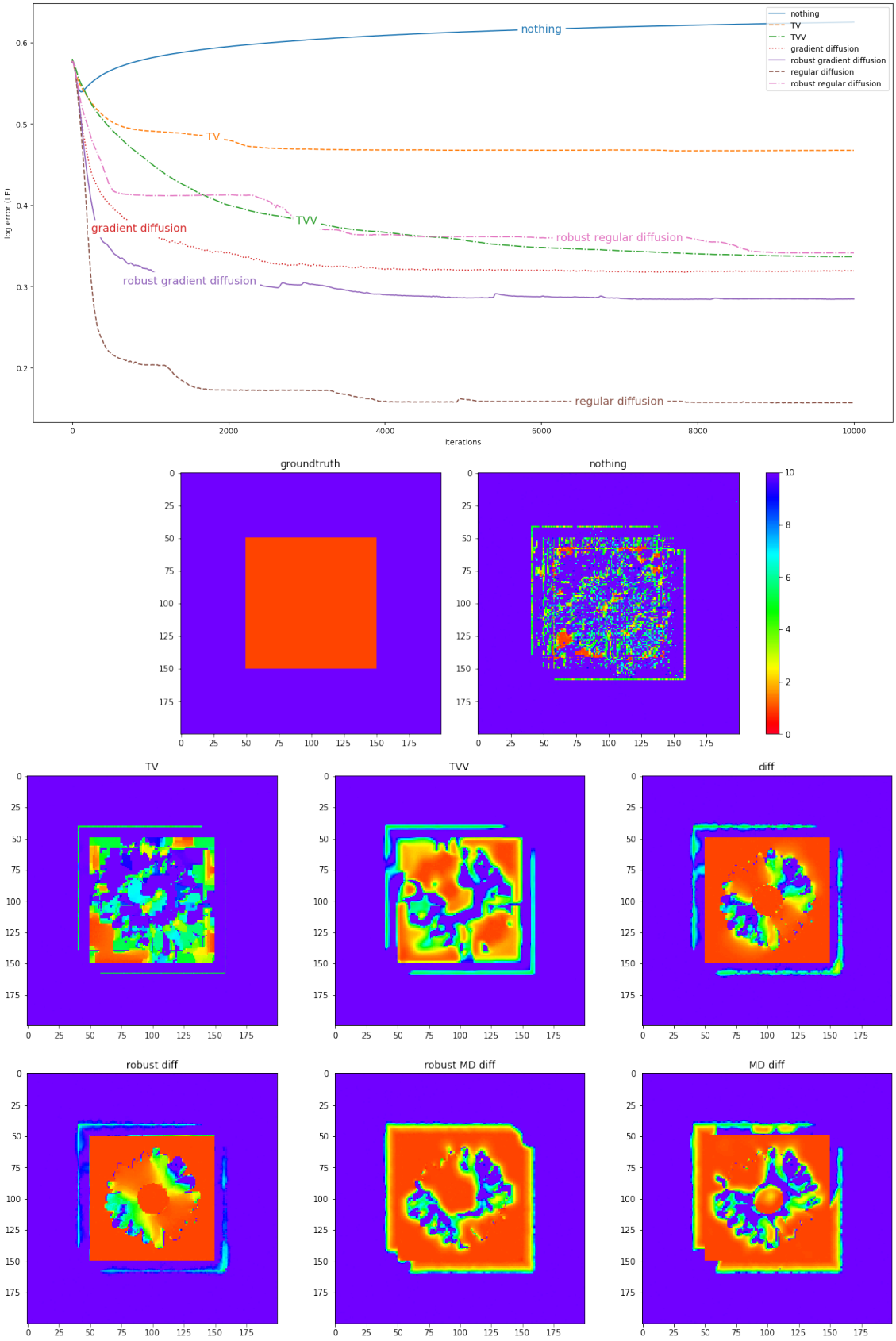


Figure 4.16: Result of a photometric optimization with optimal weights λ and κ for our 6 different smooth losses on the Flower scene after 10^4 iterations. Learning rate is 0.15

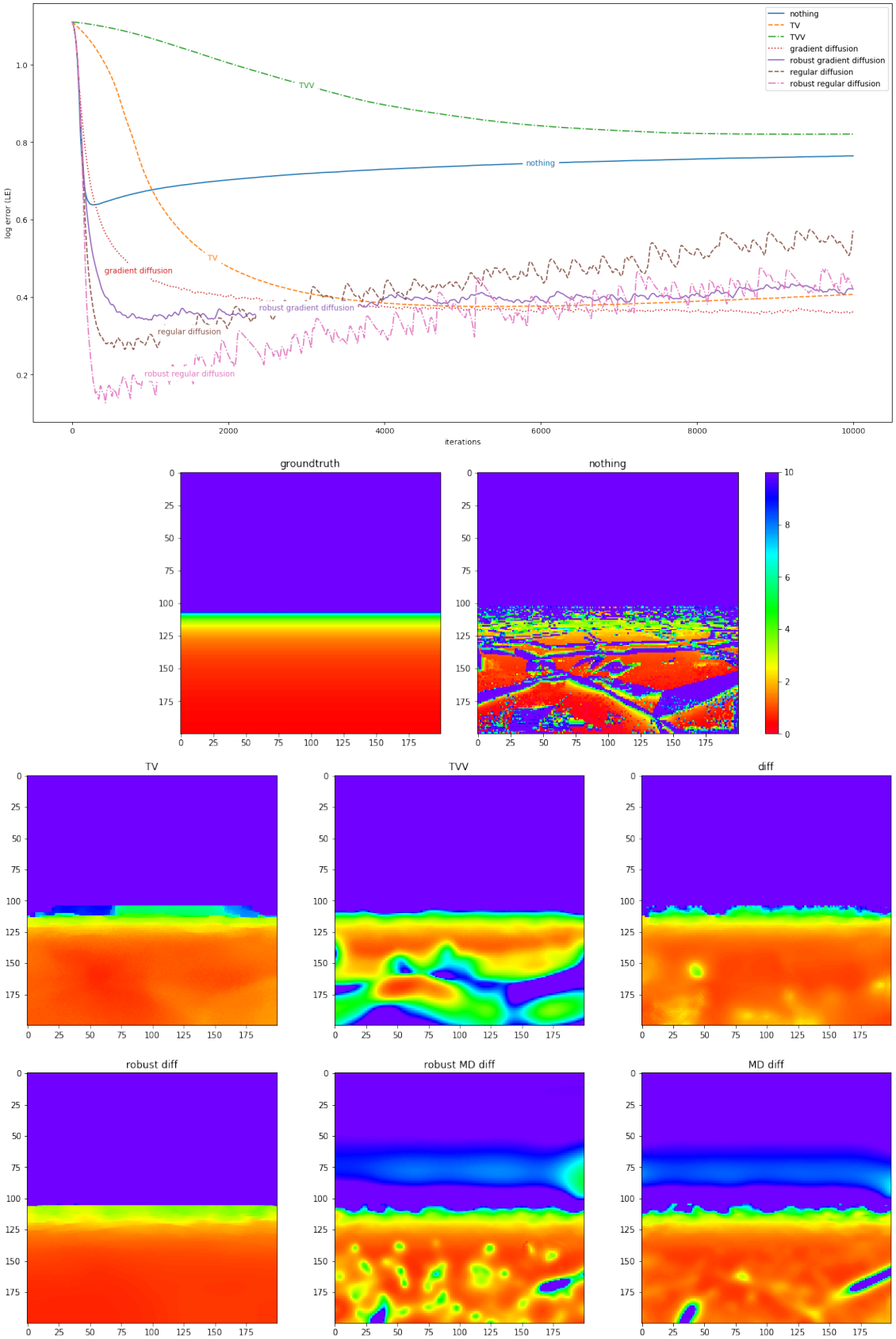


Figure 4.17: Result of a photometric optimization with optimal weights λ and κ for our 6 different smooth losses on the Horizon scene after 10^4 iterations. Learning rate is 0.15.

Using Occlusion Module

Decent results using only smooth loss regularization have to be mitigated with the fact that foreground has a totally different color distribution than background. Which then makes the assumption of high color gradient for depth discontinuities plausible. But as seen figure 4.16, this has to be done with a compromise on textured planes (with no depth discontinuity), and the foreground is not perfectly determined. Keeping a κ parameter high enough to get good results on foreground worsens the background and especially the occlusion zones.

To check the relevance of our proof of concept occlusion mapper, we test this regularization with our toy problem, using our different losses, after a new hyper-parameter search. A quantitative comparison can be seen Table 4.1.

In every smooth loss, the quality index $\mathcal{M}$ is higher, even making TVV Loss eventually a good candidate. Both optimal κ and λ values are generally lower. This is expected for λ, as now several image areas are filtered from the photometric loss. The smooth loss then has no concurrency on them and thus don't need to be high. However, this is unexpected for κ, as we would have though that low κ values would be problematic for textured planes such as the Flower scene. It can be noted that the robustness index $\Delta\mathcal{M}$ is worsened for diffusion and robust diffusion, while it was already their weak point, making gradient diffusion and robust gradient diffusion an even more interesting candidate regarding training on unknown datasets.

A final significant improvement of this occlusion regularization is the "ghosting" that is not longer appearing on our Flower scene. Figure 4.18 shows a comparison for the resulting depth map with regular diffusion.

This module then seems to help convergence without largely penalizing scenes without occlusions.

A more sophisticated warping algorithm, seems to be an interesting improvement axis to efficiently regularize occlusions.

4.7 Photometric losses

This section will discuss several variations on photometric loss. For our toy examples, we only used the simple photometric loss, which is obtained with the absolute difference between target images and reference frames, but some techniques have been studied to cope with large displacement, or non constant luminosity.

4.7.1 Multi-scale photometric loss

Multi scale (or pyramidal) loss consists in trying to minimize the photometric reprojection loss with multiple resolutions. This assumption is generally done to allow the photometric loss to have a greater receptive field, this is the main idea behind optical flow method and its variations such as Anandan [Ana89] or Farnebäck [Far03]. Indeed, when dealing with large optical flow, due to a high displacement magnitude or a small depth, gradient from photometric loss is not meaningful. Depth regularization can help, as it has been shown with our tests, but it requires an overlap of areas with the same modality. In the example of our occluded scene, if the foregrounds did not overlap, it would have been impossible to make the depth converge. As a consequence, the maximum optical flow over which foreground objects don't overlap and thus make depth impossible to converge directly depends on the size of foreground objects.

As a consequence, if we have a small foreground zone, without multi-scale loss, we might be unable to learn from it. However, for multi-scale loss to be usable, we would

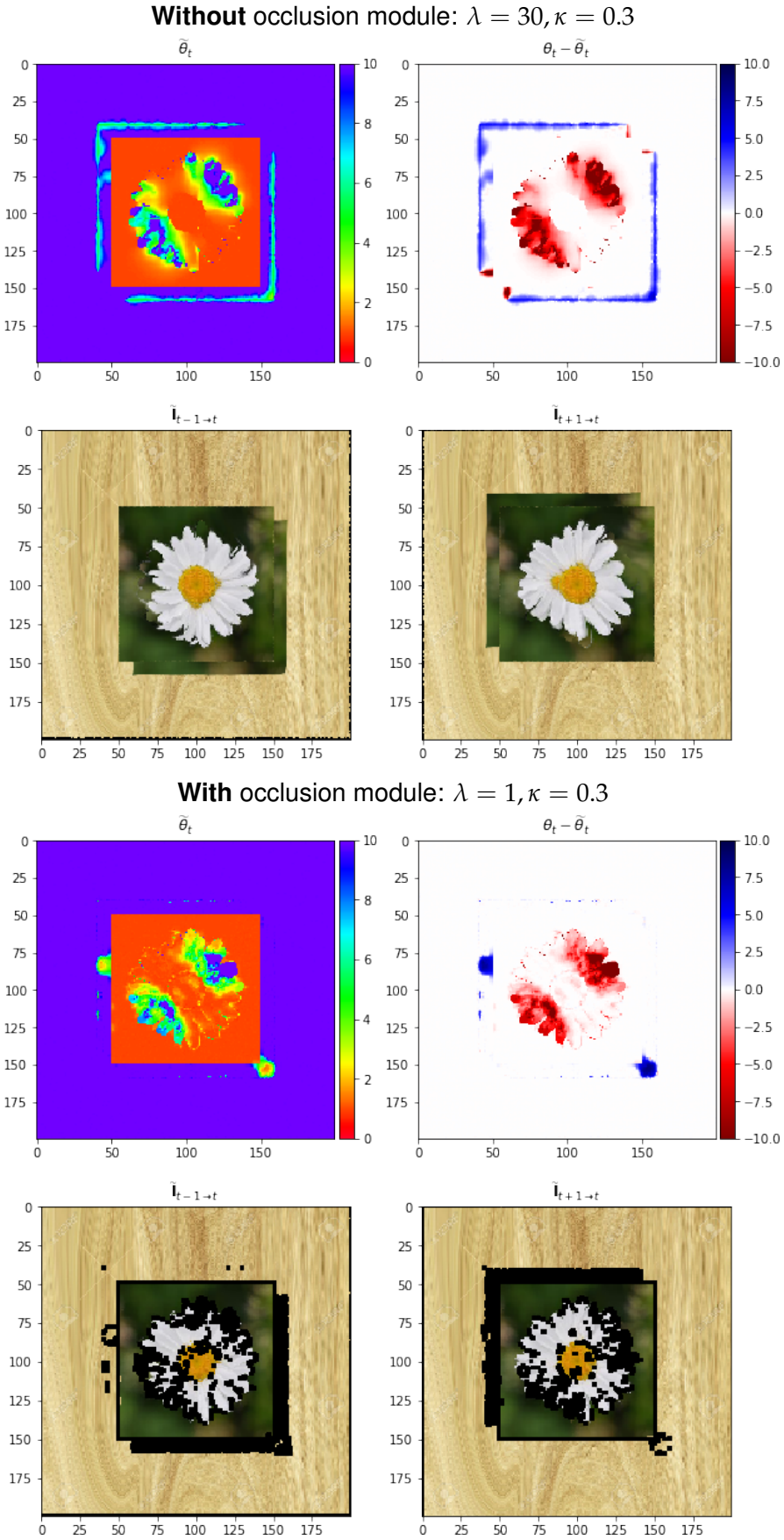


Figure 4.18: Comparison of a simple optimization (first two rows) and an optimization using occlusion module (last two rows). Optimal parameters for regular diffusion loss are used. Blacked parts of warped images are ignored thanks to our occlusion module. Thanks to this, the usual ghost effect of depth on background is avoided.

	ODM	
TV Loss	✗	✓
κ	1	0.1
λ	1	1
$\mathcal{M}$	0.34	**0.63**
$\Delta\mathcal{M}$	0.45	**0.44**
LE (Horizon)	0.37	**0.36**
LE (Flower)	0.48	**0.24**

	ODM	
TVV Loss	✗	✓
κ	3	0.3
λ	1	0.1
$\mathcal{M}$	0.36	**0.74**
$\Delta\mathcal{M}$	0.31	**0.15**
LE (Horizon)	0.86	**0.26**
LE (Flower)	0.36	**0.16**

diffusion	✗	✓
κ	0.3	0.3
λ	30	1
$\mathcal{M}$	0.67	**0.86**
$\Delta\mathcal{M}$	**0.32**	1.05
LE (Horizon)	0.42	**0.12**
LE (Flower)	0.16	**0.10**

robust diffusion	✗	✓
κ	0.3	1
λ	100	1
$\mathcal{M}$	0.69	**0.84**
$\Delta\mathcal{M}$	**0.52**	0.62
LE (Horizon)	0.44	**0.07**
LE (Flower)	**0.12**	0.14

gradient diff	✗	✓
κ	0.3	0.3
λ	30	0.3
$\mathcal{M}$	0.54	**0.77**
$\Delta\mathcal{M}$	**0.11**	0.15
LE (Horizon)	0.36	**0.16**
LE (Flower)	0.32	**0.17**

robust gradient diff	✗	✓
κ	0.3	0.3
λ	30	0.3
$\mathcal{M}$	0.57	**0.77**
$\Delta\mathcal{M}$	0.23	**0.14**
LE (Horizon)	0.40	**0.12**
LE (Flower)	0.29	**0.20**

Table 4.1: Best hyper parameters found and resulting quality indicators, when occlusion detection module (ODM) is active and when it is not. Red measures are to maximize, blue ones are to minimize

need to downscale the picture enough for the foreground zones to be less than one pixel across. This is required for gradients to be useful. In practice, this could require to downscale the picture a lot, with the loss of texture and precision that comes with it

Another important element is that as shown in DepthNet and almost every variation of FlowNet [Dos+15], multi-scale training of a wanted map (be it optical flow, stereo disparity or depth) is very beneficial to the network, even in a supervised manner. It has been shown by Godart [GMAB18] that multi-scale learning for auto-supervision was not beneficial for the photometric loss, but rather for the network. In this work, he outputted downscaled depth maps, but did not use it to make a downscaled photometric error. Instead, he upscaled it to get a full scale photometric error, and could still easily make the network converge, better than with the highest scale depth output alone.

Besides, multi-scale photometric error can trigger specific error, because the downscaling operation is similar to a low pass filter. As a consequence, high frequency pattern will lose their textureness on smaller pyramidal levels. This is one of the greatest difficulties of in-car environment datasets such as KITTI[Gei+13] or Cityscapes[Cor+16], where the road which is normally very textured appears homogeneous at low resolution. The basic idea behind this problem is that multi-scale photometric error is not robust where pyramidal refinement is not.

It thus appears that multi-scale photometric error is useful only at the beginning of the training, the same way pyramidal refinement is only useful when optical flow error is above one pixel.

The recommended training schedule can introduce multi-scale photometric error, but should be dismiss it as soon as the network can converge without it. When considering small enough displacements, it might never be useful, this is the case for KITTI where convergence is already possible at the beginning. Multi-scale photometric loss might also be dispensable when considering a pretrained network if the network performance is already decent enough.

4.7.2 Minimal loss sampling

Minimal loss sampling has been introduced by Godard et al.[GMAB18] as a way to automatically eliminate ambiguity. The assumption is the same as the one that made smooth losses work well in the first place: each ground (e.g. background or foreground) in the image has a specific color distribution. As a corollary, color differences of pixels from the same ground will be significantly lower than for colors of pixels from different planes.

When considering several inverse warps at the same time, we get multiple photometric loss maps. Minimizing the pixel wise minimal difference is deemed to be equivalent to applying optimization only on colors from the same ground.

$$\mathcal{L}_p = \sum_{\boldsymbol{p} \in \Omega} \min_k \left(d(\boldsymbol{I}_t - \widetilde{\boldsymbol{I}}_{t_k \to t})(\boldsymbol{p}) \right)$$

The consequence of this is that suppressing "ghosting" is possible for occluded areas, provided there exists another frame where that same area is not occluded. This is for example the case in the Flower scene of our toy problem.

The main advantage of this technique is its simplicity, with no need to compute any custom module, and no computational overhead.

When testing with our toy problem, we can see figure 4.19 that indeed, no false positive can be seen on the background. However, foreground depth does not succeed in converging to right values, even with a very high smooth loss. The reason why the convergence is so poor in our case while it has shown great results and adhesion elsewhere [Cas+19] needs further investigations.

It might be due to the lack of stochasticity. Indeed, it appears that convergence is stopped very quickly, being stuck in a local minimum very specific to our example. Concretely, when adding noise to our examples at each optimization step, minimal sampling was improved.

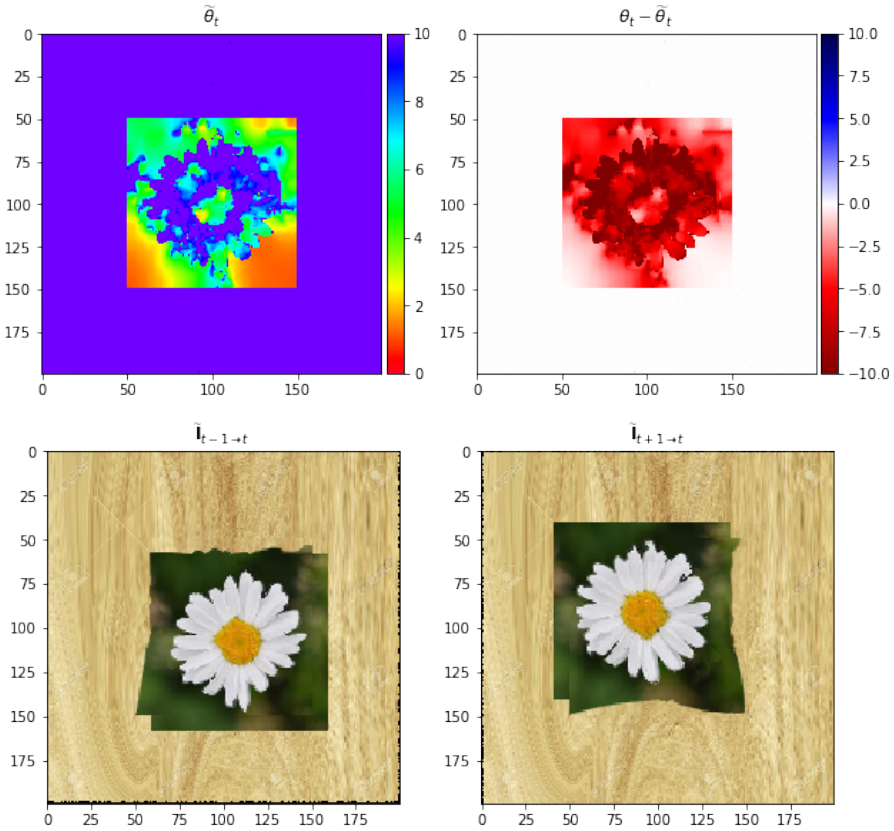


Figure 4.19: Result on Flower scene, with regular diffusion smooth loss. No ghost effect can be seen around the foreground depth, but it performs poorly on foreground depth.

4.7.3 Photometric distance

Since the beginning we worked with the simplest form of photometric distance, which is the absolute difference between color values. However, this assumes that images rightly projected have the exact same color. This is problematic as in reality some extrinsic scenes parameters can change. For example, light conditions can change and reflectance for non Lambertian surfaces makes the texture appear totally differently. Even camera parameters can change gradually such as white balance and auto-exposition. Some effects such as light changes can be assumed to change slowly and be neglected, some other effects such as auto-exposition when the camera goes out of a tunnel, or reflection patterns cannot be neglected. See an example figure 4.20. The following loss tries to address that problem by computing variations of correlation rather than absolute difference. This is particularly useful for changes of exposition.

Structural Similarity (SSIM)

As suggested subsequently by Zhao and Godart [Zha+15; GMB17], raw pixel difference can be coupled with structural similarity (SSIM) [Wan+04] maximization, in order to have a more natural quality measure. This has later been confirmed to yield good results

Figure 4.20: Example of change of luminosity inside a video, when entering the tunnel, the camera rises the exposition, and corresponding point have now different colors.

[GMAB18; Cas+19; MWA18; YS18; WB18]. SSIM (Eq. 4.27) can be decomposed into the product of three functions, each measuring a specific characteristic, detailed Eq. 4.28. l for luminance, c for contrast and s for structure. α, β, and γ are their relative weights. Usually $\alpha = \beta = \gamma = 1$, for simplicity and readability, but next notations will still hold for different values. C_1, C_2 and C_3 are arbitrary values, made to avoid edge cases, where denominators are too low. μ and σ are the local mean and variance operators, estimated by convolving the images $\boldsymbol{X}$ and $\boldsymbol{Y}$ with Gaussian kernels $\boldsymbol{G}$ of a given size (usually 3×3) Eq. 4.31.

$$\text{SSIM}(\boldsymbol{X}, \boldsymbol{Y}) = l(\boldsymbol{X}, \boldsymbol{Y})^{\alpha} c(\boldsymbol{X}, \boldsymbol{Y})^{\beta} s(\boldsymbol{X}, \boldsymbol{Y})^{\gamma} \tag{4.27}$$

$$l(\boldsymbol{X}, \boldsymbol{Y}) = \frac{2\mu_X \mu_Y + C_1}{\mu_X^2 + \mu_Y^2 + C_1} \tag{4.28}$$

$$c(\boldsymbol{X}, \boldsymbol{Y}) = \frac{2\sigma_X \sigma_Y + C_2}{\sigma_X^2 + \sigma_Y^2 + C_2} \tag{4.29}$$

$$s(\boldsymbol{X}, \boldsymbol{Y}) = \frac{\sigma_{XY} + C_3}{\sigma_X \sigma_Y + C_3} \tag{4.30}$$

$$\mu_X = \boldsymbol{G} * \boldsymbol{X} \tag{4.31}$$

$$\sigma_X^2 = \boldsymbol{G} * \boldsymbol{X}^2 - \mu_X^2 \tag{4.32}$$

$$\sigma_{XY} = \boldsymbol{G} * \boldsymbol{XY} - \mu_X \mu_Y = \boldsymbol{G} * (\boldsymbol{X} - \mu_X)(\boldsymbol{Y} - \mu_Y) \tag{4.33}$$

The SSIM function is a quality measure, which maximum value is 1, and minimum is -1. l and c have values within $[0, 1]$ and s has values within $[-1, 1]$. In order to avoid too much computation, C_3 is usually set so that $C_3 = \frac{C_2}{2}$ which simplifies SSIM equation into Eq. 4.34.

$$\text{SSIM}(\boldsymbol{X}, \boldsymbol{Y}) = \frac{(2\mu_X \mu_{\boldsymbol{Y}} + C_1)(2\sigma_{XY} + C_2)}{(\mu_X^2 + \mu_Y^2 + C_1)(\sigma_X^2 + \sigma_Y^2 + C_2)} \tag{4.34}$$

Nevertheless, general equation 4.27 can help grasp what values are actually measured and thus what is optimized when trying to maximize this measure by gradient ascent. An important point to note is that these functions are rather independent of one another. Namely, the function l is only a function of means μ_X and μ_Y, while being invariant to variance and correlation, the function c is a function of variance and s (roughly) a function of correlation when C_2 is negligible compared to $\sigma_X \sigma_Y$.

$$\text{SSIM}(\boldsymbol{X}, \boldsymbol{Y}) = l(\mu_Y, \mu_X)c(\sigma_X, \sigma_Y)s(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}})$$

$$\widehat{\boldsymbol{X}} = \frac{\boldsymbol{X} - \mu_X}{\sigma_X}$$

If we try to differentiate SSIM relative to X, we get

$$\nabla \text{SSIM} = cs\nabla l + ls\nabla c + lc\nabla s$$

The way to understand this is when trying to maximize SSIM, gradients of a function are dampened by the others. For example, when mean values don't match, l will be very low, and thus functions c and s won't be maximized. It can be problematic if at the beginning of our optimization the images patches are too dissimilar, for example when the optical flow is too high. As a consequence, it is a good idea to mix it with the more traditional L_1 distance, at least at the beginning. However, this loss adds some robustness: for a problematic sample with changing exposition (e.g, when exiting a tunnel), the difference of luminance and contrast lowers the SSIM gradient, and thus optimization is less effective on this sample. This loss is robust to changes of luminance in the sense that it filters them out of the optimization. In most works already mentioned [GMB17] [WB18] [YS18], [MWA18], [Cas+19], the photometric is then a weighted sum of the L1 loss and DSSIM (the opposite version of SSIM, which needs to be lowered)

$$\mathcal{L}_p(\widetilde{\boldsymbol{I}}, \boldsymbol{I}) = (1 - \alpha) \sum_i \frac{\|\widetilde{\boldsymbol{I}} - \boldsymbol{I}\|_1}{|\Omega|} + \alpha \frac{1 - \text{SSIM}(\widetilde{\boldsymbol{I}}, \boldsymbol{I})}{2}$$

It must also be noted that correlation is in fact the exact opposite of mean square error of normalized vectors (up to a constant value), as quickly shown Eq. 4.35. Likewise, the functions l and c can be interpreted as mean square errors, dampened by the denominator (because its value does not change much). This makes sense when considering the initial justification for SSIM, inspired by the human visual system, which supposedly has a relative scale. However, for us, this may not be the best assumption, because the higher the values $\boldsymbol{X}$ and $\boldsymbol{Y}$ we will have to match, the lower the gradient will be, inducing a vanishing gradient problem.

$$\begin{aligned} s(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}) &= \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \\ &= \boldsymbol{G} * \widehat{\boldsymbol{X}}\widehat{\boldsymbol{Y}} \\ &= \frac{1}{2}\boldsymbol{G} * (\widehat{\boldsymbol{X}}^2 + \widehat{\boldsymbol{Y}}^2 - (\widehat{\boldsymbol{X}} - \widehat{\boldsymbol{Y}})^2) \\ &= 1 - \frac{1}{2}\boldsymbol{G}(\widehat{\boldsymbol{X}} - \widehat{\boldsymbol{Y}})^2 \end{aligned} \tag{4.35}$$

Additionally, figure 4.21 shows the profile of a curve $x \mapsto \frac{2xy+C}{x^2+y^2+C}$. It appears obvious that optimization is impossible if x and y don't have the same sign. It must not be forgotten especially for the function l since in most deep learning training workflow, images are normalized before being fed to the network. Our advice if one wants to use SSIM is then to normalize images for the network, but not for the photometric loss.

For all of these reasons, SSIM seems suited to reach a better optimum, and benefit from being a widespread image quality metric, but can probably be improved in terms of gradient based optimization. In fact, in all the works that use it for image reconstruction problems, we have not seen other justification than the fact that it yields better results.

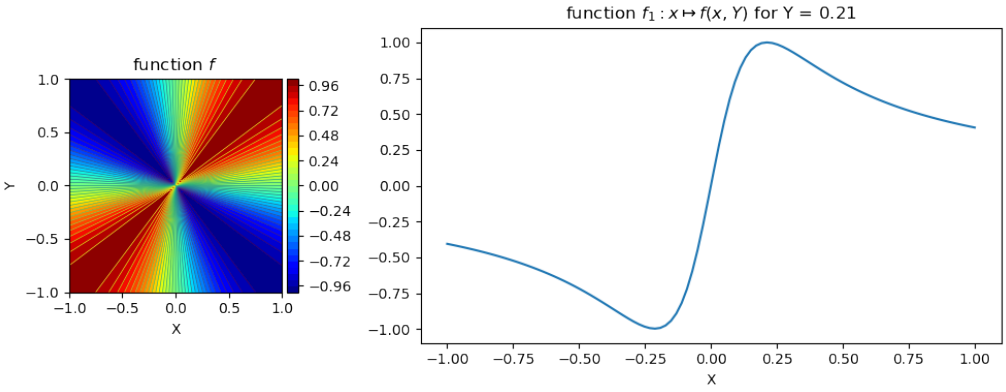


Figure 4.21: graph of the function $f : (x, y) \mapsto \frac{2xy+C}{x^2+y^2+C}$ used for l and c in SSIM (here, C is 0.01^2). This function is only convex around its maximum, which make its gradient based optimization difficult

4.8 Conclusions on literature review for unsupervised depth training

In our toy problem, we have been able to identify and inspect different optimization methods for image reconstruction. Unfortunately, it turns out most of them are only motivated by heuristics, and thus can be improved by physical and theoretical considerations. It appears that the end goals are well defined: a smooth inverse depth map, with possible discontinuities. However, the loss gradients are not heaviliy engineered, as it can be seen e.g. for TVV Loss and SSIM distance.

This tells us that this problematic is still in its "proof of concept" phase. The same way a simple network like VGG [SZ14] was improved into Inception [Sze+15], ResNet [He+16], and finally DenseNet [Hua+17], we hope photometric distances and smooth losses will be refined to more robust and stable equivalent that could potentially lead to a true unsupervised learning, without having an annotated validation set for early stopping.

Chapter 5

Unsupervised DepthNet

Contents

5.1 Shortcomings of SFMLearner 95
5.2 Workflow presentation 96
5.2.1 Pose estimation 97
5.2.2 Frame stabilization 97
5.2.3 Depth computing and pose normalization 98
5.2.4 Loss functions 98
5.3 Training datasets 98
5.4 Experiments 99
5.4.1 Qualitative results 99
5.4.2 Implementation details 99
5.4.3 Testing methodology for DepthNet 99
5.4.4 Quantitative results 100
5.4.5 Domain adaptation results 105
5.5 Conclusion on unsupervised DepthNet 105

This chapter presents our proposition for an unsupervised training of DepthNet, with robustness in mind. From the study made chapter 4 on unsupervised depth learning, we propose to discuss and adapt the work SFMLearner from Zhou et al. [Zho+17] and its variations in order to perform fine-tuning on real videos for our DepthNet network without ground-truth.

This chapter has been the subject to the following publication

[Pin+18]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. “Learning structure-from-motion from motion”. In: *ECCV GMDL Workshop*. Munich, Germany, Sept. 2018. URL: https://hal.archives-ouvertes.fr/hal-01995833.

5.1 Shortcomings of SFMLearner

As a reminder, the SFMLearner, presented section 4.3.1 trains a neural network (DispNet) to output inverse depth of an image, along with another network (PoseNet) to output frame poses of a particular sequence.

Whereas PoseNet takes all the images as input, DispNet only takes one picture $\boldsymbol{I}_t$. As such, this network is doing inverse depth from context and appearance. It should also be noted that KITTI [Gei+13] was the main dataset for this particular unsupervised learning task. Unfortunately, this dataset, which is only representative of car movements,

is not really a true 6 degrees of freedom dataset, and lacks appearance diversity. As a consequence, depth from context is a very good strategy here. Indeed, it is mentioned by Zhou et al. [Zho+17] that feeding multiple frames to a network did not yield better results in their training workflow on KITTI dataset.

As discussed chapter 2, in the context of drone videos, the dataset can present camera movements with only 3 degrees of freedom since we don't have rotation, but flying cameras will present a huge variety of contexts and appearances, with different kinds of altitude and orientation, in addition to different ground profiles on which a car could not drive.

Another important point to note is the fact the scaling factor uncertainty is never explicitly solved, while DepthNet solves it by assuming constant displacement magnitude (as explained section 3.5.1), which is then compared against actual displacement magnitude. Here, no assumption is done on PoseNet translation magnitude, and the only supervision comes from the resulting optical flow and its corresponding warping, which is proportionate to both PoseNet displacement magnitude and disparity. As such, even if the two networks converge consistently so that the optical flow is well estimated, the depth and poses alone are only known up to a scale factor compared to ground truth.

Following discussion from section 2.3.1, we will display typical scale invariant quality measures for information purposes only. The actual measure we will use to evaluate and compare our solution asks for both translation estimation and depth estimation, to make the scale factor solvable with translation magnitude measured from dedicated sensors. The two different quality evaluations will be denoted in tables 5.1, 5.2 and 5.3 by the scale factor column. When using standard relative depth measurement, the indication GT (for *Ground Truth*) is used, when using translation magnitude, the letter P (for *Pose*) is used.

5.2 Workflow presentation

Inspired from [Zho+17], we propose a framework for training a network similar to DepthNet, but from unlabeled video sequences. The approach is slightly different from SfmLearner since our network takes two stabilized images for input instead of only one. The approach can be decomposed into three tasks, as illustrated by Fig. 5.1:

- From a certain sequence of frames $(\boldsymbol{I}_i)_{0\leq i<N}$, randomly choose one target frame I_t and one reference frame $\boldsymbol{I}_r$, forming a pair to feed to DepthNet.
- For each $i \in [\![0, N[\![$, estimate pose $\boldsymbol{T}_{t\to i} = (\mathbf{R}_i, \boldsymbol{t}_i)$ of each frame $\boldsymbol{I}_i$ relative to the target frame $\boldsymbol{I}_t$. Detailed section 5.2.1.
- Compensate rotation of reference frame $\boldsymbol{I}_r$ to $\boldsymbol{I}_r^{\text{STAB}}$ before feeding it to DepthNet, leading to the same situation as original DepthNet supervised training. Detailed section 5.2.2.
- Compute the unnormalized depth map $\widetilde{\theta}_t$. DepthNet$(\boldsymbol{I}_r^{\text{STAB}}, \boldsymbol{I}_t) = \widetilde{\theta}(\boldsymbol{I}_r^{\text{STAB}}, \boldsymbol{I}_t)$
- Normalize the translation to constrain it so that the displacement magnitude t_r with respect to $\boldsymbol{I}_r$, is always the same throughout the training. This point is very important, in order to guarantee the equivariance between depth and motion, imposed by the original DepthNet training procedure that makes network rely mostly on motion and not on appearance. Detailed section 5.2.3.
- As the problem is now made equivalent to the one used in [Zho+17], perform a photometric reprojection of $\boldsymbol{I}_t$ to every other frame $\boldsymbol{I}_i$ to $\widetilde{\boldsymbol{I}}_{i\to t}$, thanks to depth $\widetilde{\theta}_t$ of $\boldsymbol{I}_t$ and poses $\boldsymbol{T}_{t\to i}$ computed before. The resulting frame $\widetilde{\boldsymbol{I}}_t$ is then compared against $\boldsymbol{I}_t$ to compute loss from photometric dissimilarity. Detailed section 5.2.4.

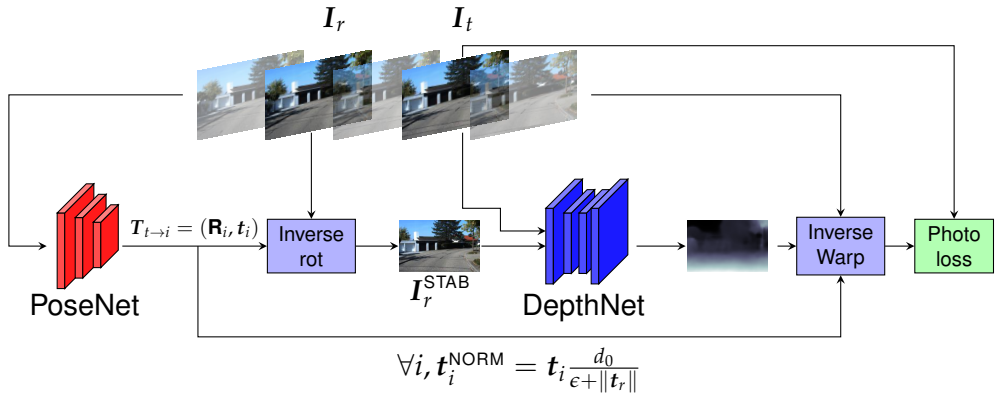


Figure 5.1: General workflow architecture. *target* and *reference* indices (t and r) are chosen randomly for each sequence (even within the same batch); output transformations of PoseNet are compensated so that $\widehat{T}_{t\to t}$ is identity. d_0 is a fixed nominal displacement.

Our algorithm, although relying on very little calibration, needs to get consistent focal length. This is due to the frame difference being dependent on focal length. However, this problem is easily avoided when training on sequences coming from the same camera. Also, as shown by Eq. 4.2, camera intrinsic matrix **K** needs to be known to compute warping and subsequent photometric reprojection loss properly. In practice, assuming optical center in the center of the focal plane worked for all our tests; this is corroborated by tests done by [MWA18] where they used uncalibrated camera, only knowing approximate focal length.

Finally, this algorithm can be simplified for stabilized videos, for which no rotation needs to be estimated, and no stabilization needs to be done before feeding I_r to the network.

5.2.1 Pose estimation

PoseNet, as initially introduced by Zhou et al [Zho+17] is a classic fully convolutional neural network that outputs 6 degrees of freedom (or 3 in the case of stabilized videos) transformation pose descriptor for each frame. Output poses are initially relative to the last frame, and then compensated to be relative to the pose of the target frame. This way, PoseNet output is not dependent on the index of the target frame. Besides, computing by default with respect to the last frame makes the inference much more straightforward, as in real condition, target frame on which depth is computed should be the last of the sequence, to reduce latency.

5.2.2 Frame stabilization

For the general case with 6 degrees of freedom, in order to cancel rotation between target and reference frame, we can apply a warping using rotation estimation from PoseNet. When considering a transformation without translation, Eq. 4.2 no longer depends on the depth of each pixel, and becomes Eq. 5.1. As such, we can warp the frame to stabilize it using orientation estimation from PoseNet, before computing any depth.

$$p_t^r = \mathbf{K}\mathbf{R}_r\mathbf{K}^{-1}p_t \tag{5.1}$$

As mentioned earlier, UAV footages are either stabilized or with a reliable estimated orientation from inertial sensors. This information can be easily leveraged in our training workflow to supervise pose rotation, giving in the end only translation to estimate

to PoseNet. In addition, when running in inference, no pose estimation is needed as DepthNet does not depend on translation estimation, and PoseNet is not used.

5.2.3 Depth computing and pose normalization

For depth to be provided assuming a constant displacement magnitude, the pose of the reference frame must be normalized to correspond to that magnitude.

As such, to get consistent poses throughout the whole sequence, we apply the same normalization ratio, as shown in Fig. 5.1. The resulting magnitude is thus only known to be scaled to d_0 between $\boldsymbol{I}_t$ and $\boldsymbol{I}_r$. Every other translation might be of different magnitude.

$$\forall i, \boldsymbol{t}_i^{\text{NORM}} = t_i \frac{d_0}{\epsilon + \|\boldsymbol{t}_r\|} \tag{5.2}$$

The main drawback of normalizing translations is the lack of guarantee about absolute output values. Since we only consider translations relatively to the reference, translations are estimated up to a scale factor that could be - when they are very large - leading to potential errors for rotation estimation, or - when they are very close to 0 - leading to float overflow problems. To overcome these possible issues, along with classic L_2 regularization to avoid high values, we add a constant value ϵ to the denominator. The normalization is then valid only when $\epsilon \ll \|\boldsymbol{t}_r\|$.

5.2.4 Loss functions

Following what was discussed on chapter 4, our training loss will be composed of a photometric loss, and a smooth loss: we chose the gradient diffusion loss with optimal parameters κ and λ defined with the hyper-parameter search on the toy problem presented section 4.6.2.

Our photometric loss $\mathcal{L}_p$ is a mixture of mean L1 distance and DSSIM, α being an empirical weight set to 0.15.

$$\mathcal{L}_p = \sum_i (1-\alpha) \frac{\|\widetilde{\boldsymbol{I}}_{i\to t} - \boldsymbol{I}_t\|_1}{|\Omega|} + \alpha \frac{1 - \text{SSIM}(\widetilde{\boldsymbol{I}}_{i\to t}, \boldsymbol{I}_t)}{2} \tag{5.3}$$

$$\mathcal{L}_s = \lambda \mathcal{L}_{gdiff}(\xi, \boldsymbol{I}_t, \kappa) = \frac{\lambda}{|\Omega|} \sum_{p\in\Omega} \exp\left(-\frac{\|\nabla \boldsymbol{I}_t\|^2}{\kappa^2}\right) \times (\Delta\xi)^2 \tag{5.4}$$

For $\mathcal{L}_s$, $\boldsymbol{I}_t$ is downscaled to fit ξ resolution.

Finally, we apply this loss to multiple scales s of DepthNet outputs, multiplied by a factor giving more importance to high resolution. In accordance with what was discussed section 4.7.1, the loss $\mathcal{L}_p^s$ is obtained with full scale target image and warped image from upscaled output of DepthNet.

Our final loss becomes

$$\mathcal{L} = \sum_s \frac{1}{2^s} \left(\mathcal{L}_p^s + \mathcal{L}_s^s\right) \tag{5.5}$$

5.3 Training datasets

Our experiments were made on three different datasets. KITTI [Gei+13] is one of the most well known datasets for training and evaluating algorithms on multiple computer vision tasks such as odometry, optical flow or disparity. It features stereo vision, LiDAR depth measures and GPS / RTK coupled with IMU for camera poses. During training

we only used monocular frames and IMU values when supervising with orientation. We used LiDAR for evaluation. We applied the same training/validation/test split as [Zho+17]: about 40k frames for training and 4k for evaluation. We also discarded the whole set scenes containing the 697 test frames from the Eigen [EPF14] split. We also constructed a filtered test set with the same frames as Eigen, but discarding 69 frames whose GPS position uncertainty was above 1 m. This set was used when displacement data was needed.

We also conducted experiments on StillBox, the dataset used in supervised DepthNet section 3.4.1, in which we added random rotations (i.e. we draw an initial rotation speed that remains the same through the sequence). As this dataset was initially developed to have depth independent on context, we can already predict that performance of single frame depth network like SFMLearner [Zho+17] will not be good. The dataset contains 1500 training scenes and 100 test scenes, ie 30k training frames and 2k validation frames.

Finally, we trained our network on a very small dataset of UAV videos, taken from the same camera the same day. We used a Bebop2 drone, with 30 frame per seconds videos, and flew over a small area of about one hectare for 15 minutes. Some of the flights have low displacement sequences and even hovering states where the drone is completely motionless. The training set contains around 14k frames while the test set is a sequence of 400 frames. This dataset is not annotated, and only subjective evaluation can be done.

5.4 Experiments

5.4.1 Qualitative results

Figure 5.2 presents a qualitative comparison between SFMLearner and Unsupervised DepthNet, on KITTI and StillBox. If results are comparable on KITTI, DepthNet is much sharper on StillBox.

5.4.2 Implementation details

As for supervised training chapter 4, we used PyTorch [Pas+17] for all tests and trainings. We used Adam optimizer [KB14] with learning rate of 2×10^{-4} and $\beta_1 = 0.9$, $\beta_2 = 0.999$. The full source code is available on *Github*[1]

5.4.3 Testing methodology for DepthNet

To stabilize the input of the network, we take a sequence of the same length as PoseNet input (here, 5 frames), with the goal of computing depth of the last frame. When training was done with orientation supervision, we stabilized the frames before feeding them to DepthNet, without the help of PoseNet.

Besides, as DepthNet output is a function of apparent movement, we have several options for the frame pairs we can use to compute depth. Our method relies on taking a pair for which the mean output of DepthNet is the closest of an arbitrary chosen value set to $50m$, regardless of the dataset. The output is then rescaled with the ratio of nominal displacement. A more sophisticated method for depth inference with stabilized frames will be presented on chapter 6.

[1] `https://github.com/ClementPinard/unsupervised-depthnet`, retrieved 07/24/2019

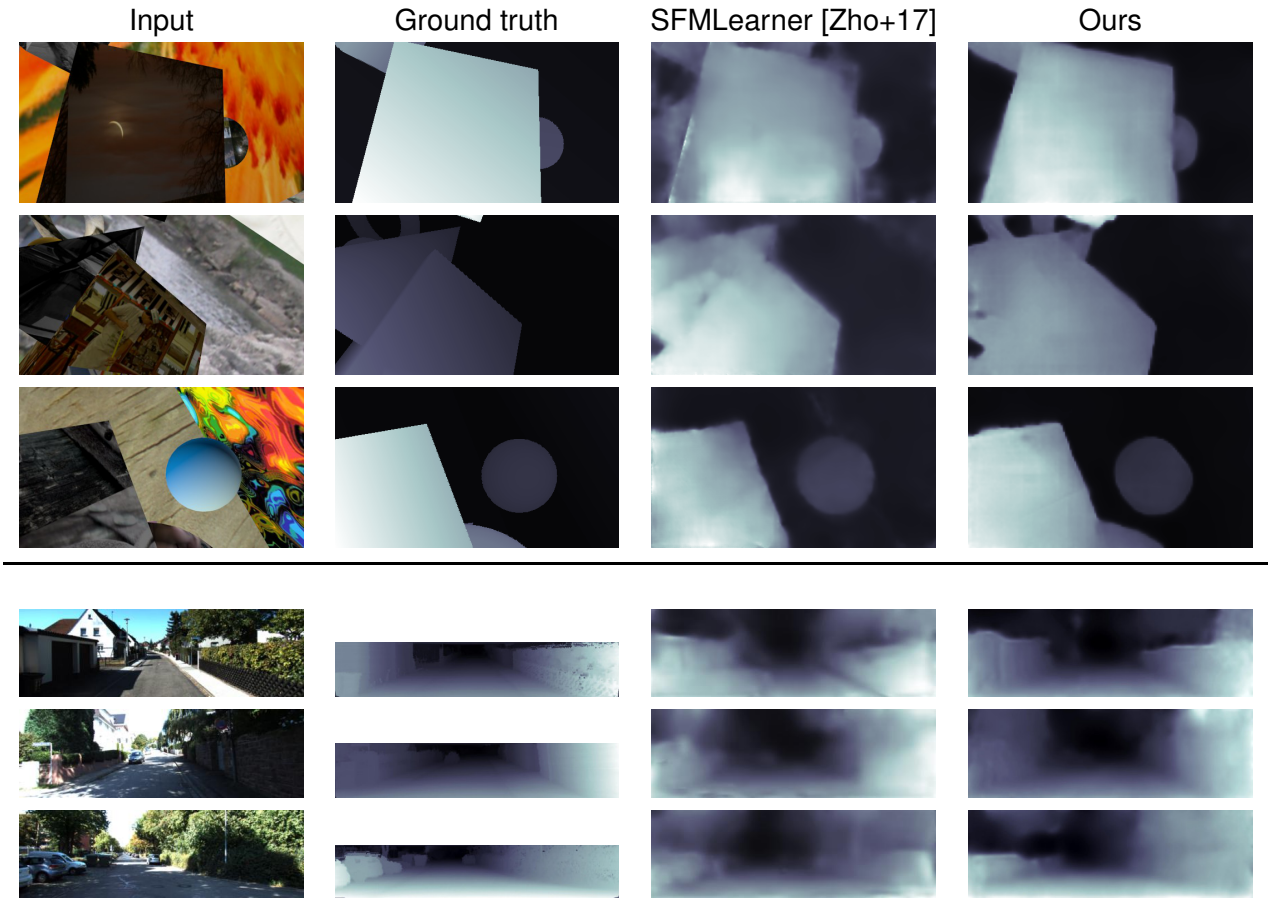


Figure 5.2: Comparison between our method and SFMLearner. [Zho+17] on StillBox and KITTI [Gei+13].

5.4.4 Quantitative results

In addition to using standard measurements from [EPF14], our goal is to measure how well a network would perform in real conditions. As stated in Section 2.3.1, depth map scale factor must be determined from reasonable external data and not from explicit depth ground truth.

We thus compare our solution to SFMLearner [Zho+17] where the output is multiplied by the ratio between estimated displacement from PoseNet and actual values. For KITTI, displacement is determined by GPS RTK, but as we only need magnitude, speed from wheels would have been sufficient.

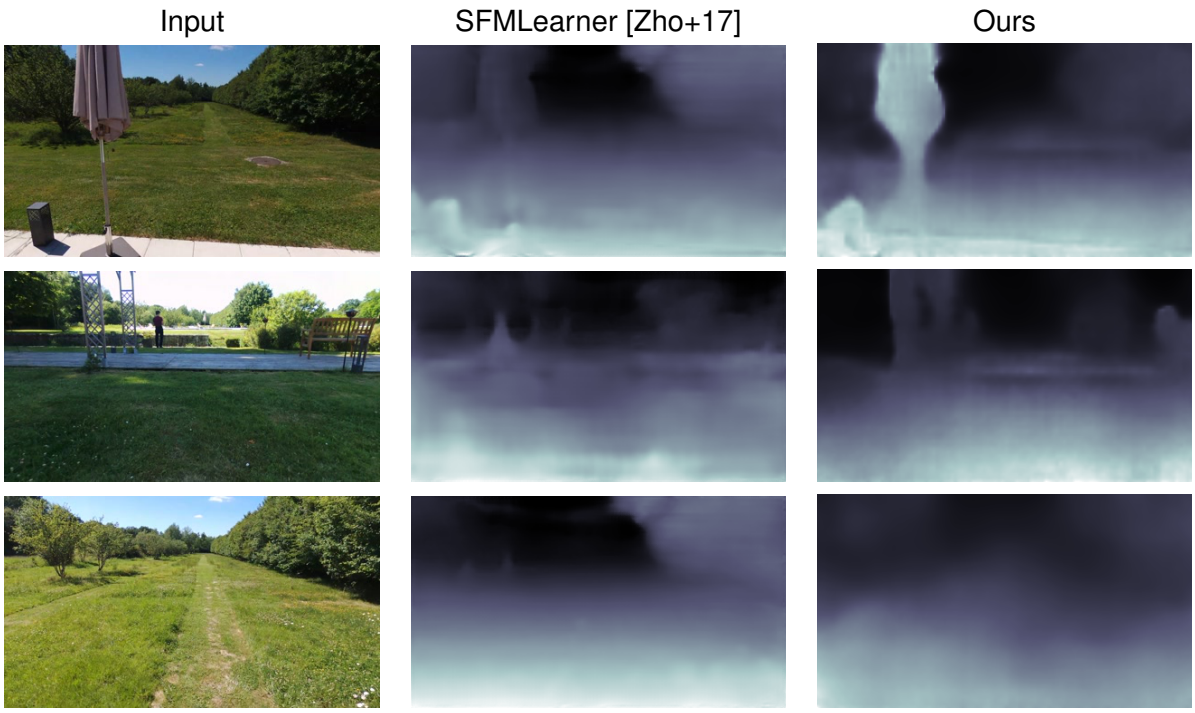


Figure 5.3: Subjective comparison of disparity maps between SFMLearner [Zho+17] and our method on a small UAV dataset.

Figure 5.4: Some failure cases of our method on KITTI. First column is a detail of a larger image. The foreground car is moving forward and it's detected as far away, while the background car is moving toward us and is detected as close. Second column is a poorly textured road.

Method	training set	scale factor	supervision	MAE	MRE	MLE	SAE	SLE	$P_{1.25}$	$P_{1.25^2}$	$P_{1.25^3}$
Eigen et al. [EPF14] Fine	K	GT	D	-	0.203	-	6.307	0.282	0.702	0.890	0.958
DORN [Fu+18]	K	GT	D	-	0.072	-	2.727	0.120	0.932	0.984	0.994
SFMLearner [Zho+17]	K	GT	✗	-	0.183	-	6.709	0.270	0.734	0.902	0.959
SFMLearner $*$	K	GT	✗	3.455	0.185	0.191	6.410	0.269	0.714	0.899	0.962
DDVO [WB18]	K	GT	✗	-	0.148	-	5.496	0.226	0.812	0.938	0.975
Casser et al. [Cas+19]	K	GT	✗	-	**0.141**	-	**5.291**	**0.215**	**0.816**	**0.945**	**0.979**
Supervised DepthNet	S	P	D + O	7.872	0.650	0.446	10.528	0.546	0.356	0.613	0.775
SFMLearner $*$	K	P	✗	4.629	0.296	0.287	7.215	0.352	0.542	0.817	0.918
Unsupervised DepthNet	K	P	✗	4.669	0.322	0.262	7.967	0.356	0.624	0.838	0.916
Unsupervised DepthNet	S $\rightarrow$ K	P	D+O $\rightarrow$ ✗	4.078	0.268	0.222	7.048	0.304	0.683	**0.883**	**0.944**
Unsupervised DepthNet	K	P	O	4.485	0.301	0.251	7.504	0.335	0.624	0.851	0.927
Unsupervised DepthNet	S $\rightarrow$ K	P	D+O $\rightarrow$ O	**4.001**	**0.262**	**0.221**	**6.872**	**0.303**	**0.687**	**0.883**	0.943

$*$ this is our reimplementation, available on `https://github.com/ClementPinard/SfmLearner-Pytorch` (accessed 02/12/2019)

Table 5.1: Quantitative tests on KITTI[Gei+13] Eigen split [EPF14]. Measures are the same as in Eigen et al. [EPF14]. For blue measures, lower is better, for red measures, higher is better. For training, K is the KITTI dataset [Gei+13], S is the Still Box dataset. For scale factor, GT is ground truth, P is pose. When scale was determined with pose, we discarded frames where GPS uncertainty was greater than $1m$. For supervision, D is depth and O is orientation. $\rightarrow$ denotes fine tuning.

Method	scale factor	supervision	Occlusion mapper	MAE	MRE	MLE	SAE	SLE	$P_{1.25}$	$P_{1.25^2}$	$P_{1.25^3}$
Supervised DepthNet	P	D + O	✗	**3.822**	0.224	0.229	**6.864**	0.446	0.718	0.856	0.927
SFMLearner $*$	GT	✗	✗	11.429	0.593	0.633	17.173	0.461	0.479	0.656	0.756
SFMLearner $*$	P	✗	✗	14.324	0.816	0.461	19.447	0.726	0.302	0.542	0.696
Unsupervised DepthNet	P	✗	✗	10.356	0.437	0.387	10.236	0.516	0.476	0.697	0.822
Unsupervised DepthNet	P	O	✗	5.248	0.259	0.232	9.600	0.380	0.709	0.887	0.928
Unsupervised DepthNet	P	✗	✓	10.412	0.435	0.383	15.450	0.510	0.503	0.716	0.820
Unsupervised DepthNet	P	O	✓	5.060	**0.212**	**0.206**	9.262	**0.333**	**0.729**	**0.899**	**0.943**

$*$ this is our reimplementation, available on `https://github.com/ClementPinard/SfmLearner-Pytorch` (accessed 02/12/2019)

Table 5.2: Quantitative tests on StillBox, no pretraining has been done. The supervised DepthNet is here to give an hint on a theoretical limit since it uses the same network, but with depth supervision

Table 5.1 presents quantitative results compared to prior works on KITTI dataset. Prior works are divided into three categories. The first one with Eigen et al. [EPF14] and DORN from Fu et al. [Fu+18] shows the single frame depth network trained with supervision, DORN being the current state of the art. It serves at giving an idea of theoretical limit. The second one with SFMLearner [Zho+17] and DDVO from Wang et al. [WB18] shows unsupervised methods where moving scenes are not explicitly considered. Last category with Casser et al [Cas+19] shows the state of the art on unsupervised training of depth with monocular camera on KITTI, which takes care of moving scenes with the help of an object detection network. These different works show that KITTI is indeed a dataset with non negligible moving scenes, which will be problematic for our solution, and that neural networks are extremely good at inferring depth from a single image on a dataset like KITTI.

We tried 5 different versions of our network. The first one is the exact same as supervised DepthNet, only trained on StillBox. It serves as a baseline purpose, without fine-tuning. The other four configurations are training from scratch or fine-tuning from StillBox, and training with orientation supervision or not. For our 5 versions we used occlusion module detection, our tests determined that this module had no influence on KITTI. This is probably due to the fact that contrary to our Stillbox dataset and typical drone videos, objects are never extremely close to the camera, and thus few occlusions happen and when they do, they induce an error that is neglectible compared to moving objects.

As we might expect, on KITTI our method fails to converge as well as single image methods using classic relative depth quality measurement. However, when scale factor is determined from poses, we match the performance of the adapted method from [Zho+17]. It can also be noted that fine-tuning provides a better starting point for our network, and that when available on a training set, orientation supervision is very advantageous.

When trying to train a depth network with stabilized videos, it is then strongly recommended to do a first supervised training on a synthetic dataset such as StillBox.

Some failed test cases can be seen on Fig.5.4. The main sources of error are moving objects and poorly textured areas (especially concrete roads), even though we applied depth smooth geometric loss. Our attempt at explaining this failure is the large optical flow value on a low textured area, meaning matching spatial structures is difficult. However, as KITTI acquisition rate is only 10 fps, we believe this problem would be less common on regular cameras, with typical rates of 30 fps or higher.

Table 5.2 presents results on the updated Still Box dataset. SFMLearner [Zho+17] performs surprisingly well given the theoretical lack of visual context in this dataset. However, our method performs better, whether from orientation supervision or completely unsupervised. We also compare it to supervised DepthNet. This can be considered a theoretical limit for training DepthNet on Still Box since it is completely supervised, and our unsupervised training method is very close to it, indicating the good convergence of our model and thus the validity of our training algorithm and loss design.

Note on Hyperparameter Search

As already discussed in chapter 4, the key idea behind our choice of regularization hyper parameters, and especially the smooth loss we chose, is domain-independence. That's why we chose gradient diffusion loss and not diffusion loss in the first place although it did perform better on the toy problem presented at section 4.6.2, because the optimum was more evenly distributed than diffusion.

For reference, in the context of unsupervised training of depth for KITTI, there has been a consensus on TV Loss for the best results. This loss is then particularly suited for

the in-car environment, but not necessarily for drone videos.

If we wanted to test the domain independance hypothesis, we would have needed to test unsupervised training with every possible loss, with every possible hyperparameter. This was possible on a simple toy problem as a single training only required one minute, but is unfortunately much harder to do on a true unsupervised training task, where a single training takes around 10 hours. It might be a future work to consider with efficient sampling techniques such as Bayesian optimization [WHD18].

5.4.5 Domain adaptation results

To test our method on our small UAV dataset, we first did a training on Still Box, then an unsupervised fine tuning. Likewise, when using SFMLearner method [Zho+17], we pretrained on KITTI before fine tuning on our video.

Fig.5.3 compares SFMLearner [Zho+17] and our method on some test frames of our small UAV dataset. Our method shows much better domain adaptation when fine tuning in a few-shot learning fashion. Especially for foreground objects, as SFMLearner [Zho+17] blends it with the trees near the horizon, which is very problematic for navigation. A more thorough qualitative comparison can be viewed in video [1].

Table 5.3 and figure 5.5 compare domain robustness without any training: we tried inference on an upside-down KITTI test set, with ground up and sky down, and our method performs much better than SFMLearner [Zho+17], which is completely lost and performs worse than inferring a constant plane. However, contrary to Supervised DepthNet which performs as well on the reverse version of the test set, our method is not as performing as with regular KITTI (not reversed). This shows that our network may have learned to infer depth from both motion and context, which can be considered as a compromise between SFMLearner relying only on context and Supervised DepthNet relying only on motion.

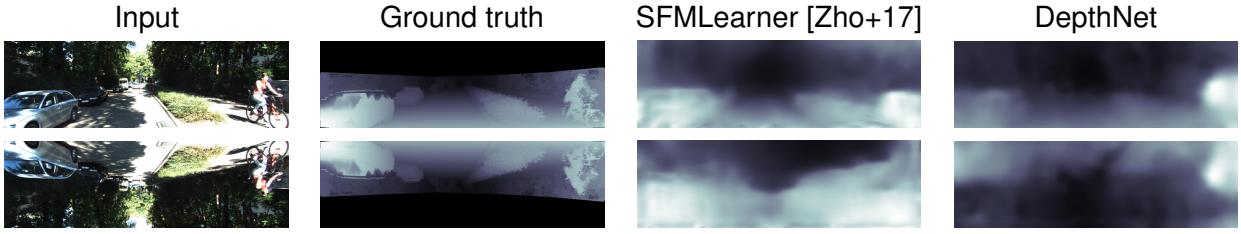


Figure 5.5: Comparison between our method and SFMLearner [Zho+17] on a normal example from KITTI [Gei+13] and its upside-down variation.

5.5 Conclusion on unsupervised DepthNet

We have presented a novel method for unsupervised depth learning, using not only depth from context but also from motion. This method leverages the context of stabilized videos, which is a midway between common use cases of stereo rigs and unconstrained ego-motion. As such, our algorithm provides a solution with embedded deployment in mind, especially for UAVs navigation, and requires only video and inertial data to be used.

Our method is also much more robust to domain changes, which is an important issue when dealing with deployment in large scale consumer electronics on which it is impossible to predict all possible contexts and situations, and our method outperforms single frame systems on unusual scenes.

[1] https://stillbox.ensta.fr/demo_depthnet.mp4

Method	training set	scale factor	supervision	MAE	MRE	MLE	SAE	SLE	$P_{1.25}$	$P_{1.25^2}$	$P_{1.25^3}$
Supervised DepthNet	S	P	D + O	**7.716**	0.637	0.443	**10.499**	0.548	0.359	0.614	0.774
Constant Plane	-	GT	-	7.547	0.458	0.476	12.124	0.600	0.296	0.547	0.752
SFMLearner [Zho+17] $*$	K	GT	-	8.433	0.551	0.551	12.469	0.680	0.255	0.482	0.668
SFMLearner [Zho+17] $*$	K	P	-	16.085	1.511	0.790	19.677	0.927	0.176	0.342	0.494
Unsupervised DepthNet	S $\rightarrow$ K	P	-	7.759	0.596	0.442	11.250	0.555	0.370	0.628	0.784
Unsupervised DepthNet	S $\rightarrow$ K	P	O	7.731	**0.604**	**0.414**	11.152	**0.536**	**0.424**	**0.673**	**0.807**

Table 5.3: Quantitative tests on upside-down KITTI [Gei+13]: sky is down and ground is up. No training has been done. Constant Plane outputs the same depth for every pixel

Chapter 6

DepthNet in the wild

Contents

6.1 Simplifying the scale factor determination by normalizing DepthNet . . . 107
6.2 Optimal frame shift determination . . . 108
6.3 Multiple shifts inference . . . 109
6.4 Training a specific depth range with a clamped DepthNet . . . 113
6.5 Our proof of concept . . . 113
6.5.1 Model Predictive Control . . . 113
6.5.2 Implementation details . . . 113
6.6 Conclusions for this chapter . . . 115

This chapter aims at studying the possibilities of using our network DepthNet to get real-time depth maps. More precisely, we consider a strategy to feed our network the optimal frame pairs for real-time inference.

This chapter has been subject to the following publication

[Pin+17b]: Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "Multi range Real-time depth inference from a monocular stabilized footage using a Fully Convolutional Neural Network". In: *European Conference on Mobile Robotics*. ENSTA ParisTech. Paris, France, Sept. 2017. URL: https://hal.archives-ouvertes.fr/hal-01587658

6.1 Simplifying the scale factor determination by normalizing DepthNet

We learned depth inference from a moving camera, assuming its velocity is always the same. As already discussed chapter 2, when running during flight, such a system can deduce the real depth map estimation $\widetilde{\theta}$ from the network output and the drone displacement, knowing that the nominal displacement was d_0 (here $0.3m$), and the temporal difference between the frames δt.

$$\widetilde{\theta}(t) = DepthNet(I_t, I_{t-\delta t}) \frac{d(t,\delta t)}{d_0}$$

$$d(t,\delta t) = \left\| \int_{t-\delta t}^{t} \boldsymbol{V}(\tau) d\tau \right\|$$

In the context of a unconstrained displacement magnitude, the actual correct interpretation of the output of DepthNet is a depth relative to the camera displacement and not an absolute one, so its value should be a dimensionless quantity. To reflect the necessity

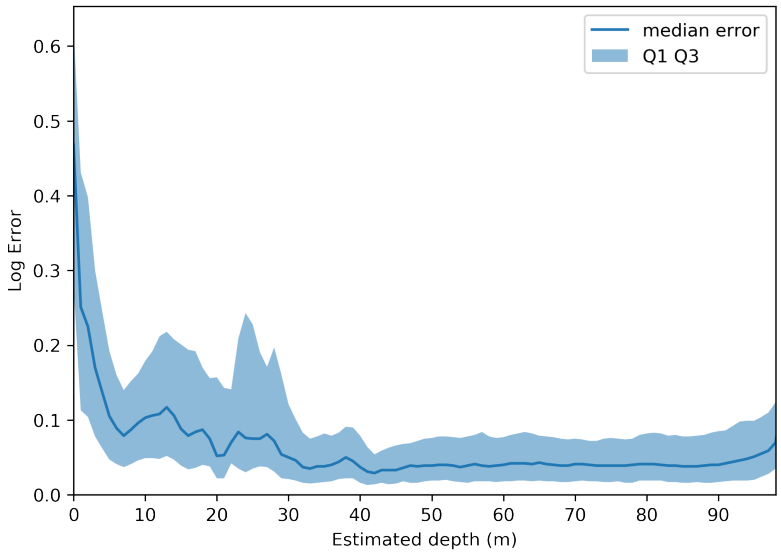


Figure 6.1: Estimation-wise log error of supervised DepthNet$_{64\to128\to256\to512}$ on StillBox.

of having a reference displacement magnitude, we introduce the function β that outputs a normalized version of the network output, which equals 1 when DepthNet reaches the maximal depth considered during training (here $100m$).

$$\begin{array}{rccc} \beta: & \mathbb{R}^{6|\Omega|} & \to & \mathbb{R}^{|\Omega|} \\ & \boldsymbol{I}_1, \boldsymbol{I}_2 & \mapsto & \frac{DepthNet(\boldsymbol{I}_1, \boldsymbol{I}_2)}{\theta_{max}} \end{array}$$

and a dimension-less parameter

$$\alpha = \frac{\theta_{max}}{d_0}$$

to estimate actual depth using the displacement magnitude d as the only distance related factor.

$$\widetilde{\theta}(t) = \alpha\beta(\boldsymbol{I}_t, \boldsymbol{I}_{t-\delta t})d(t, \delta t) \tag{6.1}$$

6.2 Optimal frame shift determination

Depending on the depth distribution of the ground-truth depth map, it may be useful to adjust frame shift δt. For example, when flying high above the ground with low speed, big structure detection and avoidance requires estimating distance values that are outside the typical range of any dedicated depth sensor. The logical strategy would then be to increase apparent movement by increasing the temporal shift between the frame pairs fed to DepthNet as inputs. More generally, one must provide inputs to DepthNet in order to ensure a well distributed depth output within its typical range.

This technique assumes that increasing temporal shift indeed increases apparent movement (and vice-versa for lowering temporal shift). This assumptions needs two conditions to be fulfilled between two frames:

1. The acceleration is small enough so that speed can be considered constant: this is the case for most smooth movements.

2. The depth is changing slowly enough to be considered constant: This is not always true, especially when distance traveled between two frames is comparable to measured depth. However, we assume this particular case to be rare.

The typical range is supposed to be a range of estimations where the error is minimal. Figure 6.1 shows the estimation-wise error for DepthNet512 on StillBox with a minimum between $40m$ and $90m$ (i.e. for $\beta \in [0.4, 0.9]$). This should not be confused with depth-wise error presented at section 3.10: a neural network giving always the same value $\widetilde{\theta}_c$ would have a depth-wise error of 0 at this particular distance. The inverse representation of estimation-wise error gives us more information on what to expect for θ compared to the estimated $\widetilde{\theta}$.

We then need to choose the optimal spatial displacement d_{opt} and corresponding temporal shift δt to minimize error on the next inference, by avoiding too low or too high values of β. We chose the optimal displacement magnitude as:

$$d_{opt} = d_0 \frac{\mathbb{E}(\beta)}{\beta_{mean}} = \frac{\mathbb{E}(\widetilde{\theta})}{\alpha \beta_{mean}}$$

With $\mathbb{E}(\widetilde{\theta})$ the mean of depth values and β_{mean} the optimal mean output of β, e.g. 0.5. δ is then computed numerically from displacement estimation to get the frame shift with the closest corresponding displacement magnitude possible. The mean $\mathbb{E}(\widetilde{\theta})$ is computed by discarding the extremal values, and especially maximal values (for example the depth values of the sky) since they are not relevant for the actual depth values we want to measure. It is worth noting that the median value of β could be used instead of mean. Using mean here actually paves the way for a more general algorithm presented section 6.3

In order to smooth potential noise in speed estimation that would result in a bad displacement magnitude estimation, we smooth out the target displacement over time. More specifically, we perform an exponential moving average (EMA) with an arbitrary factor λ

$$d_{opt} \leftarrow \lambda d_{opt} + (1 - \lambda) \frac{\mathbb{E}(\widetilde{\theta})}{\alpha \beta_{mean}} \tag{6.2}$$

The main limitation of this optimal frame shift determination is the multi-modality of the depth distribution of a particular scene. In this case, computing mean of DepthNet output might not be representative of any output value. A representative case would be a scene with bimodal depth values at $10m$ and $100m$ (see an example figure 6.2): the resulting displacement will be optimal for a depth of $50m$, but neither for $10m$ nor $100m$ where the objects of the scenes actually are.

6.3 Multiple shifts inference

As neural networks are traditionally computed within massively parallel architectures such as GPUs, multiple depth maps can be computed efficiently at the same time in a batch, especially for low resolution. Influence of batch size on inference speed has already been discussed in section 3.6.1 and has been shown to induce a less than linear slowdown on some architectures.

Batch inference of n frame pairs can then be used to compute depth with multiple shifts δt_i. These multiple depth maps can then be combined to construct a higher quality depth map, with high precision for both long and short range. We propose a dynamic range algorithm, described figure 6.4 to compute and combine different depth maps.

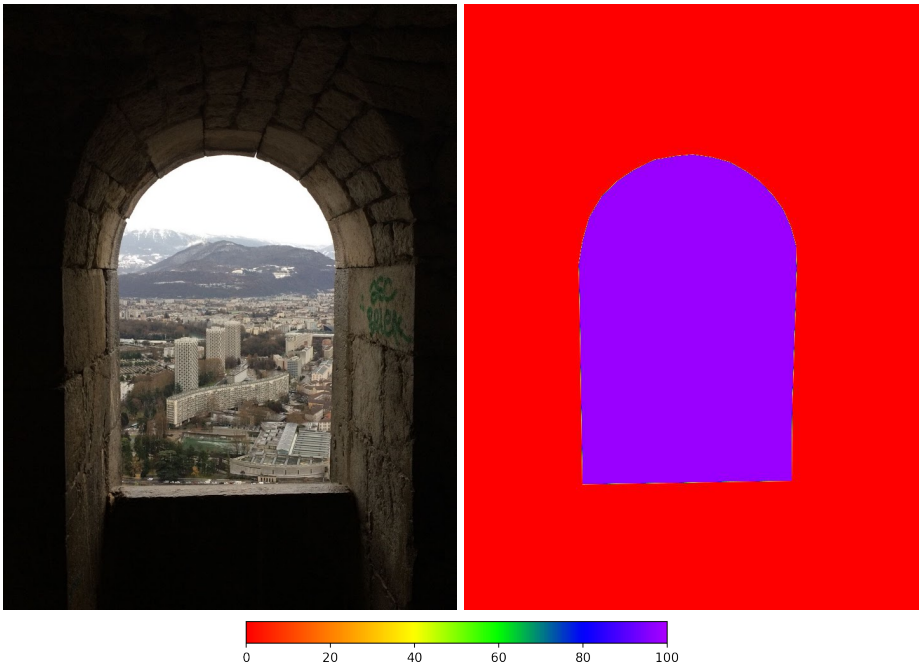


Figure 6.2: Example of a multi-modal depth distribution scene with corresponding depth: foreground is much closer than mean depth and background further.

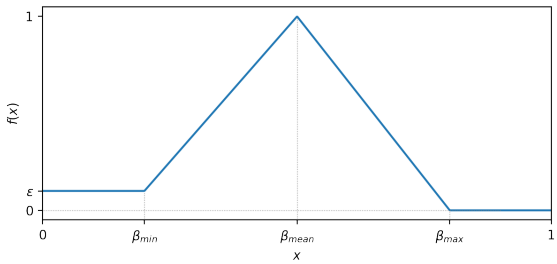


Figure 6.3: graph of function f, defined in equation 6.3

Instead of only one optimal displacement d_{opt} from $\mathbb{E}(\beta)$, we use K-mean clustering algorithm [Mac67] on the depth map to find n clusters on which each next input frame pair will be constructed to focus on. The clustering outputs a list of n centroids $c_i(\widetilde{\theta})$ and corresponding $d_{opt,i}(t)$ and δt_i. n has to be decided before inference and usually ranges from 1 to 4.

Final depth map is then computed from fusing these outputs using a weighted mean for each pixel. Each weight is actually a linear interpolation from 0 to 1 according to distance of β from a target value β_{mean}. See figure 6.3 for a graph of the weight function. That way, fusion will favor values that are closer to this optimal value. An ϵ value is added to solve fusion when every depth map is off its wanted range. This fusion process is very similar to exposure fusion [MKVR09], used in photography. Here, exposition is replaced by depth value, and the weight are based on distance to β_{mean} instead of Well-exposedness.

One can see how when $n = 1$, this technique is similar to the one described equation 6.2.

$$w_i(\boldsymbol{p}) = f(\beta(I_t, I_{t-\delta t_i}, \boldsymbol{p})) = f(\beta_i(\boldsymbol{p}))$$

$$
\begin{array}{cccc}
f & [0,1] & \rightarrow & [0,1] \\
 & x & \mapsto & \begin{cases}
\epsilon & \text{if } x < \beta_{min} \\
\epsilon + \left(\frac{x-\beta_{min}}{\beta_{mean}-\beta_{min}}\right)(1-\epsilon) & \text{if } \beta_{min} \leq x < \beta_{mean} \\
\frac{\beta_{max}-x}{\beta_{max}-\beta_{mean}} & \text{if } \beta_{mean} \leq x < \beta_{max} \\
0 & \text{if } x \geq \beta_{max}
\end{cases}
\end{array}
\tag{6.3}
$$

$$\forall \boldsymbol{p} \in \Omega, \widetilde{\theta}_f(\boldsymbol{p}) = \alpha \frac{\sum_i d_i^* w_i(\boldsymbol{p}) \beta_i(\boldsymbol{p})}{\sum_i w_i(\boldsymbol{p})}$$

Or more simply

$$\widetilde{\theta}_f = \alpha \sum_i \widehat{d}_i \beta_i \text{ where } \widehat{d}_i = \frac{d_i^* w_i}{\sum_j w_j}$$

In the rare case of $\sum_i w_i(\boldsymbol{p}) = 0$,meaning every $\beta_j(\boldsymbol{p})$ is above β_{max}, we take the convention of $\widehat{w}_i(\boldsymbol{p}) = +\infty$ and exclude this point from the K-mean clustering algorithm.

For our use-case, we set $\beta_{min} = 0.1$, $\beta_{mean} = 0.5$, $\beta_{max} = 0.9$ and $\epsilon = 10^{-3}$. i is the index of frame shift, j,k are the spatial indices. Figure 6.5 shows a result of the proposed algorithm for $n = 2$. Notice how the high shift detects buildings while low shift detects trees.

A more thorough presentation of the qualitative results can be viewed in video [1].

[1] `http://stillbox.ensta.fr/DepthNetResults.mp4`(retrieved 07/24/2019)

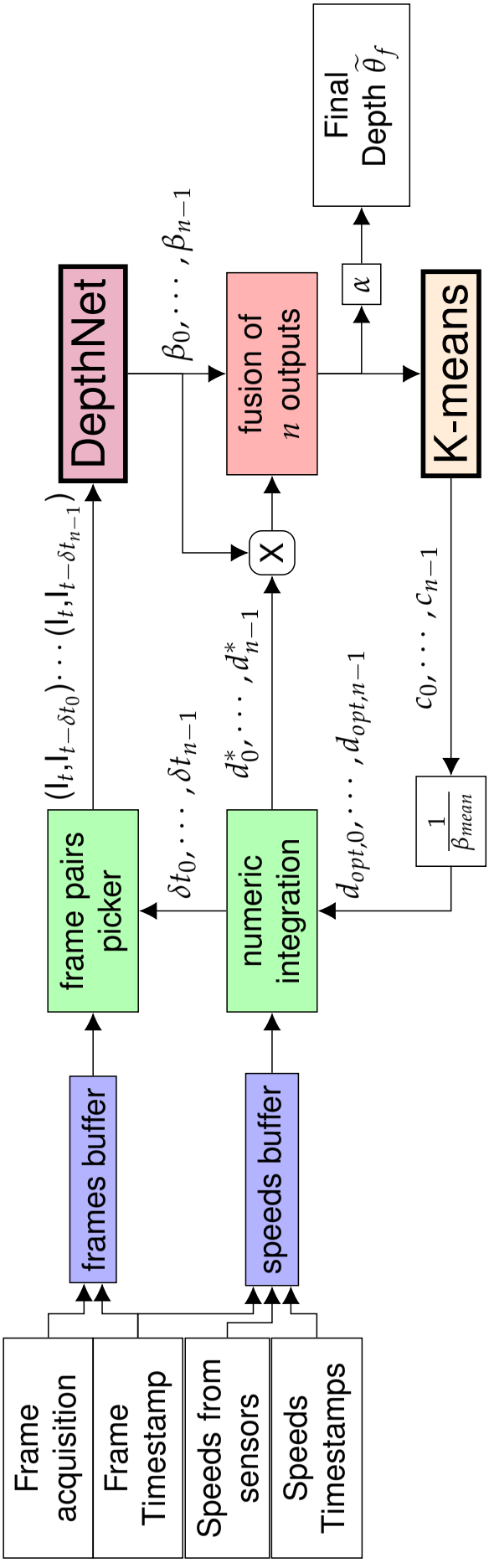


Figure 6.4: Multiple shifts architecture with n parallel depth inferences . Numeric integration, given a desired displacement d_{opt} gives the closest possible displacement between frames, denoted d^* , along with corresponding shift δt. The fusion block computes pixel-wise weights from $\beta_0, \cdots, \beta_{n-1}$ to make a weighted mean of $\beta_0 d_0^*, \cdots, \beta_{n-1} d_{n-1}^*$

6.4 Training a specific depth range with a clamped DepthNet

As discussed at the end of chapter 3, on very far objects (e.g. the sky), any minor optical flow caused by a default in stabilization will result in a massive error in depth. Moreover, our network being very good at recognizing shapes and giving it a continuous depth, this can result in the whole sky being computed as relatively close.

We thus propose an experimentation with a network designed for a simpler problem: during training on StillBox dataset, we clamp depth from $10m$ to $60m$, with a shift of 5 images (instead of 3 for DepthNet). These new parameters allow the network to only focus on mid range objects, dismissing close and far objects. This training workflow is very well suited for multiple shift depth inference. Every image pair will have a dedicated depth to analyze, preventing the fusion to be bothered with redundant data, because of the high initial range of DepthNet.

Figure 6.6 shows results for multiple synthetic 256x256 scenes with ground truth, along with inference speed and a small noise added to camera initial orientation at each frame. $R(t) = R_0 + Euler(N_0\mu(t))$, with $\mu(t)$ being a 3-dimensional random unit vector and N_0 a constant fixed to 0.001. We also report performance of a thin version of our clamped network, that shows better results than DepthNet with 1 plane only in this noisy setup. The thin network has the same depth, but every layer has half as many feature maps as the original DepthNet.

It is worth noting that DepthNet Clamped and Tiny DepthNet Clamped are just experimentations and no unsupervised fine-tuning workflow has been designed to be robust on clamped depth values with multiple fused inferences.

6.5 Our proof of concept

In order to test our Network, we developed a small use-case using a Bebop drone from Parrot. We were able to have a relatively good obstacle avoidance using DepthNet as a depth provider for an depth-powered obstacle avoidance tool in the UAV.

6.5.1 Model Predictive Control

Model Predictive Control (MPC)[Ric+78] is a control strategy that can be used for obstacle avoidance with a depth map [LH17]. This control algorithm has been internally developed at Parrot as a tool for experimentation, and has been used off-the-shelf without any attempt at improving it for the depth-map delivered by DepthNet. This tool expects a depth map from any available source, be it from our network or a disparity map from a stereo camera, or a point cloud from a Lidar.

This tool will correct a desired trajectory by either stopping or drifting around the obstacle. We are particularly interested by the second use-case, as our depth method needs motion.

6.5.2 Implementation details

The whole Obstacle avoidance proof-of-concept relies on Video streaming via Wifi into a GPU powered computer (Nvidia Quadro M1000M). Along with the video, additional information about the flight are embedded in the video stream, and extracted with Parrot internal demuxer, called PDraw [2]. That way, We can use speed and timestamps of the received frames to apply the algorithm presented figure 6.4. The depth is then discretized

[2] Available at `https://github.com/Parrot-Developers/pdraw`, retrieved 07/24/2019

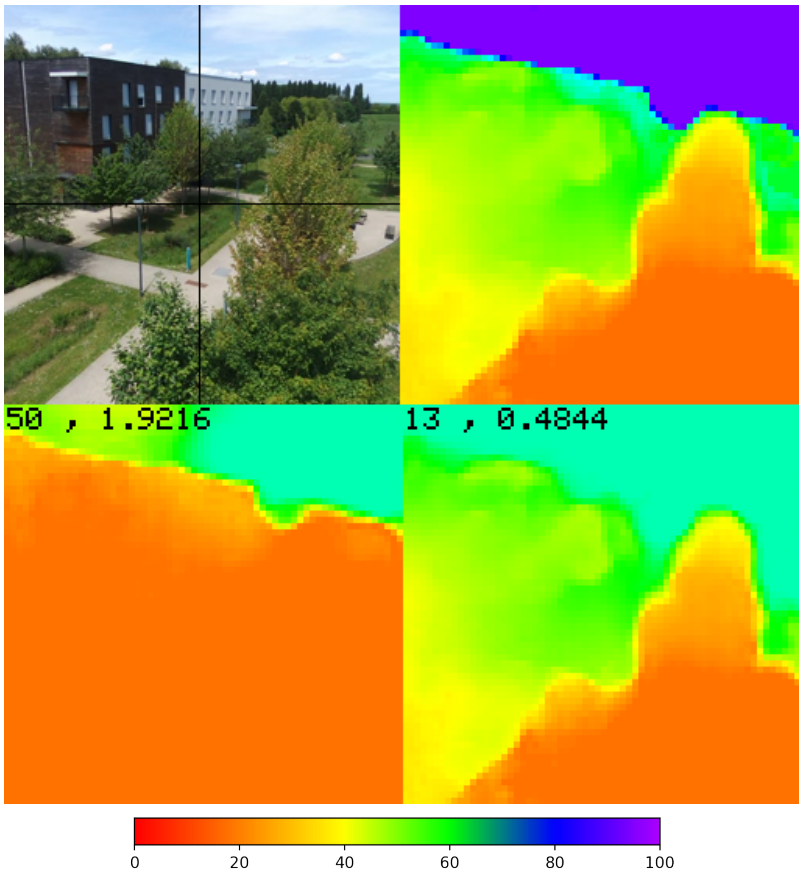


Figure 6.5: real condition application of the multi-shift algorithm with DepthNet. Top left is input. Last row are raw outputs of the network (from 0 to 100, corresponding to a β from 0 to 1). Top right is fused output, capped up to $100m$. Numbers in the upper left corner of raw inputs indicate respectively temporal frame shift and measured displacement magnitude in meters. The UAV is flying forward at a speed of $1m.s^{-1}$ and an altitude of $\approx 12m$.

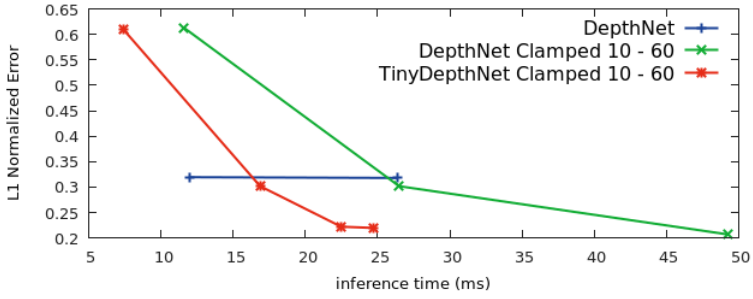


Figure 6.6: results for synthetic 256x256 scenes with noisy orientation. DepthNet has been tested with $n = 1$ and $n = 2$, DepthNet Clamped with $n = 1 \cdots 3$ Tiny DepthNet Clamped with $n = 1 \cdots 4$. Y axis is Mean Relative Error (MRE), X axis is inference speed, in ms on a Quadro M1000M GPU.

Figure 6.7: Experimental setup for our proof of concept

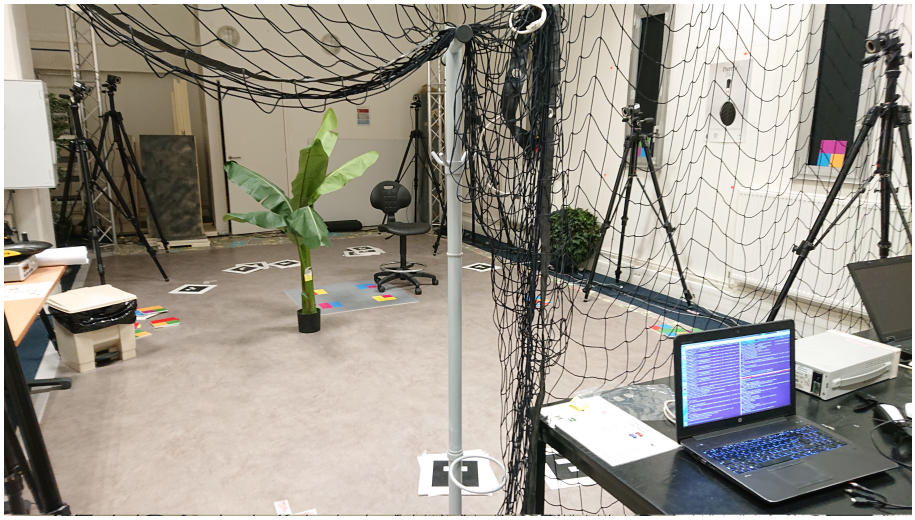


Figure 6.8: Experimental Scene where the drone is expected to travel around obstacles when desired trajectory collides with them

and sent back to the Bebop which will use it for its MPC algorithm. See figure 6.7 for a picture of the experimental setup.

This proof of concept has been used in cluttered area used by Parrot for drones prototyping, presented figure 6.8. It has been tested on two obstacles (here the plant and the desk chair), but also on the white wall which was not a problem for our depth sensing protocol and the drone could drift along the wall for multiple flights. For displacement estimation, we used the embedded odometry algorithm, using Bebop's vertical camera. A video is available for further results in our proof of concept. [3].

6.6 Conclusions for this chapter

We proposed in this chapter a way of using depth from our network to get a first working solution. We were also able to apply a multiple shift architecture to be robust to scenes with large depth heterogeneity, especially with long range obstacles.

[3] `http://stillbox.ensta.fr/poc_depthnet_mpc.mp4` (retrieved 07/24/2019)

This proof of concept is also very promising because every operation used is actually differentiable, even the MPC algorithm, thanks to a recent work by Amos et al. [Amo+18]. That way it becomes possible to imagine a training workflow where DepthNet output would not be trained to get a good depth quality measure anymore, but rather so that the resulting MPC based on it will get a good trajectory.

As discussed section 1.3.3, ensuring a good trajectory will still require expensive human supervision with the possible usage of DaGGer [RGB11]. Nonetheless, this chapter provides a good non-random starting point for training.

Chapter 7

Perspectives and Conclusions

Contents

7.1 Limitations and future perspectives 117
7.1.1 UAV depth validation dataset 117
7.1.2 Robust smooth and photometric loss 118
7.1.3 Moving scenes 118
7.1.4 Bayesian depth inference 119
7.2 Alternative Perspectives 119
7.2.1 Depth sensing for multi-task learning 119
7.2.2 Robust and evolutive passive depth sensing 119

We showed in this work a working strategy to train a neural network to sense depth based on motion from a stabilized camera, for any kind of domain. We even showed that the next step in the general obstacle avoidance strategy, which was to train a neural network to avoid obstacles from a perfect depth map provided by a simulator before concatenating it with a real depth sensing solution might be replaced by a differentiable MPC [Amo+18]. The strategy presented in the introduction (section 1.4.3) and in figure 1.8 is still valid and might be the subject for future work.

However, thanks to a thorough study on possible drawbacks of depth from vision algorithms, we could identify several drawbacks of DepthNet that might need to be solved before being used as a reliable source of information for obstacle avoidance.

7.1 Limitations and future perspectives

7.1.1 UAV depth validation dataset

As heavily discussed on chapter 5, the KITTI dataset [Gei+13] is not well suited to demonstrate the advantages of our solution. On the other hand, due to the lack of other validation sets for depth sensing in the context of outdoor flight, we could only have a subjective validation for the UAV use-case.

It thus seems necessary to construct a validation set with reliable ground-truth depth and navigation data the same way KITTI was constructed. Several solutions can be considered:

- Construct a UAV with a calibrated Lidar, similarly to KITTI. It is highly probable that such a device would not be easy to construct and would be particularly heavy and dangerous. However, it would be robust to moving objects.

- Use a heavy structure-from-motion algorithm to get high quality depth map from 3D reconstruction. Algorithms such as Colmap [Sch+16] can be used, or even proprietary solutions like pix4D [1]. The latter has the advantage of automatizing the flight for an optimized mapping. We can then imagine a two step recording, the first one being dedicated to mapping and the second one to realistic flight conditions in the same scene, getting the ground truth depth map from its 3D reconstruction. The obvious drawback of this solution is the necessity of a rigid scene.

- Using structure from motion with a stereo camera could be an interesting mid-point. Moving objects can then be detected within a particular range.

7.1.2 Robust smooth and photometric loss

As heavily discussed in chapter 4, today's state of the art regarding self supervision for depth sensing mostly relies on heuristics, themselves based on validation results on the KITTI dataset. A clear understanding of theoretical aspect of self supervision is needed to perform a truly robust training workflow. A recent interesting work [Bar19] proposed to replace the L1 photometric loss with a more general equation, where a differentiable parameter could modify the photometric loss profile, and make it more or less sensitive to outliers, depending on the sample. In this work, among other tests, Barron showed that by simply replacing L1 photometric loss with its general form without changing anything else from original SFMLearner code [Zho+17], he could improve the results initially obtained.

Also, as already discussed section 4.4.5, smoothing could benefit from already developed solutions for large displacement optical flow. A workflow using the same two step joint optimization as DTAM [NLD11] for smoothing seems very promising.

Finally, as hinted by the SSIM, a photometric loss that would also take neighboring pixels into account could be beneficial in order to get more information than only color to decide whether two points match or not. This was already studied by Brox and Malik [BM11]. In this work, they considered several features and descriptors to be matched and deduced that including descriptor matching inside the energy minimization process was beneficial, especially for large displacement. The problem at the time was the lack of differentiability and sub pixel accuracy of descriptors. This problem could be easily adapted for photometric loss with descriptors composed of convolution kernels, resulting in a feature matching loss.

More generally, self supervision should rely more on techniques developed for optical flow using the variational approach, because occlusion problems along with efficient smoothing and illumination robustness are definitely not new.

7.1.3 Moving scenes

The most important limitation of our work is of course the problem of moving scenes which is explicitly not taken into account in the structure-from-motion paradigm.

As shown by Ranjan et al. [Ran+18], it is possible to estimate moving objects in a scene, based on the fact that optical flow does not match a specific pattern from estimated camera movement. That way, depth completion techniques can be used to deduce their depth from context. Unfortunately, in addition to having to run another potentially heavy network, that takes us back to the robustness problems we raised in section2.2.1 for depth from a single frame.

A possible solution could also be to use a stereo rig for ambiguous cases. This solution has obviously a limited range, but might be enough for safety regarding humans: if

[1] `https://www.pix4d.com/`, retrieved 07/24/2019

the drone is moving too fast for emergency braking to be possible (typically at more than $10m.s^{-1}$), the human will be moving slowly compared to the camera, making the resulting structure-from-motion less erroneous.

If we want a truly moving scene robust depth inference solution, we might have to look at time-sequence based models such as recurrent neural networks, that explicitly exploit the time continuity of information: ambiguous perspectives and scales can be solved with anterior depth inferences [Man+17].

Finally, it should be noted that depth from moving scene problem might only be important when the end goal is a quality depth map. It might be interesting to consider the possibility of a neural network dedicated to obstacle avoidance pretrained using depth-from-motion and thus not being robust to moving scenes at first, but then being trained to avoid moving obstacle as well.

7.1.4 Bayesian depth inference

For obstacle avoidance, it is interesting to know how confident the network is about a particular inference. As heavily stressed by e.g. Kendall et al. [KG17; McA+17], Bayesian inference, not only giving an estimate of a particular value (here the depth) but also the uncertainty, and more generally the distribution of possible values can be very beneficial on decision making. That way, particular system can take extra precautions when in an ambiguous situation. Our main motivation being robustness, it's easy to see how beneficial it could be to safety when operating with moving objects or in an unknown domain.

7.2 Alternative Perspectives

This work has been very directed toward the end goal of obstacle avoidance, which might lead to believe that depth sensing is only interesting for pretraining purpose in a fully neural avoidance system.

However, some other applications derived from this work can be used in other contexts.

7.2.1 Depth sensing for multi-task learning

As discussed in section 1.3.4 and shown by Kendall et al. [KGC18], multi task is not only a way to combine several tasks efficiently in term of computational cost, but also in term of quality when tasks are correlated. It is particularly interesting in the context of object proposal tasks. Indeed, products like Parrot or DJI usually propose a follow-me/ActiveTrack functionality that proposes targets to track in the streamed footage on the user's phone. There is then already a dedicated training for object proposal networks, grabbing a pretrained general purpose object detection such as SSD [Liu+16] or YOLO [Red+16] and finetuning it in a dedicated follow-me training set. The addition of depth sensing for this training set has the potential to significantly improve the feature extraction and thus the finetuning, because as illustrated figure 7.1, typical proposed objects highly correlate with depth discontinuities.

7.2.2 Robust and evolutive passive depth sensing

An interesting perspective that has been developed throughout this thesis it the idea of having a robust and evolutive depth sensing system based on vision. In other words, such a system does not require any kind of active sensors, and can improve during its

Figure 7.1: Multiple different but correlated semantic levels of a particular scene. From left to right: input, semantic segmentation masks, instance masks, depth. Figure taken from Kendall et al. [KGC18]

real-time usage. This has been discussed by Casser et al. [Cas+19], where it was used to overfit to each example of the test set (with a reset of the network between examples).

This technique can be very beneficial in the context of stealth off-road autonomous vehicles. Due to their stealthiness, no active depth sensors can be used and depth has to be deduced from cameras. Besides, off-road environment are typically more heterogeneous than a clean road, and our robustness oriented solution can be beneficial compared to a classical structure from motion analytical algorithm or an evolutive single frame depth solution.

-

Appendix A

Illustration of compensating effect of linking depth scaled factor to movement estimation

This appendix is first referenced in section 2.1.

Let's consider the simplest obstacle avoidance maneuver which consists in braking with a constant force opposed to velocity vector in front of an obstacle at a distance θ knowing the vehicle velocity magnitude v.

The vehicle speed needs to get to 0 before having traveled the distance θ. A simple application of work to kinetic energy relation can tell us that the force magnitude needed will be

$$F_0 = \frac{mv^2}{2\theta} \tag{A.1}$$

where m is the mass of the vehicle. Now if we consider an error on measured speed so that the estimation is $\widetilde{v} = \alpha v$ with $\alpha \neq 1, \alpha > 0$, in case of a depth estimated independently of $\widetilde{v}$ (say from a Lidar, or a stereo camera), the estimated force needed to brake on time will be $\widetilde{F} = \alpha^2 F_0$, and the actual distance to stop will be $\frac{\theta}{\alpha^2}$.

On the other hand, in the case described section 2.1, where the scale factor is given by the ratio between apparent speed and measured speed, the estimated depth $\widetilde{\theta}$ will be $\widetilde{\theta} = \alpha\theta$, the resulting force will be $\widetilde{F} = \alpha F_0$ and the distance to brake will be $\frac{\theta}{\alpha}$.

The error in navigation strategy, while still present, is less impacted with depth estimated relative to motion than absolute depth.

Appendix B

Proof on under-estimated depth over-representation when optimizing relative error for a validation set

This appendix is first referenced in section 2.4.2. The goal is to prove the proposition 1

Proposition. *Given an estimator e responsible for estimations $\{\widetilde{\theta}\}$, in a corresponding validation dataset $V = \{(\widetilde{\theta}, \theta)\}$, if the estimator is optimized for that validation set for a mean relative difference error, it will have more under-estimations than over-estimations.*

$$P(\widetilde{\theta} < \theta | (\widetilde{\theta}, \theta) \in V) > 0.5$$

Corollarily, if we have an evenly balanced estimator $\{\widetilde{\theta}\}$ such that $P(\widetilde{\theta} < \theta | (\widetilde{\theta}, \theta) \in V) = 0.5$, then we can find $\alpha < 1$ such that $\{\alpha\widetilde{\theta}\}$ is a better estimator according to mean relative difference MRE.

Proof. Let's consider the mean relative error of a set of estimations $\widetilde{\theta}$ with respect to groundtruth values θ. The error is computed on the set of the estimation/ground truth pairs. $V = \left\{(\widetilde{\theta}, \theta)\right\}$.

$$MRE(\{\widetilde{\theta}\}) = \mathbb{E}_{(\widetilde{\theta},\theta)\in V} \frac{|\widetilde{\theta} - \theta|}{\theta}$$

We consider an estimator $\{\widetilde{\theta}\}$ to be optimized if for every possible real function f (which is not dependent to the input of the estimator), the estimator $\{f(\widetilde{\theta})\}$ has a worse error. $MRE(\{\widetilde{\theta}\}) < MRE(\{f(\widetilde{\theta})\})$

In this proof we assume that the estimator is not perfect so that no value $\widetilde{\theta}$ is exactly the same as the corresponding θ. $\forall(\widetilde{\theta}, \theta) \in V, \widetilde{\theta} \neq \theta$, which is reasonable considering continuous values.

Let's then consider a value ϵ such that $|\epsilon| < \min_{(\widetilde{\theta},\theta)\in V} \frac{|\widetilde{\theta} - \theta|}{\widetilde{\theta}}$.

For ease of notation we write $V_- = \left\{(\widetilde{\theta}, \theta), \widetilde{\theta} < \theta\right\}$ and $V_+ = \left\{(\widetilde{\theta}, \theta), \widetilde{\theta} > \theta\right\}$. These are the sets where the target values are respectively under-estimated and over-estimated. We can already see that $|V_-| + |V_+| = |V|$. The goal of this demonstration is then to show that $|V_-| > |V_+|$

If we then consider the estimator $\{\widetilde{\theta}(1 + \epsilon)\}$, we should get a worse mean relative error.

$$
\begin{aligned}
MRE(\{\widetilde{\theta}(1+\epsilon)\}) &= \mathbb{E}\left(\frac{|\widetilde{\theta}+\epsilon\widetilde{\theta}-\theta|}{\theta}\right) \\
&= \frac{|V_+|}{|V|}\mathbb{E}_{V_+}\left(\frac{\widetilde{\theta}+\epsilon\widetilde{\theta}-\theta}{\theta}\right) + \frac{|V_-|}{|V|}\mathbb{E}_{V_-}\left(\frac{\theta-\widetilde{\theta}-\epsilon\widetilde{\theta}}{\theta}\right) \\
&= MRE(\{\widetilde{\theta}\}) + \frac{\epsilon}{|V|}\left(|V_+|\mathbb{E}_{V_+}\left(\frac{\widetilde{\theta}}{\theta}\right) - |V_-|\mathbb{E}_{V_-}\left(\frac{\widetilde{\theta}}{\theta}\right)\right)
\end{aligned}
$$

Since the maximum is obtained for ϵ = 0, we cannot have the term in parenthesis different to zero, otherwise we could find a better estimator by choosing $\epsilon \neq 0$ with the same sign of the term in parenthesis.

As consequence we have

$$
|V_+|\mathbb{E}_{V_+}\left(\frac{\widetilde{\theta}}{\theta}\right) = |V_-|\mathbb{E}_{V_-}\left(\frac{\widetilde{\theta}}{\theta}\right) \tag{B.1}
$$

By construction, we have $\mathbb{E}_{V_-}\left(\frac{\widetilde{\theta}}{\theta}\right) < \mathbb{E}_{V_+}\left(\frac{\widetilde{\theta}}{\theta}\right)$.

As a consequence, we have $|V_-| > |V_+|$.

By contraposition, we can say that if we have $|V_-| = |V_+|$, we can find a scale factor $\alpha < 1$ such that $\{\alpha\widetilde{\theta}\}$ will have a better mean relative error.

□

Appendix C

Proof on smoothing an inverse depth map

This appendix is first referenced in section 4.4.3. The goal is to prove the proposition 2

Proposition. *The inverse depth map $\xi = \frac{1}{\theta}$ of a perfect plane has a null Laplacian and is then stable with respect to smoothing.*

Proof. The surface the camera is looking at can be modelized as an affine function of x and y in the world coordinates $(0, \boldsymbol{u}_x, \boldsymbol{u}_y, \boldsymbol{u}_z)$, $\boldsymbol{u}_z$ standing for the optical axis of the camera.

$$z = \theta_0 + \alpha x + \beta y$$

θ_0, α and β being constant values.
Besides, from the pinhole model, for a point (x, y, z), we get the coordinates

$$u(x,z) = f\frac{x}{z} + u_0$$
$$v(y,z) = f\frac{y}{z} + v_0$$

As a consequence, we can have the depth θ as a function of image coordinates (u, v)

$$\theta(u,v) = z = \theta_0 + \frac{\alpha z(u-u_0)}{f} + \frac{\beta z(v-v_0)}{f}$$
$$\left(1 + \frac{\alpha(u-u_0)}{f} + \frac{\beta(v-v_0)}{f}\right)\theta(u,v) = \theta_0$$
$$\frac{1}{\theta(u,v)} = \frac{f + \alpha(u-u_0) + \beta(v-v_0)}{f\theta_0}$$

As a consequence, there exist two constant values γ_1, γ_2 such that:

$$\forall (u,v) \in \mathbb{R}^2, \frac{1}{\theta}(u,v) = \gamma_1 + \gamma_2 u$$

This implies that the Laplacian of inverse depth is 0 everywhere, and thus the diffusion operation equivalent to identity.

Then, making the Laplacian of inverse depth $\frac{1}{\theta} = \xi$ converge to 0 is stable with respect to any perfect plane. □

Appendix D

A Collection of Proofs on diffusion

D.1 Preamble

This appendix is first referenced in section 4.4.4.

This is a collection of proofs regarding propositions on diffusion and gradient diffusion. They all use to some extent a form of the multi-dimensional integration by parts theorem:

Theorem 1. *let $\Omega \subset \mathbb{R}^n$ be a regular region, and let $\boldsymbol{n}$ be the unit outward normal to $\partial\Omega$ (boundaries of Ω) suppose $\phi : \overline{\Omega} \to \mathbb{R}$ and $\boldsymbol{v} : \overline{\Omega} \to \mathbb{R}^n$ are $\mathcal{C}^1$ functions. Then*

$$\int_\Omega \phi \text{div}(\boldsymbol{v}) dV = \int_{\partial\Omega} \phi \boldsymbol{v}.\boldsymbol{n} dS - \int_\Omega \boldsymbol{v}.\nabla\phi dV \tag{D.1}$$

As a basis for our future demonstrations, we can reformulate the proof from Perona and Malik [PM90]:

Theorem 2. *let $\Omega \subset \mathbb{R}^n$ be a regular region, and let $\boldsymbol{n}$ be the unit outward normal to $\partial\Omega$ (boundaries of Ω). Finally, let $\mathcal{L}_s$ be the function defined on scalar fields defined on Ω as*

$$\begin{array}{cccc} \mathcal{L}_s & \mathbb{R}^\Omega & \to & \mathbb{R} \\ & \xi & \mapsto & \frac{1}{2}\iint_\Omega c(\|\nabla\xi\|^2) dV \end{array} \tag{D.2}$$

Then the pointwise gradient defined on Ω as $\nabla_\xi \mathcal{L}_s$:

$$\forall \boldsymbol{p} \in \Omega, \nabla_\xi \mathcal{L}_s(\boldsymbol{p}) = -\textit{div}(c'(\|\nabla\xi(\boldsymbol{p})\|^2)\nabla\xi) + \begin{cases} c'(\|\nabla\xi\|^2)\nabla\xi.\boldsymbol{n} & \textit{if } \boldsymbol{p} \in \partial\Omega \\ 0 & \textit{otherwise} \end{cases} \tag{D.3}$$

Proof. Let's consider the smooth loss introduced at equation 4.4

$$\mathcal{L}_s(\xi) = \frac{1}{2}\iint_\Omega c(\|\nabla\xi\|^2) dV \tag{D.4}$$

Now if we consider h a $\mathcal{C}^1$ function on Ω and $t \in \mathbb{R}$ we can try to compute the function $\mathcal{L}_s(\xi + th)$ and especially its derivate at $t = 0$

$$\begin{aligned} \frac{d}{dt}_{t=0} \mathcal{L}_s(\xi + th) &= \frac{1}{2}\iint_\Omega \frac{d}{dt}_{t=0} c(\|\nabla\xi + t\nabla h\|^2) dV \\ &= \iint_\Omega c'(\|\nabla\xi\|^2)\nabla\xi.\nabla h dV \end{aligned}$$

By identifying $\phi = h$ and $v = c'(\|\nabla\xi\|^2)\nabla\xi$ in theorem 1, we can apply our theorem and deduce

$$\frac{d}{dt}_{t=0}\mathcal{L}_s(\xi + th) = -\iint_\Omega \text{div}(c'(\|\xi\|^2)\nabla\xi)hdV + \int_{\partial\Omega} c'(\|\xi\|^2)\nabla\xi h.\boldsymbol{n}dS$$

this equation is valid for every function h. As a consequence, we can deduce the aforementioned result. □

D.2 Diffusion stability

First referenced in section 4.4.5.

We can consider the infinite 1D function

$$\forall i \in \mathbb{Z}, u_0(i) = u(i,0) = (-1)^i \tag{D.5}$$

The diffusion equation with a step size of η states

$$\forall (i,n) \in \mathbb{Z}\times\mathbb{N}^+, u(i,n\eta) = u(i,(n-1)\eta) + \eta\Delta(u(i,(n-1)\eta))$$

Also, the discrete Laplacian of u_0 can be computed to get

$$\Delta u(i,0) = -2u(i,0) + u(i-1,0) + u(i+1,0) = -4u(i,0)$$

As a consequence,

$$u(i,\eta) = u(i,0) - 4\eta u(i,0) = (1-4\eta)u(i,0)$$

By recurrence, it's easy to see

$$\forall (i,n) \in \mathbb{Z}\times\mathbb{N}, u(i,n\eta) = (-1)^i \times (1-4\eta)^n \tag{D.6}$$

If η is above 0.5, the solution diverges. The middle value oscillates around the null-Laplacian solution with increasing magnitude.

D.3 Diffusion of Inverse

Also first referenced in section 4.4.5.

A simple and pratical example of instability is when the trainable parameters are the depth map θ and diffusion is applied on ξ. We then get:

$$\begin{array}{rccc} f: & \mathbb{R}^{|\Omega|} & \to & \mathbb{R}^{|\Omega|} \\ & \theta & \mapsto & \xi = \frac{1}{\theta} \end{array}$$

The jacobian $\mathbf{J}_f$ (of size $|\Omega|^2$) can be written as

$$\mathbf{J}_f(i,j) = -\delta_{ij} \times \frac{1}{\theta_i^2}$$

Where δ_{ij} is 1 if $i = j$ otherwise 0.

$$\partial_t\xi = \frac{\Delta\xi}{\theta^4} = \Delta\xi \times \xi^4 \tag{D.7}$$

The closer depth θ is to 0, the larger the effective optimization step, and possibly above the divergence threshold. (see section D.2)

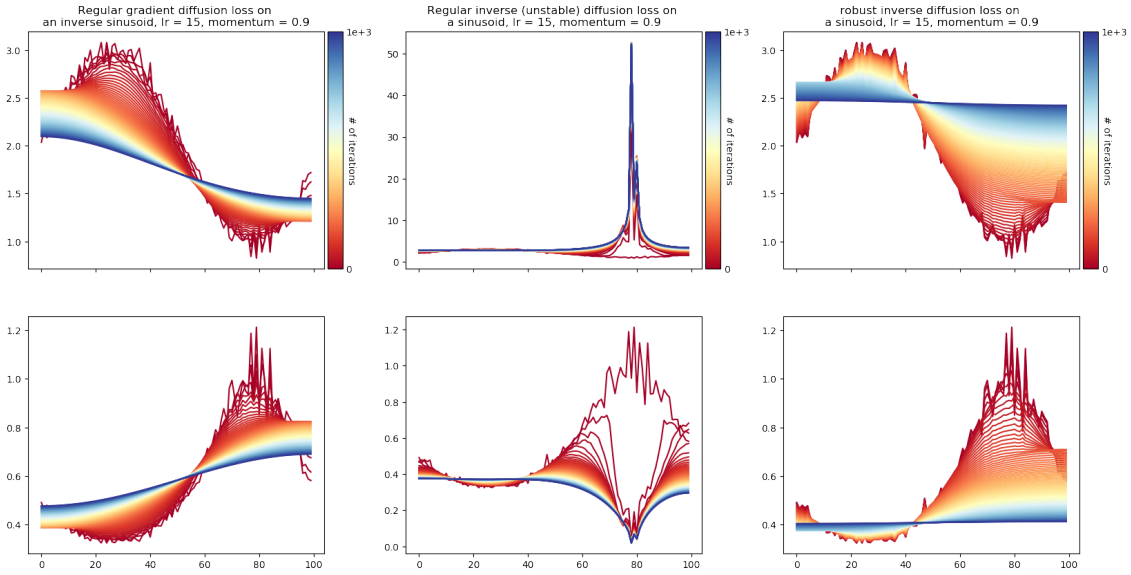


Figure D.1: Left: diffusion of the noisy function $x \mapsto \frac{1}{2+sin(x)}$ (bottom) and its corresponding inverse (top). Center: diffusion of the inverse of the noisy function $x \mapsto 2 + sin(x)$ (bottom) and the resulting function (top). Right: Evolution obtained by interleaving diffusion (10 iterations, learning rate of 0.4) and minimization of difference with diffused.

Figure D.1 and figure D.2 show our first method and the deemed more "robust" one when trying to smooth the inverse of $2 + sin(x)$ with noise, with both regular diffusion loss (4.4) and diffusion of gradient (4.5). While not being exactly the same as a regular diffusion, it appears that the robust method helps to get a more regularized smoothed map, especially where the gradient from trainable map to the map we want to diffuse is the highest: naive application of diffusion loss makes the parameters diverge.

D.4 Diffusion of Gradient

D.4.1 Laplacian minimization

First referenced in section 4.4.7. The goal is to prove the equation 4.12

$$\nabla_\xi \mathcal{L}_{s\Delta} = -\Delta(c'((\Delta\xi)^2)\Delta\xi)$$

with

$$\mathcal{L}_{s\Delta} = \frac{1}{2}\iint_\Omega c\left((\Delta\xi)^2\right) dS$$

Proof. We proceed the same way as for theorem 2. The considered loss function is

$$\mathcal{L}_s = \frac{1}{2}\int_\Omega c((\Delta\xi)^2) dV \tag{D.8}$$

We consider h a $\mathcal{C}^2$ function on Ω and $t \in \mathbb{R}$ and compute the expression

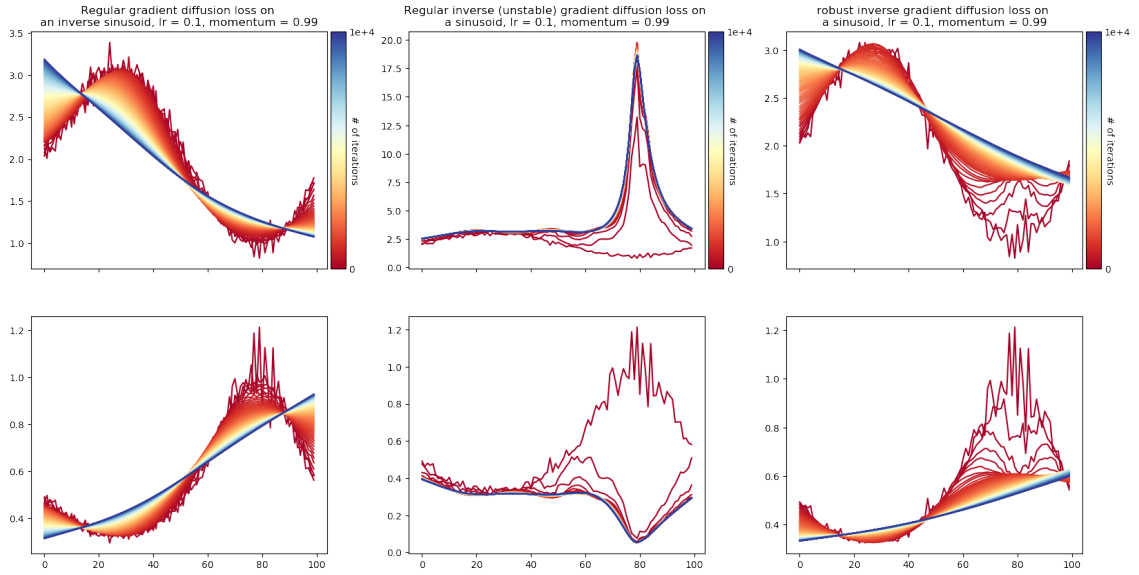


Figure D.2: Gradient diffusion equivalent of figure D.1.

$$\begin{aligned}\frac{d}{dt}_{t=0}\mathcal{L}_s(\xi+th) &= \frac{1}{2}\iint_\Omega \frac{d}{dt}_{t=0} c(\|\Delta\xi + t\Delta h\|^2)dV \\ &= \iint_\Omega c'(\|\Delta\xi\|^2)\Delta\xi\Delta h dV \\ &= \iint_\Omega c'(\|\Delta\xi\|^2)\Delta\xi \text{div}\nabla h dV\end{aligned}$$

For ease of notation, we will now identify $c'(\|\Delta\xi\|^2)\Delta\xi$ as K. From integration by part formula (theorem 1), we identify $\phi = \nabla h$ and $\boldsymbol{v} = K$.

$$\frac{d}{dt}_{t=0}\mathcal{L}_s(\xi+th) = -\iint_\Omega \nabla K.\nabla h dV + \int_{\partial\Omega}(K\boldsymbol{n}).\nabla h dS$$

This time, we apply the formula twice. For first term (on Ω), we identify $\phi = h$ and $\boldsymbol{v} = \nabla K$.

$$\iint_\Omega \nabla K.\nabla h dV = -\iint_\Omega h\text{div}\nabla K dV + \int_{\partial\Omega} h\nabla K.\boldsymbol{n} dS \tag{D.9}$$

For second term (on $\partial\Omega$), we identify $\phi = h$ and $\boldsymbol{v} = K\boldsymbol{n}$. It can be noted that since $\partial\Omega$ is a closed set, its boundaries are an empty set $\partial\partial\Omega = \{\emptyset\}$. We can also that $\boldsymbol{n}$ is piecewise constant in our case where $\partial\Omega$ is a rectangle. Thus div$\boldsymbol{n} = 0$.

$$\int_{\partial\Omega}(K\boldsymbol{n}).\nabla h dS = -\int_{\partial\Omega} h\text{div}(K\boldsymbol{n})dS = -\int_{\partial\Omega} h\nabla K.\boldsymbol{n} dS \tag{D.10}$$

We can combine equations D.9 and D.10 to get:

$$\frac{d}{dt}_{t=0}\mathcal{L}_s(\xi+th) = \iint_\Omega h\Delta K dV - 2\int_{\partial\Omega} h\nabla K.\boldsymbol{n} dS$$

We can deduce the following expression for $\nabla_{\xi}\mathcal{L}_s$:

$$\forall \boldsymbol{p} \in \Omega, \nabla_{\xi}\mathcal{L}_s(\boldsymbol{p}) = -\Delta(c'((\Delta\xi)^2)\Delta\xi) + \begin{cases} \nabla(c'((\Delta\xi)^2)\Delta\xi).\boldsymbol{n} & \text{if } \boldsymbol{p} \in \partial\Omega \\ 0 & \text{otherwise} \end{cases} \tag{D.11}$$

□

D.4.2 Divergence minimization

First referenced in section 4.4.8. The goal is to prove the equation 4.14

$$\nabla_{\boldsymbol{A}}\mathcal{L}_{s\Delta} = -\nabla\left(c'\left(\text{div}(\boldsymbol{A})^2\right)\text{div}(\boldsymbol{A})\right)$$

with

$$\mathcal{L}_{s\Delta} = \iint_{\Omega} c\left(\text{div}(\boldsymbol{A})^2\right) dS$$

Proof. We want to compute $\nabla_{\boldsymbol{A}}\mathcal{L}_s$

We now consider $\boldsymbol{h}$ a $\mathcal{C}^1$ multidimensional function on Ω and $t \in \mathbb{R}$ and compute the expression

$$\begin{aligned} \frac{d}{dt}_{t=0}\mathcal{L}_s(\boldsymbol{A}+t\boldsymbol{h}) &= \frac{1}{2}\iint_{\Omega}\frac{d}{dt}_{t=0}c((\text{div}\boldsymbol{A}+t\text{div}\boldsymbol{h})^2)dV \\ &= \iint_{\Omega} c'((\text{div}\boldsymbol{A})^2)\text{div}\boldsymbol{A}\text{div}\boldsymbol{h}dV \end{aligned}$$

We now identify $K = c'((\text{div}\boldsymbol{A})^2)\text{div}\boldsymbol{A}$. We also apply formula from theorem 1 by identifying $\boldsymbol{v} = \boldsymbol{h}$ and $\phi = K$.

$$\frac{d}{dt}_{t=0}\mathcal{L}_s(\boldsymbol{A}+t\boldsymbol{h}) = -\iint_{\Omega}\boldsymbol{h}.\nabla K dV + \int_{\partial\Omega} K\boldsymbol{h}.\boldsymbol{n}dS$$

We can then deduce:

$$\forall \boldsymbol{p} \in \Omega, \nabla_{\boldsymbol{A}}\mathcal{L}_s(\boldsymbol{p}) = -\nabla(c'((\text{div}\boldsymbol{A})^2)\text{div}\boldsymbol{A}) + \begin{cases} (c'((\text{div}\boldsymbol{A})^2)\text{div}\boldsymbol{A})\boldsymbol{n} & \text{if } \boldsymbol{p} \in \partial\Omega \\ 0 & \text{otherwise} \end{cases} \tag{D.12}$$

□

D.4.3 Isotropic gradient diffusion

First referenced in section 4.4.8. The goal is to prove the equation 4.16

$$\nabla_{\boldsymbol{A}}\mathcal{L}_{gdiff} = -\Delta\boldsymbol{A}$$

where

$$\mathcal{L}_{gdiff} = \iint_{\Omega}(\text{div}\boldsymbol{A})^2 dS$$

and $\nabla(\text{div}\boldsymbol{A}_0) = \Delta\boldsymbol{A}_0$ at the begining of the optimization.

Proof. If we retake the result from equation 4.15, we get:

$$\partial_t \boldsymbol{A} = \nabla(\text{div}(\boldsymbol{A})) = \Delta\boldsymbol{A} + \rho(\boldsymbol{A}) = \Delta\boldsymbol{A} + \begin{bmatrix} \frac{\partial^2 \boldsymbol{A}_y}{\partial y \partial x} - \frac{\partial^2 \boldsymbol{A}_x}{\partial y^2} \\ \frac{\partial^2 \boldsymbol{A}_x}{\partial x \partial y} - \frac{\partial^2 \boldsymbol{A}_y}{\partial x^2} \end{bmatrix} \tag{D.13}$$

The second term of this equation $\rho\boldsymbol{A}$ can be interpreted as the double rotational. Indeed, when considering the 3D extension $\boldsymbol{A}_3$ of our vector $\boldsymbol{A}$, $\boldsymbol{A}_3 = \begin{bmatrix} \boldsymbol{A}_x \\ \boldsymbol{A}_y \\ 0 \end{bmatrix}$, we can easily verify that

$$\text{rot}(\text{rot}(\boldsymbol{A}_3)) = \nabla \wedge (\nabla \wedge \boldsymbol{A}_3) = \begin{bmatrix} \frac{\partial^2 \boldsymbol{A}_y}{\partial y \partial x} - \frac{\partial^2 \boldsymbol{A}_x}{\partial y^2} \\ \frac{\partial^2 \boldsymbol{A}_x}{\partial x \partial y} - \frac{\partial^2 \boldsymbol{A}_y}{\partial x^2} \\ 0 \end{bmatrix} = \begin{bmatrix} \rho(\boldsymbol{A}) \\ 0 \end{bmatrix}$$

from the known relation

$$\Delta\boldsymbol{A}_3 = \nabla(\text{div}(\boldsymbol{A}_3)) - \text{rot}(\text{rot}(\boldsymbol{A}_3))$$

we can extrapolate for our 2D case. Besides, by noting that for any 3D potential field V we have the equality

$$\text{rot}(\nabla V)) = \boldsymbol{0}$$

we can retrieve the vectorwise diffusion equation. It can also be shown that the second term of our equation is constant throughout the whole diffusion

We thus have to prove that for any 2D vector field $\boldsymbol{A}$ such that $\rho(\boldsymbol{A}) = \boldsymbol{0}$, the minimization of its divergence is equivalent to vectorwise diffusion.

This is already verified at the beginning of the diffusion process, when t=0. Our goal is then to show that the vector

$$\rho(\boldsymbol{A}) = \begin{bmatrix} \frac{\partial^2 \boldsymbol{A}_y}{\partial y \partial x} - \frac{\partial^2 \boldsymbol{A}_x}{\partial y^2} \\ \frac{\partial^2 \boldsymbol{A}_x}{\partial x \partial y} - \frac{\partial^2 \boldsymbol{A}_y}{\partial x^2} \end{bmatrix}$$

is constant. By looking at the first coordinate of this vector, we get

$$\partial_t \rho(\boldsymbol{A})_x = \partial_t \left(\frac{\partial^2 \boldsymbol{A}_y}{\partial y \partial x} - \frac{\partial^2 \boldsymbol{A}_x}{\partial y^2} \right) = \frac{\partial}{\partial y} \left(\frac{\partial}{\partial x} \partial_t \boldsymbol{A}_y - \frac{\partial}{\partial y} \partial_t \boldsymbol{A}_x \right)$$

By substituing $\partial_t \boldsymbol{A}_x$ and $\partial_t \boldsymbol{A}_y$ with the first form of equation 4.13:

$$\partial_t \rho(\boldsymbol{A})_x = \frac{\partial}{\partial y} \left(\frac{\partial}{\partial x} \left(\frac{\partial^2 \boldsymbol{A}_y}{\partial y^2} + \frac{\partial^2 \boldsymbol{A}_x}{\partial x \partial y} \right) - \frac{\partial}{\partial y} \left(\frac{\partial^2 \boldsymbol{A}_x}{\partial x^2} + \frac{\partial^2 \boldsymbol{A}_y}{\partial x \partial y} \right) \right) = 0$$

We proceed with the same technique for $\partial_t \rho(\boldsymbol{A})_y$ to finally show that the vector $\rho(\boldsymbol{A})$ is constant no matter its initial value. □

Bibliography

[Aan+16] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl.
"Large-Scale Data for Multiple-View Stereopsis".
In: *International Journal of Computer Vision* (2016), pp. 1–16.

[Amo+18] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter.
"Differentiable MPC for End-to-end Planning and Control".
In: *Advances in Neural Information Processing Systems*. 2018,
pp. 8299–8310.

[Ana89] Padmanabhan Anandan. "A computational framework and an algorithm for the measurement of visual motion".
In: *International Journal of Computer Vision* 2.3 (1989), pp. 283–310.

[Bak+11] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski.
"A database and evaluation methodology for optical flow".
In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31.

[Bar19] Jonathan T Barron. "A More General Robust Loss Function".
In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
2019.

[Ben+09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston.
"Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[Bes88] Paul J Besl. "Active, optical range imaging sensors".
In: *Machine vision and applications* 1.2 (1988), pp. 127–152.

[BM11] Thomas Brox and Jitendra Malik. "Large displacement optical flow: descriptor matching in variational motion estimation". In: *IEEE transactions on pattern analysis and machine intelligence* 33.3 (2011), pp. 500–513.

[Bot10] Léon Bottou.
"Large-scale machine learning with stochastic gradient descent".
In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[But+12] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black.
"A naturalistic open source movie for optical flow evaluation".
In: *European Conf. on Computer Vision (ECCV)*.
Ed. by A. Fitzgibbon et al. (Eds.) Part IV, LNCS 7577.
Springer-Verlag, Oct. 2012, pp. 611–625.

[Cad+16] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif,
Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard.
"Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age".
In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[Cas+19] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. "Unsupervised Learning of Depth and Ego-Motion: A Structured Approach". In: *AAAI-19*. 2019.

[Con19] AE Conrady. "Lens-systems, decentered". In: *Monthly notices of the royal astronomical society* 79 (1919), pp. 384–390.

[Cor+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Cor72] Stanley Coren. "Subjective contours and apparent depth." In: *Psychological Review* 79.4 (1972), p. 359.

[CP11] Antonin Chambolle and Thomas Pock. "A first-order primal-dual algorithm for convex problems with applications to imaging". In: *Journal of mathematical imaging and vision* 40.1 (2011), pp. 120–145.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[DN33] Lorente De Nó R. "Vestibulo-ocular reflex arc". In: *Archives of Neurology & Psychiatry* 30.2 (1933), pp. 245–291. URL: http://dx.doi.org/10.1001/archneurpsyc.1933.02240140009001.

[Dos+15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15.

[Eag01] David M Eagleman. "Visual illusions and neurobiology". In: *Nature Reviews Neuroscience* 2.12 (2001), p. 920.

[EPF14] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in neural information processing systems*. 2014, pp. 2366–2374.

[Far03] Gunnar Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Scandinavian conference on Image analysis*. Springer. 2003, pp. 363–370.

[FB01] B Filzek and B Breuer. "Distance behaviour on motorways with regard to active safety-a comparison between adaptive cruise control (acc) and driver". In: *Proceedings: International Technical Conference on the Enhanced Safety of Vehicles*. Vol. 2001. National Highway Traffic Safety Administration. 2001, 8–p.

[Fir16] Michael Firman. "RGBD Datasets: Past, Present and Future". In: *CVPR Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis*. 2016.

[Fra16] Assemblée nationale Française. “Article L6111-1”.
In: *Code de l’aviation civile, Livre 1er*. 2016.
URL: https://www.legifrance.gouv.fr/affichCodeArticle.do?idArticle=LEGIARTI000033304159&cidTexte=LEGITEXT000023086525.

[Fu+14] Changhong Fu, Miguel A Olivares-Mendez, Ramon Suarez-Fernandez, and Pascual Campoy. “Monocular visual-inertial slam-based collision avoidance strategy for fail-safe UAV using fuzzy logic controllers”.
In: *Journal of Intelligent & Robotic Systems* 73.1-4 (2014), pp. 513–533.

[Fu+18] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao.
“Deep Ordinal Regression Network for Monocular Depth Estimation”.
In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Fuk80] Kunihiko Fukushima.
“Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”.
In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[Gar+16] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid.
“Unsupervised cnn for single view depth estimation: Geometry to the rescue”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 740–756.

[Gei+13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun.
“Vision meets robotics: The KITTI dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[Giu+16] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. “A machine learning approach to visual perception of forest trails for mobile robots”.
In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 661–667.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”.
In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[GMAB18] Clément Godard, Oisin Mac Aodha, and Gabriel Brostow.
“Digging Into Self-Supervised Monocular Depth Estimation”.
In: *arXiv preprint arXiv:1806.01260* (2018).

[GMB17] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow.
“Unsupervised Monocular Depth Estimation with Left-Right Consistency”.
In: *CVPR*. 2017.

[GPG17] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta.
“Learning to fly by crashing”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3948–3955.

[Han+12] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud.
Time-of-flight cameras: principles, methods and applications.
Springer Science & Business Media, 2012.

[Han+15] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg.
"Matchnet: Unifying feature and metric learning for patch-based matching".
In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3279–3286.

[HCC17] Hann Woei Ho, Guido Croon, and Q Chu. "Distance and velocity estimation using optical flow from a monocular camera".
In: *International Journal of Micro Air Vehicles* 9 (Mar. 2017), pp. 198–208.
DOI: 10.1177/1756829317695566.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
"Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Hen+18] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger.
"Deep reinforcement learning that matters".
In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[Hor+13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013). Software available at http://octomap.github.com.
DOI: 10.1007/s10514-012-9321-0. URL: http://octomap.github.com.

[HS81] Berthold KP Horn and Brian G Schunck. "Determining optical flow".
In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.

[Hua+17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks".
In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean.
"Distilling the Knowledge in a Neural Network". In: *stat* 1050 (2015), p. 9.

[Ilg+16] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox.
"Flownet 2.0: Evolution of optical flow estimation with deep networks".
In: *arXiv preprint arXiv:1612.01925* (2016).

[Ilg+18] E. Ilg, T. Saikia, M. Keuper, and T. Brox.
"Occlusions, Motion and Depth Boundaries with a Generic Network for Disparity, Optical Flow or Scene Flow Estimation".
In: *European Conference on Computer Vision (ECCV)*. 2018. URL: http://lmb.informatik.uni-freiburg.de/Publications/2018/ISKB18.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift".
In: *arXiv preprint arXiv:1502.03167* (2015).

[Jan+18] Joel Janai, Fatma Güney, Anurag Ranjan, Michael J. Black, and Andreas Geiger.
"Unsupervised Learning of Multi-Frame Optical Flow with Occlusions".
In: *European Conference on Computer Vision (ECCV)*.
Vol. Lecture Notes in Computer Science, vol 11220.
Springer, Cham, Sept. 2018, pp. 713–731.

[Jég+17] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. “The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE. 2017, pp. 1175–1183.

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. “Spatial transformer networks”. In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[Kan55] G. Kanizsa. “Margini Quasi-Percettivi in Campi con Stimolazione Omogenea”. In: *Rivista di Psicologia* 49 (1955), pp. 7–30.

[KB14] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[KG17] Alex Kendall and Yarin Gal. “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in neural information processing systems*. 2017, pp. 5574–5584.

[KGC18] Alex Kendall, Yarin Gal, and Roberto Cipolla. “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7482–7491.

[Kin89] Rudolph Kingslake. “A History of the Photographic Lens, Academic Press”. In: (1989), pp. 59–62.

[KM07] Georg Klein and David Murray. “Parallel tracking and mapping for small AR workspaces”. In: *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE. 2007, pp. 225–234.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[Lai+11] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. “A large-scale hierarchical multi-view rgb-d object dataset”. In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 1817–1824.

[LB14] Matthew M. Loper and Michael J. Black. “OpenDR: An Approximate Differentiable Renderer”. In: *Computer Vision – ECCV 2014*. Vol. 8695. Lecture Notes in Computer Science. Springer International Publishing, Sept. 2014, pp. 154–169.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LH17] Brett T Lopez and Jonathan P How.
"Aggressive 3-d collision avoidance for high-speed navigation". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5759–5765.

[Li+18] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen.
"Differentiable Monte Carlo Ray Tracing through Edge Sampling".
In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (2018), 222:1–222:11.

[Lin+17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature Pyramid Networks for Object Detection".
In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 936–944.

[Lin96] Tony Lindeberg. "Scale-space: A framework for handling image structures at multiple scales". In: (1996).

[Liu+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg.
"Ssd: Single shot multibox detector".
In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[Loq+18] Antonio Loquercio, Ana I Maqueda, Carlos R del Blanco, and Davide Scaramuzza. "Dronet: Learning to fly by driving".
In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1088–1095.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell.
"Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[Mac67] J. MacQueen.
"Some methods for classification and analysis of multivariate observations".
In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*.
Berkeley, Calif.: University of California Press, 1967, pp. 281–297.
URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

[MAMT15] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós.
"ORB-SLAM: a Versatile and Accurate Monocular SLAM System".
In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163.
DOI: 10.1109/TRO.2015.2463671.

[Man+17] Michele Mancini, Gabriele Costante, Paolo Valigi, Thomas A Ciarfuglia, Jeffrey Delmerico, and Davide Scaramuzza. "Toward domain independence for learning-based monocular depth estimation".
In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1778–1785.

[MBW04] I Christopher McManus, Joseph Buckman, and Euan Woolley.
"Is light in pictures presumed to come from the left side?"
In: *Perception* 33.12 (2004), pp. 1421–1436.

[McA+17] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. "Concrete problems for autonomous vehicle safety: advantages of Bayesian deep learning". In: International Joint Conferences on Artificial Intelligence, Inc. 2017.

[MJ97] Leonard McMillan Jr. "AN IMAGE-BASED APPROACH TO THREE-DIMENSIONAL COMPUTER GRAPHICS".
PhD thesis. University of North Carolina at Chapel Hill, 1997.

[MKVR09] Tom Mertens, Jan Kautz, and Frank Van Reeth. “Exposure fusion: A simple and practical alternative to high dynamic range photography”. In: *Computer graphics forum*. Vol. 28. 1. Wiley Online Library. 2009, pp. 161–171.

[Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[Mul+06] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. “Off-road obstacle avoidance through end-to-end learning”. In: *Advances in neural information processing systems*. 2006, pp. 739–746.

[MWA18] Reza Mahjourian, Martin Wicke, and Anelia Angelova. “Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints”. In: *CVPR*. 2018.

[NE86] Hans-Hellmut Nagel and Wilfried Enkelmann. “An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5 (1986), pp. 565–593.

[NKG09] Lazaros Nalpantidis, Ioannis Kostavelis, and Antonios Gasteratos. “Stereovision-based algorithm for obstacle avoidance”. In: *International Conference on Intelligent Robotics and Applications*. Springer. 2009, pp. 195–204.

[NLD11] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. “DTAM: Dense tracking and mapping in real-time”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2320–2327.

[N.M+16] 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 International Conference on Computer Vision and Pattern Recognition (CVPR)*. arXiv:1512.02134. 2016. URL: http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16.

[Pas+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[Pin+17a] Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. “End-to-end depth from motion with stabilized monocular videos”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W3 (2017), pp. 67–74. DOI: 10.5194/isprs-annals-IV-2-W3-67-2017. URL: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W3/67/2017/.

[Pin+17b] Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. “Multi range Real-time depth inference from a monocular stabilized footage using a Fully Convolutional Neural Network”. In: *European Conference on Mobile Robotics*. ENSTA ParisTech. Paris, France, Sept. 2017. URL: https://hal.archives-ouvertes.fr/hal-01587658.

[Pin+18] Clément Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. "Learning structure-from-motion from motion". In: *ECCV GMDL Workshop*. Munich, Germany, Sept. 2018.
URL: https://hal.archives-ouvertes.fr/hal-01995833.

[PM90] Pietro Perona and Jitendra Malik.
"Scale-space and edge detection using anisotropic diffusion".
In: *IEEE Transactions on pattern analysis and machine intelligence* 12.7 (1990), pp. 629–639.

[Pre12] Lutz Prechelt. "Early Stopping — But When?" In:
Neural Networks: Tricks of the Trade: Second Edition.
Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller.
Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67.
ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_5.
URL: https://doi.org/10.1007/978-3-642-35289-8_5.

[Ran+18] Anurag Ranjan, Varun Jampani, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J. Black. "Adversarial Collaboration: Joint Unsupervised Learning of Depth, Camera Motion, Optical Flow and Motion Segmentation".
In: *CoRR* abs/1805.09806 (2018). arXiv: 1805.09806.
URL: http://arxiv.org/abs/1805.09806.

[Red+16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi.
"You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
"U-net: Convolutional networks for biomedical image segmentation".
In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning".
In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[RHW+88] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al.
"Learning representations by back-propagating errors".
In: *Cognitive modeling* 5.3 (1988), p. 1.

[Ric+78] Jacques Richalet, A Rault, JL Testud, and J Papon.
"Model predictive heuristic control".
In: *Automatica (Journal of IFAC)* 14.5 (1978), pp. 413–428.

[ROF92] Leonid I Rudin, Stanley Osher, and Emad Fatemi.
"Nonlinear total variation based noise removal algorithms".
In: *Physica D: nonlinear phenomena* 60.1-4 (1992), pp. 259–268.

[Rom+15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio.
"FitNets: Hints for Thin Deep Nets". In: *In Proceedings of ICLR*. 2015.

[Ros+13] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert.
"Learning monocular reactive uav control in cluttered natural environments".
In: *2013 IEEE international conference on robotics and automation*. IEEE. 2013, pp. 1765–1772.

[Ros58] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.”
In: *Psychological review* 65.6 (1958), p. 386.

[SC02] Jianhong Shen and Tony F Chan.
“Mathematical models for local nontexture inpaintings”.
In: *SIAM Journal on Applied Mathematics* 62.3 (2002), pp. 1019–1043.

[Sch15] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”.
In: *Neural networks* 61 (2015), pp. 85–117.

[Sch+16] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm.
“Pixelwise View Selection for Unstructured Multi-View Stereo”.
In: *European Conference on Computer Vision (ECCV)*. 2016.

[SCN08] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng.
“3-d depth reconstruction from a single still image”.
In: *International journal of computer vision* 76.1 (2008), pp. 53–69.

[Sha+17] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”.
In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065.
URL: https://arxiv.org/abs/1705.05065.

[She+96] Bhavin R. Sheth, Jitendra Sharma, S. Chenchal Rao, and Mriganka Sur.
“Orientation Maps of Subjective Contours in Visual Cortex”.
In: *Science* 274.5295 (1996), pp. 2110–2115. ISSN: 0036-8075.
DOI: 10.1126/science.274.5295.2110. eprint:
http://science.sciencemag.org/content/274/5295/2110.full.pdf.
URL: http://science.sciencemag.org/content/274/5295/2110.

[Sil+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al.
“Mastering the game of Go with deep neural networks and tree search”.
In: *NATURE* 529 (2016), p. 28.

[SK07] Kahlouche Souhila and Achour Karim.
“Optical flow based robot obstacle avoidance”.
In: *International Journal of Advanced Robotic Systems* 4.1 (2007), p. 2.

[SPC09] Frank Steinbrücker, Thomas Pock, and Daniel Cremers.
“Large displacement optical flow computation withoutwarping”.
In: *Computer Vision, 2009 IEEE 12th International Conference on*.
IEEE. 2009, pp. 1609–1614.

[SRSP16] Sebastian Stabinger, Antonio Rodriguez-Sanchez, and Justus Piater.
“Monocular obstacle avoidance for blind people using probabilistic focus of expansion estimation”. In: Mar. 2016, pp. 1–9.
DOI: 10.1109/WACV.2016.7477608.

[SS02] Daniel Scharstein and Richard Szeliski. “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms”.
In: *International journal of computer vision* 47.1-3 (2002), pp. 7–42.

[Stu+12] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers.
“A Benchmark for the Evaluation of RGB-D SLAM Systems”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*.
Oct. 2012.

[Sun+10] Wenxiu Sun, Lingfeng Xu, Oscar C Au, Sung Him Chui, and Chun Wing Kwok.
"An overview of free view-point depth-image-based rendering (DIBR)". In: 2010.

[Sun+18] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume".
In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[SZ14] K. Simonyan and A. Zisserman.
"Very Deep Convolutional Networks for Large-Scale Image Recognition".
In: *CoRR* abs/1409.1556 (2014).

[Sze+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[TLL95] Igor V Tetko, David J Livingstone, and Alexander I Luik.
"Neural network studies. 1. Comparison of overfitting and overtraining".
In: *Journal of chemical information and computer sciences* 35.5 (1995), pp. 826–833.

[Uhr+17] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. "Sparsity Invariant CNNs".
In: *International Conference on 3D Vision (3DV)*. 2017.

[Umm+17] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox.
"DeMoN: Depth and Motion Network for Learning Monocular Stereo".
In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. URL: http://lmb.informatik.uni-freiburg.de//Publications/2017/UZUMIDB17.

[Vij+17] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki.
"SfM-Net: Learning of Structure and Motion from Video".
In: *CoRR* abs/1704.07804 (2017). arXiv: 1704.07804.
URL: http://arxiv.org/abs/1704.07804.

[Wan+04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli.
"Image quality assessment: from error visibility to structural similarity".
In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[Wat89] Christopher John Cornish Hellaby Watkins.
"Learning from delayed rewards".
PhD thesis. King's College, Cambridge, 1989.

[WB18] Chaoyang Wang and José Miguel Buenaposada.
"Learning Depth from Monocular Videos using Direct Methods". In: *IEEE Conference on Computer Vision and Pattern Recognition (pp. 2022-2030).* 2018.

[WHD18] James Wilson, Frank Hutter, and Marc Deisenroth.
"Maximizing acquisition functions for Bayesian optimization".
In: *Advances in Neural Information Processing Systems*. 2018, pp. 9884–9895.

[Whi05] Turner Whitted. “An improved illumination model for shaded display”. In: *ACM Siggraph 2005 Courses*. ACM. 2005, p. 4.

[XGF16] Junyuan Xie, Ross Girshick, and Ali Farhadi. “Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 842–857.

[XOT13] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. “Sun3d: A database of big spaces reconstructed using sfm and object labels”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1625–1632.

[YS18] Zhichao Yin and Jianping Shi. “GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose”. In: *CVPR*. 2018.

[YZ15] Xiang Yu and Youmin Zhang. “Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects”. In: *Progress in Aerospace Sciences* 74 (2015), pp. 152–166.

[Zam+18] Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. “Taskonomy: Disentangling Task Transfer Learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Zha+15] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. “Is L2 a Good Loss Function for Neural Networks for Image Processing?” In: (Nov. 2015).

[Zho+17] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. “Unsupervised Learning of Depth and Ego-Motion from Video”. In: *CVPR*. 2017.

[Zin+10] Simon Zingg, Davide Scaramuzza, Stephan Weiss, and Roland Siegwart. “MAV navigation through indoor corridors using optical flow”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3361–3368.

[ZK15] Sergey Zagoruyko and Nikos Komodakis. “Learning to Compare Image Patches via Convolutional Neural Networks”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[ZL16] Jure Zbontar and Yann LeCun. “Stereo matching by training a convolutional neural network to compare image patches”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–32.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre: Apprentissage robuste d'une carte de profondeur pour l'évitement d'obstacle dans le cas des caméras volantes, monoculaires et stabilisées

Mots clés: carte de profondeur, robuste, caméra stabilisée, caméra monoculaire, reseau de neurones, apprentissage profond

Résumé: Les drones orientés grand public sont principalement des caméras volantes, stabilisées et de bonne qualité. Ceux-ci ont démocratisé la prise de vue aérienne, mais avec leur succès grandissant, la notion de sécurité est devenue prépondérante. Ce travail s'intéresse à l'évitement d'obstacle, tout en conservant un vol fluide pour l'utilisateur. Dans ce contexte technologique, nous utilisons seulement une caméra stabilisée, par contrainte de poids et de coût.
Pour leur efficacité connue en vision par ordinateur et leur performance avérée dans la résolution de tâches complexes, nous utilisons des réseaux de neurones convolutionnels (CNN). Notre stratégie repose sur un système de plusieurs niveaux de complexité dont les premières étapes sont de mesurer une carte de profondeur depuis la caméra. Cette thèse étudie les capacités d'un CNN à effectuer cette tâche.
La carte de profondeur, étant particulièrement liée au flot optique dans le cas d'images stabilisées, nous adaptons un réseau connu pour cette tâche, FlowNet, afin qu'il calcule directement la carte de profondeur à partir de deux images stabilisées. Ce réseau est appelé DepthNet.
Cette méthode fonctionne en simulateur avec un entraînement supervisé, mais n'est pas assez robuste pour des vidéos réelles. Nous étudions alors les possibilités d'auto-apprentissage basées sur la reprojection différentiable d'images. Cette technique est particulièrement nouvelle sur les CNNs et nécessite une étude détaillée afin de ne pas dépendre de paramètres heuristiques.
Finalement, nous développons un algorithme de fusion de cartes de profondeurs pour utiliser DepthNet sur des vidéos réelles. Plusieurs paires différentes sont données à DepthNet afin d'avoir une grande plage de profondeurs mesurées.

Title: Robust Learning of a depth map for obstacle avoidance with a monocular stabilized flying camera

Keywords: depth map, robust, stabilized camera, monocular camera, neural networks, deep learning

Abstract: Customer unmanned aerial vehicles (UAVs) are mainly flying cameras. They democratized aerial footage, but with their success came security concerns.
This works aims at improving UAVs safety with obstacle avoidance, while keeping a smooth flight. In this context, we use only one stabilized camera, because of weight and cost incentives.
For their robustness in computer vision and their capacity to solve complex tasks, we chose to use convolutional neural networks (CNN). Our strategy is based on incrementally learning tasks with increasing complexity which first steps are to construct a depth map from the stabilized camera. This thesis is focused on studying ability of CNNs to be trained for this task.
In the case of stabilized footage, the depth map is closely linked to optical flow. We thus adapt FlowNet, a CNN known for optical flow, to output directly depth from two stabilized frames. This network is called DepthNet.
This experiment succeeded with synthetic footage, but is not robust enough to be used directly on real videos. Consequently, we consider self supervised training with real videos, based on differentiably reproject images. This training method for CNNs being rather novel in literature, a thorough study is needed in order not to depend too much on heuristics.
Finally, we developed a depth fusion algorithm to use DepthNet efficiently on real videos. Multiple frame pairs are fed to DepthNet to get a great depth sensing range.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France