Self-Supervised Learning for Visual Obstacle Avoidance

Technical report

Tom van Dijk

TU Delft

Self-Supervised Learning for Visual Obstacle Avoidance – Technical report

Tom van Dijk
`j.c.vandijk-1@tudelft.nl`
Micro Air Vehicle Lab (MAVLab)
Faculty of Aerospace Engineering
Delft University of Technology
Delft, The Netherlands

Review: Enrico Spinielli, Guido de Croon

Keywords: Obstacle Avoidance, Stereo Vision, Depth Perception, Self-Supervised Learning, Computer Vision

Publisher: TU Delft OPEN Publishing

ISBN/EAN: 978-94-6366-509-4
DOI: 10.34641/mg.19

This work is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence, except otherwise stated.
© 2022 published by TU Delft OPEN on behalf of the authors

Disclaimer:
Every attempt has been made to ensure the correct source of images and other potentially copyrighted material was ascertained, and that all materials included in this book have been attributed and used according to their license. If you believe that a portion of the material infringes someone else's copyright, please contact `j.c.vandijk-1@tudelft.nl`.

Contributors statement
Conceptualization: Tom van Dijk, Guido de Croon
Investigation, writing and visualization: Tom van Dijk
Supervision, resources and funding: Guido de Croon
Cover image: Murat Bronz
Final review: Enrico Spinielli, Guido de Croon

Preface

In October 2018 I went through the go/no-go evaluation for my PhD studies. For this evaluation I wrote an extensive report on the results of my first year. Since I spent most of that year reviewing literature, the majority of that report consisted of a literature survey on visual obstacle avoidance, primarily focused on the practical implementation on small drones. While the report was only intended for my evaluation, I have started getting requests from MSc students to view my literature survey and it has recently had its first citation in an MSc thesis. Since this survey might also be of help to other people in the field of visual obstacle avoidance, I have decided to make this literature review public.

This technical report is an extract from my original go/no-go report, from which my personal information has been removed. The rest of the document remains unchanged, except for the removal of copyrighted images, the fixing of occasional typos, and minor corrections after having been peer reviewed. I hope that reading this report will be as useful to you as the writing of it was to me.

Tom van Dijk
Delft, March 2020

Contents

List of Acronyms **vii**

1 Problem statement **1**

2 Literature review **3**
- 2.1 Obstacle Avoidance 3
  - 2.1.1 Sensing 3
    - 2.1.1.1 Stereo vision 4
    - 2.1.1.2 Optical flow 8
    - 2.1.1.3 Appearance 13
  - 2.1.2 Avoidance 16
    - 2.1.2.1 Motion planning 16
    - 2.1.2.2 Maps 16
    - 2.1.2.3 Odometry 18
  - 2.1.3 Performance evaluation 18
- 2.2 Deep Learning for depth perception 20
  - 2.2.1 Problems in depth perception 20
    - 2.2.1.1 Depth prediction 20
    - 2.2.1.2 Depth completion 21
    - 2.2.1.3 Combined tasks 21
  - 2.2.2 Training 21
  - 2.2.3 Analysis of trained networks 22

3 Preliminary results **23**
- 3.1 Monocular depth perception 23
- 3.2 Flight tests 30
  - 3.2.1 Stop-before-obstacle with OptiTrack. 30
  - 3.2.2 Implementation of ‘embedded Visual Odometry’ 31
  - 3.2.3 Outdoor test flights 33

Bibliography **35**

List of Acronyms

CAS Collision Avoidance System

CoG Center-of-Gravity

CNN Convolutional Neural Network

EGPWS Enhanced Ground Proximity Warning System

EKF Extended Kalman Filter

ELAS Efficient Large-Scale Stereo Matching

ENAC Ecole Nationale de l’Aviation Civile

eVO embedded Visual Odometer

FoE Focus-of-Expansion

GAN Generative Adversarial Network

GDE Giga Disparity Evaluations

GPS Global Positioning System

IMU Inertial Measurement Unit

MAV Micro Aerial Vehicle

MPC Model Predictive Control

MRF Markov Random Field

P3P Perspective-3-Point

ROI Region of Interest

ROS Robot Operating System

RRT Rapidly-exploring Random Tree

SGM Semiglobal Matching

SLAM Simultaneous Localization and Mapping

SGM Semiglobal Matching

SGBM Semiglobal Block Matching

SSL Self-Supervised Learning

TCAS Traffic Collision Avoidance System

UAV Unmanned Aerial Vehicle

VO Visual Odometry

1

Problem statement

With a growing number of drones, the risk of collision with other air traffic or fixed obstacles increases. New safety measures are required to keep the operation of Unmanned Aerial Vehicles (UAVs) safe. One of these measures is the use of a *Collision Avoidance System* (CAS), a system that helps the drone autonomously detect and avoid obstacles.

The design of a Collision Avoidance System is a complex task with many smaller subproblems, as illustrated by Albaker and Rahim [1]. How should the drone sense nearby obstacles? When is there a risk of collision? What should the drone do when a conflict is detected? All of these questions need to be answered to develop a functional Collision Avoidance System. However, all of these subproblems – except the sensing of obstacles – only concern the *behavior* of the vehicle. They can be solved independently of the target platform as long as it can perform the required maneuvers; it does not matter whether it is a UAV or a larger vehicle.

The *sensing of the environment*, on the other hand, is the only subproblem that places requirements on the hardware, specifically the sensors that should be carried by the UAV. It is the hardware that sets UAVs apart from other vehicles. Unlike autonomous cars, other ground-based vehicles or larger aircraft, UAVs have only a small payload capacity. It is therefore not practical to carry large or heavy sensors such as LIDAR or radar for obstacle avoidance. Instead, obstacle avoidance on UAVs requires clever use of lightweight sensors: cameras, microphones or antennae. This literature survey will therefore focus on the *sensing* of the environment.

Out of the sensors mentioned above – cameras, microphones and antennae – cameras are the only ones that can detect nearly all ground-based obstacles and other air traffic; microphones and antennae are limited to detection of sources of noise or radio signals[1]. Therefore, this review will focus on the *visual detection of obstacles*.

The field of computer vision is well-developed; it may already be possible to find an adequate solution for visual obstacle detection using existing stereo vision methods like Semiglobal Matching (SGM) [22]. These methods, however, only use a fraction of the information present in the images to estimate depth – the *disparity*. Other cues such as the apparent size of known objects are completely ignored. The use of *appearance cues* for depth estimation is a relatively new development driven largely by the advent of Deep Learning, which allows these cues to be learned from large, labeled datasets. As long as the UAV's operational environment is similar to this training dataset it should be possible to use appearance cues in a CAS. However, this is difficult to guarantee and may require a prohibitively large training set.

Self-Supervised Learning may provide a solution to this problem. After training on an initial dataset, the UAV will continue to collect new training samples during operation. This allows it to 'adapt' to its operational environment and to learn new depth cues that are relevant in that environment. Self-Supervised Learning for depth map estimation is a young field, the first practical examples started to appear around 2016 (e.g. [16]). Most of the current literature is focused on automotive applications or on datasets captured at eye-level. It is still an open question whether Self-Supervised Learning techniques can be used for visual obstacle avoidance on UAVs.

---

[1] They could be used to detect *reflections* of sound or radio waves – this is the working principle behind ultrasonic ranging and radar – but since these are active measurements the power consumption is assumed to be too large for use on UAVs. Additionally, in the case of ultrasonic measurements the range might be too short.

2

Literature review

This chapter presents an overview of relevant literature for visual obstacle avoidance. The review consists of two parts: section 2.1 presents an overview of obstacle avoidance systems and their components, paying special attention to the visual detection of obstacles. Then, section 2.2 takes a closer look at the use of neural networks for depth estimation.

2.1. Obstacle Avoidance

Practical Collision Avoidance Systems (CAS) need to solve a number of subproblems in order to detect and avoid obstacles. An overview of the tasks involved and possible solutions is given in [1, 41]. In general, a CAS contains the following elements:

- Sensing of the environment
- Conflict detection
- Avoidance maneuver: planning and execution

The system should have some way to *sense* potential obstacles in its environment. In this review, sensing will be primarily performed through vision, but other sensors could also be used to detect obstacles. Communication with other aircraft also falls under this element. *Conflict detection* is used to decide whether an evasive maneuver should be performed. It usually requires a method to predict future states of the UAV and of detected obstacles or aircraft and a threshold or minimum safe region that should stay free of obstacles. When the conflict detection indicates that a collision is imminent, an *escape maneuver* has to be performed to avoid this potential collision. Depending on the method, this maneuver can be performed using simple rules, planned by optimizing some cost function or even performed in collaboration with other aircraft (e.g. TCAS).

The elements listed above are typical for the avoidance of other vehicles but can also be used for the avoidance of static obstacles. In this case, the conflict detection is often skipped or simplified since only the UAV itself is moving; instead it is often performed implicitly during the planning of the escape maneuver around the obstacle.

Subsections 2.1.1 and 2.1.2 take a closer look at the sensing of the environment and planning of escape maneuvers. Conflict detection is not considered for now as this review is primarily aimed at the avoidance of static obstacles. Subsection 2.1.3 will briefly highlight the literature (or lack thereof) on the performance evaluation of Collision Avoidance Systems.

2.1.1. Sensing

The goal of sensing is to detect and locate nearby obstacles. The range of sensors that could be used for obstacle detection is large, but a number of these can be ruled out for usage on UAVs because of their weight or power requirements. This subsection will focus on the visual detection of obstacles.

The localization of obstacles through vision can be split into two parts: estimation of the bearing towards the obstacle and estimation of the distance. As long as the obstacle can be reliably found in

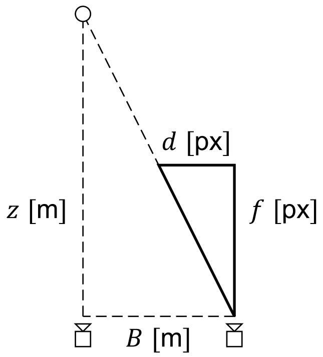


Figure 2.1: Stereo vision uses the disparity d to estimate the depth z towards obstacles. The camera baseline B and focal length f are constant and obtained through calibration.

the image, estimation of the *bearing* is fairly straightforward. The position of the obstacle in the image is a direct result of its bearing relative to the camera and this relation can be inverted.

Estimation of the *distance* towards the obstacle is more complicated. Since the obstacle is projected onto the image plane, the depth information is lost. Other cues need to be used to estimate the distance towards the obstacle. These cues can be broadly split into three categories:

- Stereo vision
- Optical flow
- Appearance

Stereo vision uses two images taken *at the same time* from different locations, optical flow uses two images taken *at different times* and appearance is based on *single* images. The next subsections take a closer look at these depth estimation methods.

2.1.1.1. Stereo vision

Stereo vision uses images taken at the same time from different viewpoints to estimate depth. The difference in viewpoints causes the obstacle to appear in different positions in the images. The difference in these positions – the *disparity* – is inversely related to the depth of the object.

An example of depth estimation using stereo vision is shown in Figure 2.1 for an obstacle at distance z observed using a stereo camera with focal length f and baseline B. Using equal triangles, the disparity of the obstacle is:

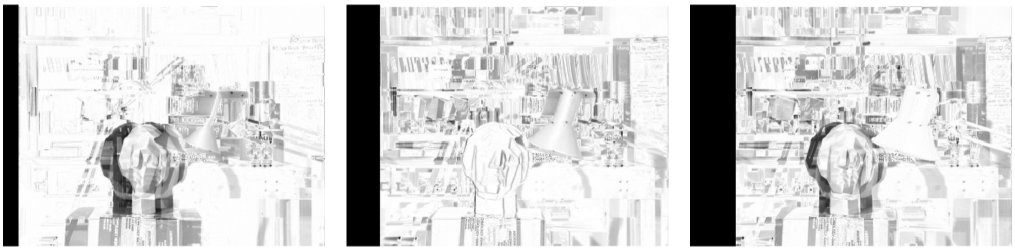
$$d = B\ f\ z^{-1} \tag{2.1}$$

This equation can be solved for z to find a distance estimate $\hat{z}$ given a disparity d:

$$\hat{z} = B\ f\ d^{-1} \tag{2.2}$$

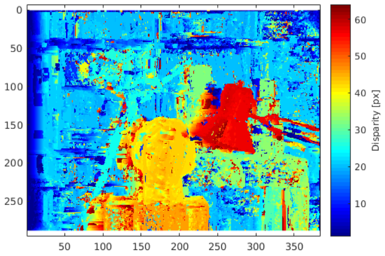
The camera parameters B and f are found beforehand through calibration.

The main challenge of stereo vision is to find this disparity; it is often difficult to find out which pixels in the images belong to the same point in the world. A first way to categorize stereo vision algorithms is to make a distinction between sparse and dense algorithms. *Sparse* algorithms estimate the disparity of a small number of highly recognizable points in the images. The disparity accuracy tends to be good as these points are easy to match, but because only a small number of points is considered the resulting depth map can contain large holes, especially in environments with little texture. Sparse stereo algorithms are therefore a poor choice for obstacle detection, but they sometimes appear as part of Visual Odometry (VO) or Simultaneous Localization and Mapping (SLAM) algorithms. *Dense* algorithms, on the other hand, estimate the depth for the entire image. They should therefore be able to estimate the distance towards all obstacles in view.

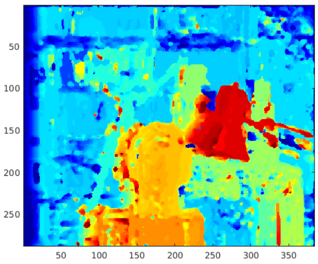


(a) *Matching cost computation*. The matching cost is calculated per pixel for all disparities under consideration. In this example the pixel difference is used as matching cost. Shown are difference images at three different disparities, where white indicates a low matching cost and black a high cost. In the left image, the disparity is roughly equal to the true disparity of the background: the background has a low matching cost (white). In the middle image, the disparity is close to that of the head; in the right image it is close to that of the lamp.

(b) *Cost aggregation*. Sometimes individual pixels can be hard to match. In this case, information on neighboring pixels can make the matching task easier. In this example, the matching cost images are convolved with a 3×3 averaging filter to take nearby pixels into account.



(c) *Disparity optimization*. Using the aggregated matching cost, the per-pixel disparity can be found through optimization. In this example, the per-pixel argmax over the disparities is used. This is a form of *local* optimization as the pixel disparities can be found independently.



(d) *Disparity refinement*. Post-processing is used to clean up the disparity map from the previous step. In this example, a median filter is used to remove outliers.

Figure 2.2: Example of the *block matching* stereo algorithm broken down into the four steps described by Scharstein and Szeliski [50]. Input image: *Tsukaba*, Middlebury Stereo Vision Dataset 2001 [50] courtesy of the University of Tsukuba.

In [50], Scharstein and Szeliski present an extensive taxonomy of dense stereo vision algorithms. According to the authors, most dense stereo vision algorithms perform the following steps to find a disparity map:

1. Matching cost computation
2. Cost aggregation
3. Disparity optimization
4. Disparity refinement

An example of these four steps is shown in Figure 2.2 for the *block matching* algorithm.

An important distinction can be made between global and local algorithms, which differ in the way the disparity optimization is performed. *Global* algorithms try to optimize a single cost function that depends on all pixel disparities. These algorithms can produce accurate depth maps even for scarcely textured scenes, but tend to be slower than local algorithms. *Local* algorithms independently optimize the disparities of pixels or small regions. These algorithms are easier to parallelize and typically faster, but less accurate.

An in-depth review of stereo vision methods is out of scope for this report. While it is important to understand the way these stereo vision algorithms work, their run-time performance and accuracy are perhaps more relevant for their use on UAVs. These are difficult to predict from first principles and are instead measured on benchmarks, of which the Middlebury Stereo benchmark[1] [50] and the KITTI Stereo benchmark[2] [34] are commonly-used examples.

In [56], Tippetts et al. perform an extensive review of stereo vision algorithms for resource-limited systems. The authors collected run-time and accuracy measurements for a large number of algorithms and use these to produce scatterplots of their performance. Where possible, the run-times were normalized based on the hardware for which they were reported. The article provides an excellent starting point for the selection of stereo algorithms, its only downside being that it was written in 2012 and that it is therefore not fully up-to-date.

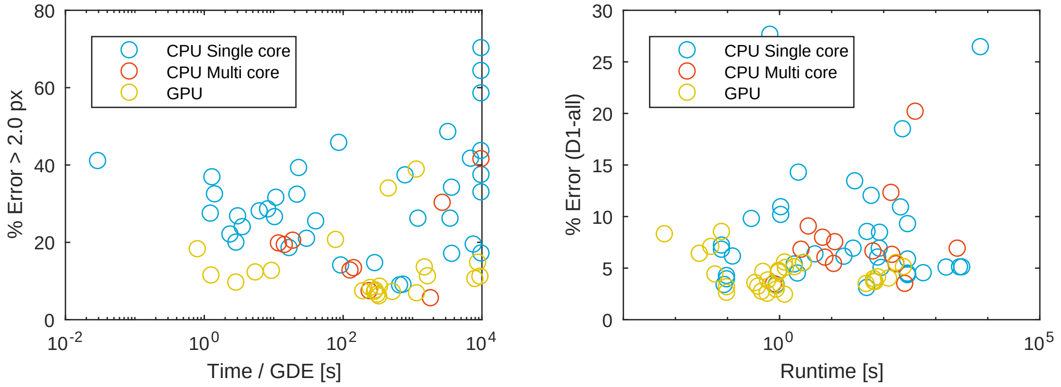
A similar review was performed for this literature study, so that algorithms published after 2012 could also be included. Run-time and accuracy measures were obtained from the Middlebury and KITTI benchmarks. Run-time figures were not normalized, as the majority of methods are evaluated on similar platforms (CPU-based methods on an unspecified 2.5 GHz processor, GPU-based methods on an NVIDIA Titan X). The main focus of this comparison is on algorithms for which code is publicly available. The results are shown in Figure 2.3.

The following conclusions are drawn from these results: first of all, there exist close-to-optimal stereo vision algorithms for which code is publicly available. This means that it is not necessary to write an own implementation of a state-of-the-art algorithm. Secondly: from the CPU-based methods, ELAS [17] and SGM/SGBM variants [22] are still among the best performers. The inclusion of SGBM in OpenCV makes this an ideal algorithm for initial development. Thirdly: the use of a GPU can significantly increase performance, mainly in terms of accuracy. However, it is currently unclear how this performance improvement weighs up against the increase in weight and power consumption of such a platform. The 250 W power required by the NVIDIA Titan X is quite high for a UAV, and the performance benefit seen in the benchmarks might be significantly smaller on an embedded GPU.

---

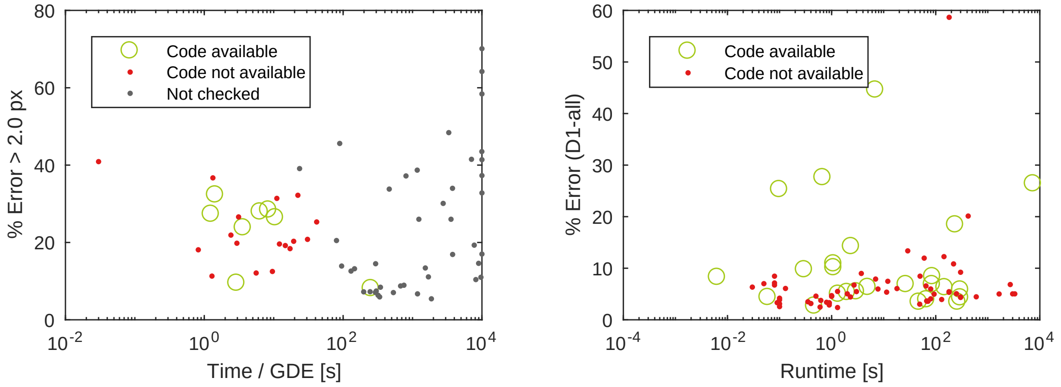
[1] `http://vision.middlebury.edu/stereo/`

[2] `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo`



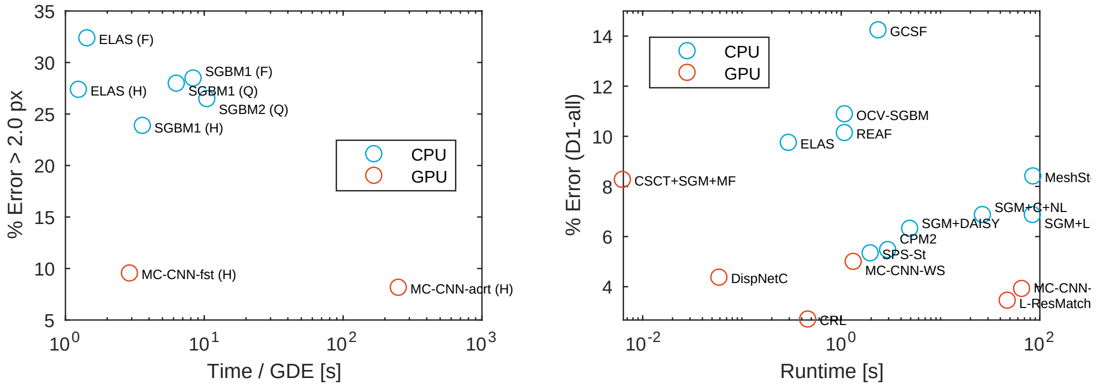
(a) Platforms and performance on the Middlebury benchmark.

(b) Platforms and performance on the KITTI benchmark.



(c) Code availability on the Middlebury benchmark.

(d) Code availability on the KITTI benchmark.



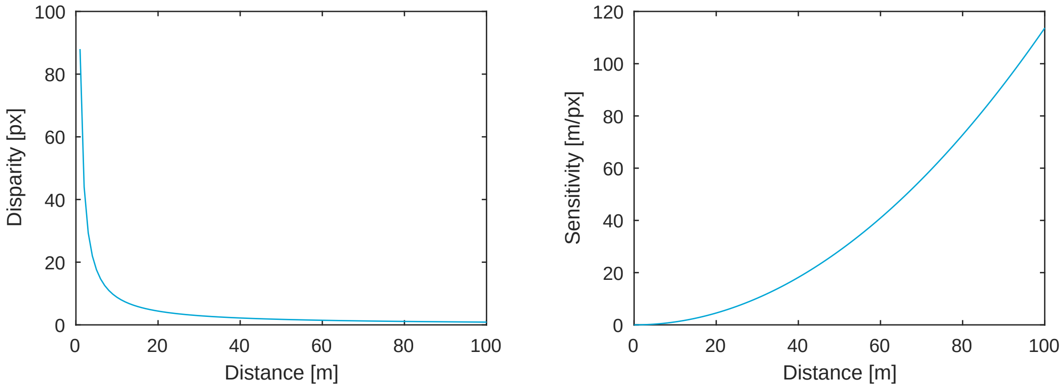
(e) Best performing methods on the Middlebury benchmark for which code is available.

(f) Best performing methods on the KITTI benchmark for which code is available.

Figure 2.3: Scatterplots of accuracy versus runtime performance on the Middlebury[1] and KITTI[2] stereo vision benchmarks. Runtime on the Middlebury benchmark is measured in seconds per Giga Disparity Evaluations (GDE), which is found by multiplying the image width, height and maximum number of disparities. Data obtained on 27/11/2017. *a, b*: Methods running on the GPU tend to perform better than those running on the CPU. On the Middlebury benchmark they perform better in terms of accuracy, while on the KITTI benchmark they also outperform CPU methods in terms of runtime – perhaps because runtime performance is more important for automotive applications than for the static pictures of Middlebury. *c, d*: While code is not available for every method, there are enough close-to-optimal algorithms for which source code has been published. *e, f*: These methods should be considered first when choosing a stereo vision algorithm, as they perform well and their code is publicly available. Popular choices are ELAS [17] and SGM/SGBM [22]; the latter is also included in OpenCV.

[1]: `https://vision.middlebury.edu/stereo/` [50]

[2]: `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo` (CC BY-NC-SA 3.0)



(a) Disparity vs. distance. As the distance increases, the disparity converges to zero. For far-away objects the disparity hardly changes with distance anymore.

(b) Sensitivity vs. distance. The sensitivity is defined as $-\mathrm{d}z/\mathrm{d}d$, i.e. the distance error for a 1 px error in the disparity estimate.

Figure 2.4: Maximum range of stereo vision. As the distance increases, the sensitivity to stereo matching errors increases quadratically. Example plots generated for a camera with baseline $B = 20\,\mathrm{cm}$ and focal length $f = 400\,\mathrm{px}$. Best-case disparity errors are in the order of 0.5 px to 0.1 px [43] depending on the algorithm.

On a higher level, stereo vision has the advantage over other depth cues that its depth estimate is based on the *baseline* between the two cameras. This is an advantage because the baseline is constant and easy to measure or calibrate. In comparison, the distance between successive images for optical flow is often unknown; it has to be estimated and therefore leads to more uncertainty in the distance estimate. Appearance cues have a similar disadvantage, the size of certain cues in the environment is not exactly known, also leading to uncertainty in the depth estimate.

Stereo vision also has limitations. First of all it requires two or more cameras. The resulting weight will be larger for this setup than for depth estimation based on optical flow or appearance cues.

Secondly, the range of stereo vision is limited, although not as badly as commonly thought [43]. As the distance to obstacles increases, the disparity decreases inversely (see Figure 2.4). This means that for far-away objects the disparity hardly changes with distance. As a result, the sensitivity to measurement errors $\mathrm{d}\hat{z}/\mathrm{d}d$ increases with distance until it becomes impractically large:

$$\frac{\mathrm{d}\hat{z}}{\mathrm{d}d} = -B\;f\;d^{-2} \tag{2.3}$$

$$= -\frac{z^2}{B\;f} \tag{2.4}$$

This growing uncertainty limits the maximum range of stereo vision. The disparity errors are the result of incorrect matching of pixels in the input images and are typically independent of distance. If the stereo algorithm only searches for discrete disparities, these errors will be in the order of 0.5 px at best. Stereo algorithms for long-range distances therefore need to estimate subpixel disparities. According to Pinggera et al., it is possible to reach a consistent error limit of 0.1 px under real-world conditions [43]. The sensitivity to measurement errors can also be reduced by increasing the baseline B or focal length f of the cameras.

Finally, the matching of features between the input images is often a weak point of stereo vision. As a result, it may perform badly with the following obstacles: textureless surfaces, finely or repetitively textured surfaces, textures oriented parallel to the baseline, reflections and transparency. Furthermore, depending on the algorithm, slanted surfaces and occlusions can be problematic.

2.1.1.2. Optical flow

Optical flow tracks the movement of image features over time. In a static environment, the shift of these features depends on the movement of the camera and the distance to the features; in general, features further away from the camera will move less than those that are nearby. If the movement of the camera is known, the distance to the features can be obtained. When only the rotation is known the distance cannot be found; however it is still possible to estimate the time-to-contact, which is sufficient for some forms of obstacle avoidance.

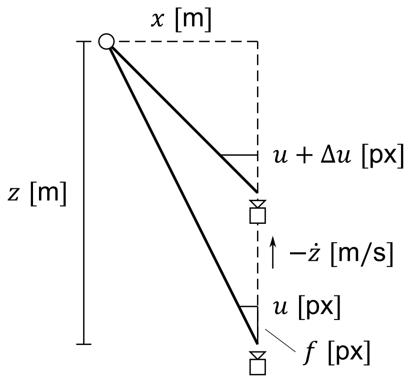


Figure 2.5: Optical flow for forward motion. The image position u of an obstacle located at (x, z) changes as the UAV moves forward with a velocity of $-\dot{z}$.

Figure 2.5 shows an example of optical flow and its use for depth estimation. The example assumes forward motion[3] at a known velocity without rotation of the camera. Given the obstacle's position (x, z) and the camera's focal length f, its image position u can be found using equal triangles:

$$u = x\ f\ z^{-1} \tag{2.5}$$

Taking the time derivative produces the instantaneous optical flow $\dot{u}$ of the obstacle or feature:

$$\begin{aligned} \dot{u} &= -x\ f\ z^{-2}\ \dot{z} && (2.6) \\ &= -u\ z^{-1}\ \dot{z} && (2.7) \end{aligned}$$

In practice, however, the optical flow is estimated between two images separated by a time interval Δt. The result is a shift in position Δu instead of the flow $\dot{u}$:

$$\begin{aligned} \Delta u &\approx \dot{u}\ \Delta t && (2.8) \\ &\approx -u\ z^{-1}\ \dot{z}\ \Delta t && (2.9) \end{aligned}$$

The depth $\hat{z}$ can be found by solving this equation for z:

$$\hat{z} = -u\ \dot{z}\ \Delta t\ \Delta u^{-1} \quad \forall \Delta u \neq 0 \implies \forall u \neq 0 \tag{2.10}$$

and, if velocity $\dot{z}$ is not available, the time-to-contact τ is found using:

$$\begin{aligned} \tau &= \hat{z} / \dot{z} && (2.11) \\ &= -u\ \Delta t\ \Delta u^{-1} && (2.12) \end{aligned}$$

Note, however, that from Equation 2.7 and 2.9 it follows that the flow $\dot{u}$ and shift Δu will be zero in the center of the image where u is zero (the Focus-of-Expansion). It is therefore not possible to estimate depth at the Focus-of-Expansion as the result is undefined.

The main problem of optical flow is not the estimation of depth but the tracking of features between images. It is therefore very similar to stereo vision. The main difference, however, is that stereo vision only searches for matches along one dimension, while optical flow is two-dimensional. Optical flow is therefore more difficult to compute.

As in stereo vision, a distinction can be made between sparse and dense optical flow algorithms. *Sparse* algorithms track highly recognizable points, typically corners. Sparse tracking is frequently found in VO or SLAM. Like sparse stereo vision, sparse optical flow is not suitable for obstacle detection as it may leave large holes in the depth map. *Dense* algorithms estimate optical flow for the complete image and are therefore better suited for obstacle detection.

---

[3] Optical flow from sideways or vertical motion has slightly different characteristics, but will not be explained here to keep the explanation short. A forward-facing camera is the most relevant example for obstacle avoidance.

An overview of optical flow techniques is presented in [2]. The survey is similar to [50] in that it breaks down the algorithms into a few key components. According to Baker et al., most dense optical flow algorithms perform a global optimization (i.e. for all pixels at the same time) of the following energy function: $E_{\text{data}} + \lambda E_{\text{prior}}$, where the *data term* E_{data} follows from the content of the images (similar to the matching cost in stereo vision) and the *prior term* E_{prior} encodes assumptions of the flow field such as its smoothness [2]. The final component of a dense optical flow algorithm is the *optimization algorithm*.

An in-depth overview of optical flow algorithms is again beyond the scope of this report. Instead, existing optical flow algorithms are compared by benchmark results. The results are obtained from Baker et al., 2011 [2] (the more up-to-date Middlebury website[4] unfortunately does not report run-times) and from the KITTI optical flow 2015 benchmark[5] [34]. The results are shown in Figure 2.6.

The KITTI results show that code is available for fast and accurate optical flow estimation on GPUs. Code for the best performing CPU-based algorithms is not available; SPyNet [45] might be used as an equally fast alternative, but it has a higher error percentage than the best-performing algorithms. The other CPU-based algorithms for which code is available have run-times larger than one second. While it may be possible to reduce their run-times by, for instance, lowering the resolution of the images, there is no guarantee that they will run fast enough for practical use in obstacle avoidance. From the results of Baker et al., 2011 only FOLKI has a run-time of one second, while the others are in the order of ten seconds or more.

Compared to stereo vision, the main advantage of optical flow is that it only requires a single camera, which saves weight. However, optical flow also has a number of disadvantages. First of all, if a metric depth estimation is required, the velocity of the UAV should be known. Estimation of this velocity is not trivial and uncertainties in this estimate are an additional source of error for depth estimation.

A second problem is that the optical flow approaches zero near the FoE. By definition the FoE lies in the direction of travel, exactly the place where obstacles *should* be detected. Since the flow needs to be inverted to estimate distance, this makes the depth estimate extremely sensitive to measurement errors in shift Δu. This is demonstrated with the sensitivity $\mathrm{d}\hat{z}/\mathrm{d}\Delta u$, i.e. the error in the distance estimate for a 1 px error in Δu, which is found by differentiating Equation 2.10 with respect to Δu:

$$\frac{\mathrm{d}\hat{z}}{\mathrm{d}\Delta u} = u\ \dot{z}\ \Delta t\ \Delta u^{-2} \tag{2.13}$$

$$= \frac{z^2}{\dot{z}\ u\ \Delta t} \tag{2.14}$$

For reference, the best average end-point errors in the KITTI optical flow 2012 benchmark[6] [18] lie in the order of 1 px. The expected flow and sensitivity are shown in Figure 2.7 for a drone traveling at 10 m/s. The conclusion drawn from this figure is that it may be difficult to get an adequate measurement range near the FoE, as the sensitivity to measurement errors rapidly increases for $|u| < 100$ px.

Equation 2.14 suggests a few ways to reduce the sensitivity to errors. First of all, the UAV can fly faster; this results in larger flow vectors relative to the measurement error. Secondly, the frame rate can be *reduced*, this will also increase the size of the flow vectors. Note, however, that there is an upper limit to Δt as the resulting Δu should remain small enough that features remain in view. The frame rate should also remain high enough to detect obstacles in time. Finally, the sensitivity can be reduced by using a higher-resolution camera or a zoom lens, as u will be larger (note that the sensitivity does not depend on the camera's focal length).

The final disadvantage of optical flow is that it requires sufficient texture to match pixels between successive images. Like stereo vision, it can produce incorrect results for textureless surfaces, finely or repetitively textured surfaces, reflections and transparency.

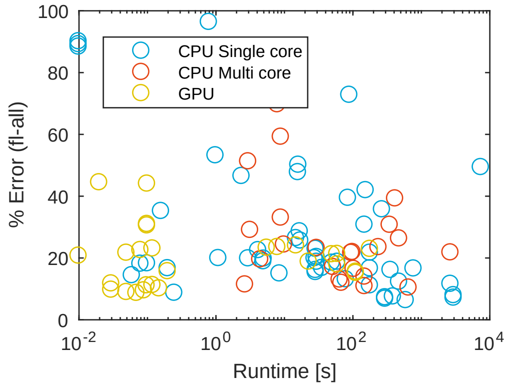
Not mentioned in this review is *scene flow*, the 3D equivalent of optical flow. The result of scene flow is a 3-dimensional velocity vector for each pixel, together with a depth or disparity. A review of this field is left for future work.

---

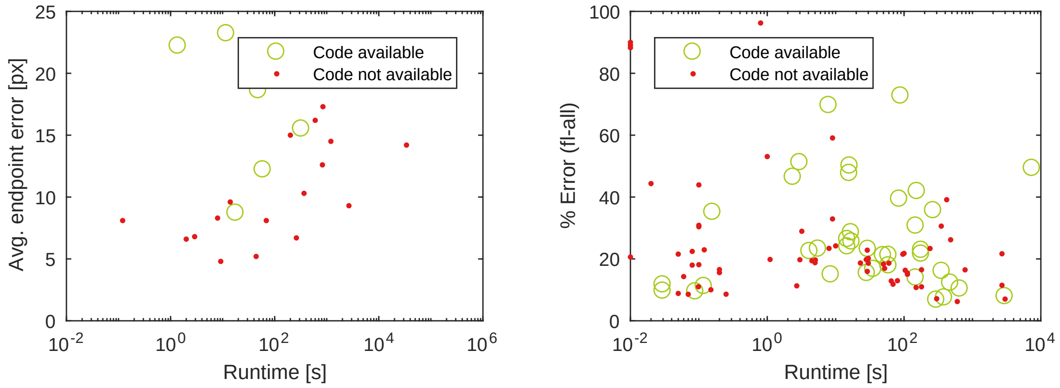
[4] `http://vision.middlebury.edu/flow/`

[5] `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow`

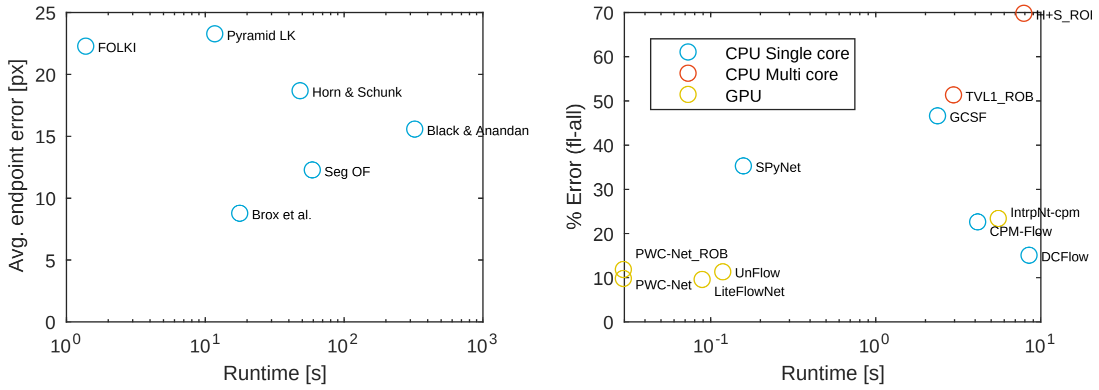
[6] `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=flow`



(a) Platform and performance on the KITTI benchmark.



(b) Code availability of the methods reviewed in Baker et al., 2011. (c) Code availability on the KITTI benchmark.

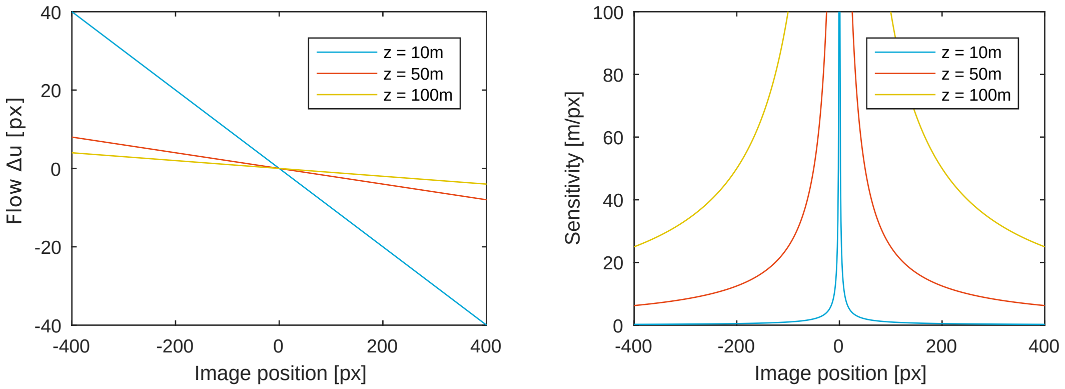


(d) Best performing methods in Baker et al., 2011 for which code is available. (e) Best performing methods on the KITTI benchmark for which code is available.

Figure 2.6: Scatterplots of dense optical flow estimation accuracy and run-time performance. Data is obtained from Baker et al., 2011[1] [2] and the KITTI optical flow 2015 benchmark[2] [34] (data obtained on 28/08/2018). *a*: GPU-based methods tend to have lower runtimes and error percentages than CPU-based algorithms. (Platform information is not available for Baker et al., 2011). *b, c*: Of the methods listed in Baker et al., 2011, source code is not available for the best performing ones. This is slightly better for the KITTI benchmark. *d, e*: These are the best performing methods for which code is publicly available. GPU-based methods perform significantly better than CPU-based ones. There is little overlap between Baker et al., 2011 and KITTI in terms of algorithms, but note that there is a 7-year gap between the two benchmarks. The algorithm by Brox et al. is included in OpenCV [4]. For the other methods code is available, but it might take more work to integrate these into research code.

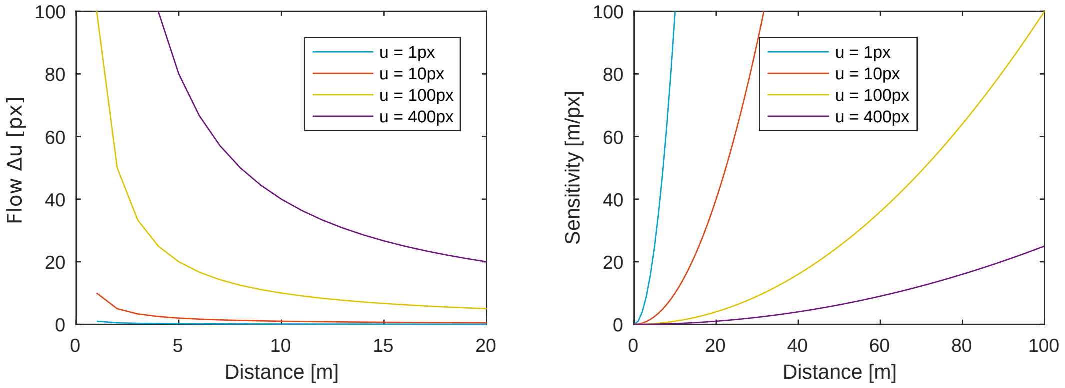
[1]: Baker et al., 2011 [2] (CC BY-NC 2.0)

[2]: `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=flow` (CC BY-NC-SA 3.0)



(a) Expected optical flow Δu as a function of image position u for three obstacle distances z.

(b) Sensitivity to measurement errors in Δu as a function of image position u for three obstacle distances z.



(c) Expected optical flow Δu as a function of obstacle distance z for four image positions u.

(d) Sensitivity to measurement errors in Δu as a function of obstacle distance z for four image positions u.

Figure 2.7: Example of expected optical flow and sensitivity to measurement errors in Δu. Data generated for a camera traveling at 10 m/s with an optical flow algorithm running at 10 Hz ($\Delta t = 0.1$ s). Plot *b* shows that the sensitivity to errors strongly increases near the center of the image and approaches infinity at the Focus-of-Expansion (FoE). Plot *d* shows that obstacles near the FoE ($u = 1$ px) can only be detected at short ranges where the sensitivity to measurement errors is low, while the range is significantly larger near the edge of the image ($u = 400$ px).

2.1.1.3. Appearance

Unlike stereo vision or optical flow, appearance cues can be found inside a *single* image. As humans we are already familiar with appearance-based cues because we use them all the time, such as when looking at photographs. Photographs do not contain disparities since they are flat, nor do they produce optical flow as they do not move. Still, it is possible to estimate depth from these images; this is the field of *monocular depth estimation*.

'Appearance' is not really a single cue, as is the case for stereo vision which relies entirely on disparities or optical flow which results only from the flow vectors. Instead, appearance cues are a collection of image features that depend in one way or another on depth. An extensive treatment of depth cues used by humans can be found in [19]. The following is a non-exhaustive list of appearance cues:

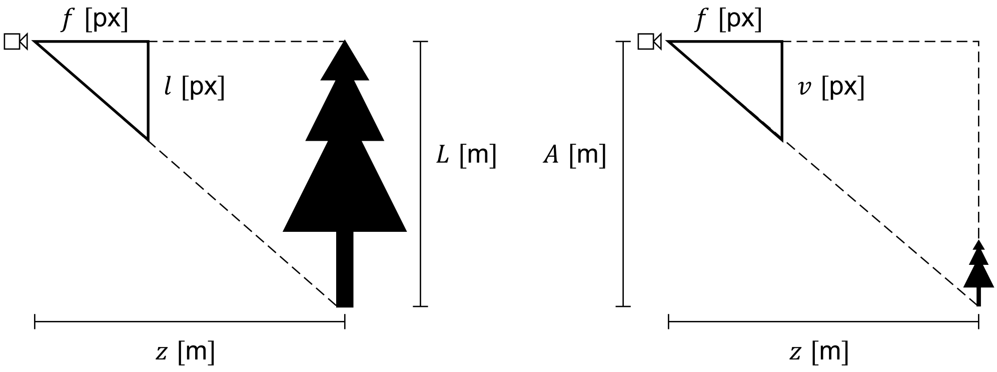
- Occlusion. Nearby objects cover those further away.
- Image size of known objects. Using the focal length of the camera, this can be transformed back into a distance estimate.
- Different image sizes of similar objects. The objects that appear smaller in the image are further away.
- Perspective. Parallel lines in the environment appear to converge in the image, their distance provides an indication of depth.
- Vertical image position. Objects that appear higher in the image are further away.
- Texture gradient. Surface textures will appear more fine-grained if they are further away.
- Light and shadow. This cue is especially relevant for surface relief. Light typically comes from above, brighter regions are assumed to face upwards.
- Atmospheric haze. Far-away objects take on a blue-ish tint.
- Sky segmentation. The sky is infinitely far away.

Most of these cues require knowledge about the environment, such as the presence of a flat ground or parallel lines, knowledge about the size of objects, and so on. This makes appearance-based depth estimation more difficult to implement than stereo vision or optical flow. If it is even possible to implement some of these cues, this quickly leads to rather ad-hoc solutions. For this reason, appearance-based cues have seen relatively little use in computer vision until recently.

One of the first practical examples of monocular depth estimation for arbitrary outdoor images is Saxena et al.'s *Make3D* [47, 49], first published in 2006. The system relies on a combination of superpixel segmentation and hand-crafted features. These are fed into a Markov Random Field (MRF) to model the relations between the regions in the image. Another example of monocular depth estimation using classical machine learning methods, this time based on the texture gradient cue, is given in [58].

The field of monocular depth estimation really took off with the arrival of Deep Learning. Using Convolutional Neural Networks (CNNs), it is no longer necessary to develop feature descriptors by hand. Instead, these features and the relations between them are learned from a large dataset of example images. Eigen et al. are the first to use a CNN for monocular depth estimation in [8, 9]. Their network is trained on color images labeled with the true depth map obtained with a Kinect (NYU Depth v2) or LIDAR (KITTI). The first example of *Self-Supervised Learning* for depth estimation is published in 2016 by Garg et al. [16]. Instead of training to predict a depth map, their CNN is trained to predict the other image in a stereo pair. Deep learning has made it possible to use appearance for depth estimation by taking away the need to manually implement an estimator for these cues. Section 2.2 will go into more detail on Deep Learning for depth estimation.

Appearance-based depth estimation has the advantage that it only requires a single camera. Unlike optical flow, however, it can work without an estimate of the UAV's velocity. Secondly, appearance-based depth estimation relies on different features than stereo vision and optical flow. As a result, appearance cues may work better for obstacles where the previous algorithms are likely to fail. Appearance-based depth estimation could therefore be a valuable addition for depth estimation, but this is not yet proven. Whether obstacle avoidance will truly benefit from appearance cues is still an open question.



(a) Depth estimation using the known size L of an object and its image size l.

(b) Depth estimation using the vertical image position v and the altitude A of the UAV.

Figure 2.8: Two examples of depth estimation based on appearance features.

The main disadvantage of appearance-based depth perception is that it is inaccurate, especially with regards to scale. Monocular depth perception lacks a reliable reference length by which the scene can be scaled. In stereo vision this is provided by the baseline between the cameras; in optical flow by the distance between the two images. In monocular depth estimation, the only obvious source of this information is the known size of objects, but this has to be learned from the training set and may vary between different object instances.

The depth scale, however, is not the only problem of monocular depth estimation. The relative depth between objects also suffers from large inaccuracies. This is effectively demonstrated by Smolyanskiy et al. in [53]. The authors show that the depth map produced by *MonoDepth* [20] looks visually correct; however, an overhead view of the resulting point cloud shows that this is clearly not the case. It is not clear whether this is a limitation of MonoDepth or its training set, or a more fundamental issue with monocular depth estimation.

Estimating the sensitivity to measurement errors of appearance cues is a bit more difficult than for stereo vision or optical flow as the cues are not always clearly defined or based on simple geometry. An attempt is made to model the uncertainty of the two depth cues: the size of known objects in the image and the vertical position of objects in the image. These examples are shown in Figure 2.8. Using equal triangles, the image size l of an object can be found as follows:

$$l = f\ L\ z^{-1} \tag{2.15}$$

Similarly, given the drone's height A above the terrain, the vertical position v in the image is found using:

$$v = f\ A\ z^{-1} \tag{2.16}$$

Note that these equations are exactly the same when $L = A$ and $l = v$. For brevity only the first cue will be discussed in more detail, the results also apply to the second case.

Equation 2.15 can be solved for z to produce a depth estimate $\hat{z}$:

$$\hat{z} = f\ L\ l^{-1} \tag{2.17}$$

There are two sources of uncertainty in this equation. First of all, there may be a small error in the length measurement l in the image. Sensitivity to this error is found to be:

$$\begin{aligned} \frac{\partial \hat{z}}{\partial l} &= -f\ L\ l^{-2} \quad (2.18) \\ &= \frac{z^2}{f\ L} \quad (2.19) \end{aligned}$$

While this sensitivity also increases quadratically with distance, its magnitude remains relatively small compared to the errors of stereo vision or optical flow: when observing an object with size $L = 10\,\text{m}$

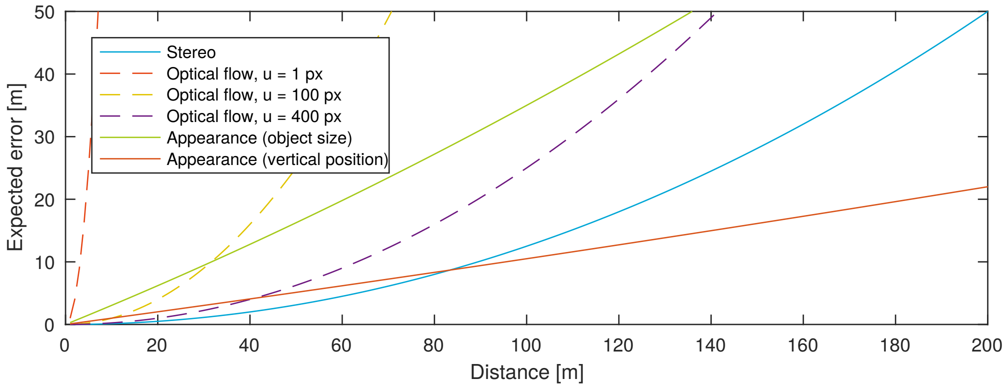


Figure 2.9: Comparison of expected error bounds for selected depth estimation methods.

Table 2.1: Parameters used to generate Figure 2.9. Error bounds ϵ_l, ϵ_L and ϵ_A are an educated guess, the bounds ϵ_d and $\epsilon_{\Delta u}$ are based on literature and the KITTI benchmark.

B	20 cm	ϵ_d	0.1 px
f	400 px	$\epsilon_{\Delta u}$	1.0 px
$\dot{z}$	10 m/s	ϵ_l	2 px
Δt	0.1 s	ϵ_L	3 m
L	10 m	ϵ_A	10 m
A	100 m		

(e.g. a tree, or the length or wingspan of a Cessna 172) at a distance of 100 m with a focal length of $f = 400$ px, the sensitivity to length measurement errors is only 2.5 m/px, compared to $\sim$ 100 m/px for a stereo camera with baseline $B = 20$ cm and the same focal length.

The second source of error is uncertainty about the object's true size L. Sensitivity to these errors is found as follows:

$$\frac{\partial \hat{z}}{\partial L} = f\ l^{-1} \tag{2.20}$$

$$= \frac{z}{L} \tag{2.21}$$

Note that unlike all error sensitivities found before, this one only grows linearly with distance. This suggests that appearance-based depth estimation might have an advantage over stereo vision or optical flow at longer distances, as long as the error in the image length measurement l remains sufficiently small.

Sensitivity to errors in the image length measurement (Equation 2.19) can be reduced with a larger focal length f. There is, however, no way to reduce the sensitivity to errors in L (Equation 2.21), as L mainly depends on the object that happens to be in front of the UAV.

In the example of the vertical image position, however, L is equal to the altitude of the drone. This altitude is mostly likely larger than the size of objects the drone will encounter, which means this depth estimation method will be more accurate than using the size of the object. Secondly, the sensitivity to errors can in this case be reduced by flying higher, thereby increasing L.

This section on sensing is concluded with a comparison of the expected errors of stereo vision, optical flow and appearance-based depth estimation. The expected error is calculated by multiplying the sensitivity (e.g. $\mathrm{d}\hat{z}/\mathrm{d}d$) with an estimated upper bound on said error (e.g. $\epsilon_d = 0.1$ px for stereo vision with subpixel disparities). Note that this is only a first-order approximation of the error, the results may not be realistic as the expected error approaches or exceeds the true distance z. A comparison chart of the depth estimation methods is shown in Figure 2.9. The parameters used to generate this chart are listed in Table 2.1.

While the results should be taken with a grain of salt, they do highlight the trends found in this literature review. The error of optical flow is prohibitively large near the center of the image ($u = 1$ px and 100 px), but comparatively decent near the edge of the image ($u = 400$ px). The error could be reduced by flying faster, a speed of 10 m/s was assumed for this comparison. Stereo vision appears to be the best choice in this scenario for obstacles up to a distance of $\sim$ 80 m. Unlike optical flow, however, this depth estimate should also be accurate near the center of the image. The result plotted here is based on a stereo vision algorithm that can estimate subpixel disparities. Finally, at larger distances the depth estimate based on the vertical position of objects performs best, due to its predominantly linear increase in sensitivity to measurement errors.

2.1.2. Avoidance

When an obstacle is detected along the UAV's direction of travel, it should perform an avoidance maneuver to prevent a collision. There are different ways to handle this, from the very simple and lightweight reflexive behaviors, to high-level planning in maze-like environments.

The execution of an avoidance maneuver typically requires the following components: 1) *motion planning*, which determines the actions the UAV should take; 2) a *map*, a representation of the obstacles in the vicinity of the UAV and 3) *odometry*, which is often required to accurately perform the planned maneuver. These components will be briefly discussed in the following subsections.

2.1.2.1. Motion planning

Motion planning determines the action the UAV should take to avoid collisions while moving towards its goal location. An overview of motion planning and obstacle avoidance algorithms can be found in [21, 35].

Minguez et al. [35] make a distinction between global planning and local planning (called 'motion planning' and 'obstacle avoidance' in their article, these terms will not be used here to avoid confusion with the overall task of obstacle avoidance). *Global planning* assumes that the location of all obstacles is known, the goal is to find a trajectory that optimizes a given performance measure. *Local planning* assumes that only obstacles detected by the UAV's sensors are known. The goal here is to adapt the current trajectory of the UAV to avoid a collision with nearby obstacles. Local planning has the disadvantage that it can get trapped in certain situations (mazes for example, but these situations are unlikely in outdoor flight). However, unlike global planning it can function in unknown environments. Local planning is therefore the most relevant for UAV obstacle avoidance.

Motion planning algorithms can be broadly divided into the following classes: reactive planning, planning without dynamics and planning with dynamics. *Reactive planning* refers to a class of algorithms that prescribe a control input or motion based directly on the presence of obstacles. An example is the use of potential fields to determine the direction of travel of the UAV: detected obstacles 'repel' the drone, preventing a collision. In *planning without dynamics* the goal is to find a path for the UAV that guides it past the detected obstacles. This path should also minimize a cost function, setting these algorithms apart from reactive planning. Once a path is found, it is left to a lower-level controller to actually follow it. An example is [33] where a Rapidly-exploring Random Tree (RRT) is used to plan a path through a forest. *Planning with dynamics* also optimizes a cost function, but includes a dynamic model of the UAV. Model Predictive Control (MPC) is an example of this. The inclusion of dynamics ensures that the maneuver can actually be performed, but requires a dynamic model of the UAV to be available. The use of dynamics is particularly suitable for high-performance manneuvers (e.g. drone racing), while planning without dynamics is more suitable for general-purpose applications as it does not require a model.

For brevity this section only lists examples of algorithms. The reader is referred to the cited reviews for a more extensive overview of methods.

2.1.2.2. Maps

Motion planning requires a map, but the exact function of the map differs per algorithm. At the very least, the map serves to document the location of nearby obstacles; even reactive planning will need this information. For more complicated algorithms, the map allows the planning of an avoidance maneuver around the obstacles. Finally, a map allows multiple observations of obstacles to be combined, which is the basic idea behind SLAM.

Maps can be made at different levels of detail. Ground robots and autonomous cars often create highly detailed maps of their immediate surroundings. These types of maps are also applicable to UAVs flying at low altitudes or indoors, but their creation is computationally intensive. An example of

Table 2.2: Overview of common properties of map types. This table only lists the typical properties of these maps; exceptions can likely be found for many entries in this table. Note that combinations of these maps are possible (e.g. a cartesian voxel map for static obstacles and an EKF for the positions of other aircraft).

	Image-space	Discretized space (voxels)		Continuous space	
		Cartesian	Polar	Point cloud	Obstacle positions
Computational complexity	Low	High	High	High	Low
Volumetric	2.5D[a]	Yes	Yes	No[b]	No[c]
Probabilistic	No	Occupancy	Occupancy	Position	Position
Dynamic	No	No	No	Yes	Yes
Single-frame	Yes	Yes	Yes	Yes	Yes
Multi-frame	No	Yes	No	Yes	Yes
Reference frame	Body	World	Body	Any	Any

[a] Volumetric in horizontal and vertical directions but not in depth.

[b] It is possible to fit a mesh on the point cloud or assume a small, fixed volume around each point.

[c] A fixed volume can be assumed for the obstacle, if known.

less detailed maps for aircraft is the Enhanced Ground Proximity Warning System (EGPWS), which uses relatively coarse-scaled static maps to prevent terrain collisions on passenger aircraft. Such a map could also be used on UAVs as a form of geofencing, but this would primarily apply to cruise flight as such a static map is difficult to keep up to date at a high enough level of detail for take-offs and landings.

Table 2.2 lists map types that could be used to model the immediate surroundings of the UAV during flight. The maps are divided into three classes: image-space maps, discretized space maps and continuous space maps. *Image space* maps are essentially the same as depth maps: they consist of pixels for which the distance towards the first obstacle is stored. *Discretized space* maps split the surroundings of the drone into a collection of discrete cells that can be free or occupied. These maps are commonly used for range-sensor-based SLAM on indoor robots. Finally, *continuous space* maps do not discretize the space around the UAV, but store a continuous position estimate for each measurement point. A point cloud is a typical example of this map, but it is also possible to track the position of entire objects.

The table furthermore lists the following properties: *computational complexity* gives an indication of the amount of processing power and memory required to build and maintain the map. Single-frame maps are relatively lightweight, as are filter-based maps if they have small covariance matrices (e.g. an EKF of obstacle positions). Maps that are constructed from multiple image frames tend to require more processing and memory. The next properties describe the information a map may contain: *volumetric* maps describe non-zero volumes of space, such as voxels. On the other hand, non-volumetric maps such as point clouds model obstacles using infinitesimally small points. Additional processing (e.g. mesh- or surface fitting, or expanding the points by a certain volume) of these maps is required before they can be used to test for collisions. *Probabilistic* maps can explicitly model uncertainty; either in the occupancy of parts of the environment, or in the position estimate of points or obstacles. *Dynamic* maps can also model the velocities of obstacles. Continuous-space maps are particularly suitable for this, as they can be updated over multiple frames without introducing quantization errors by storing intermediate states into a discretized map.

In principle all of the maps in Table 2.2 can successfully be used on UAVs, but it depends on the application which map is the most suitable. The most important decision is whether the map should combine multiple measurements (*multi-frame*) or represent only a single measurement (*single-frame*). Combining multiple measurements allows the drone to map large and complex environments; it is therefore particularly suited for indoor operations but its use is limited to larger drones as the underlying algorithms can be computationally intensive. Cartesian voxel maps are a common choice for this application (e.g. [32, 51]). If the environment is simple enough that it can be captured in a single measurement, then image-space maps are a logical choice as these require very little processing to create and because other map types do not provide additional advantages if they do not fuse multiple measurements. An example of the use of an image-space map for UAV obstacle avoidance is found in [33]. For the avoidance of other aircraft, a continuous-space map is a good choice as such a map is easy to update and can also model the velocity of the other aircraft. An Extended Kalman Filter (EKF) with the states of the detected aircraft is an example of such a map.

2.1.2.3. Odometry

To perform all but the most basic avoidance maneuvers, the UAV will need an estimate of its velocity. Outdoors GPS is often available, but reflections can make it inaccurate in densely built areas. Indoors, GPS is not available for navigation so a different solution needs to be found.

A common solution for GPS-less flight is Visual Odometry (VO), where a camera is used to estimate the velocity of the drone. The simplest methods directly transform the optical flow from a bottom-facing camera into a velocity estimate; this is commonly combined with sonar measurements to provide a sense of scale. More complex VO algorithms are closely related to SLAM but lack loop closure capabilities. These algorithms often estimate the UAV's pose relative to a *keyframe*. The use of a keyframe instead of the integration of velocities prevents drift over time; errors only accumulate when new keyframes are created.

VO algorithms can be separated into dense and sparse algorithms, and direct and indirect methods. A good description of these categories is provided in the introduction of [12]. The dense and sparse attributes are similar to those in stereo vision and optical flow: sparse algorithms only track a small number of keypoints, while dense algorithms use the entire input image. Direct and indirect refers to the way that keypoints are matched or tracked: direct methods rely only on the intensities of neighboring pixels, while indirect methods first need to construct feature descriptors.

Both monocular, stereo and RGB-D vision can be used for VO. Stereo and RGB-D have the advantage that the map can be initialized from a single observation; this is not the case for monocular VO as one observation can only provide the bearing of the keypoints. Since a depth map is already required for obstacle avoidance, it should also be used for VO.

Another design consideration is the use of an Inertial Measurement Unit (IMU). IMUs measure accelerations and angular velocities, which can be integrated to track the drone's pose. Additionally, it can provide an estimate of the gravity vector. The IMU typically has a higher update rate than the camera and is also not sensitive to the appearance of the environment. The integration of small measurement errors, however, causes the pose estimate to drift over time – especially in the horizontal plane [61]. It is therefore not practical to rely solely on the IMU, it needs to be fused with other measurements like VO. There are two approaches to the fusion of IMU data with VO: tight coupling and loose coupling. With tight coupling, the IMU measurements are used in the same filter that performs the visual pose estimation, for instance in the update step of an EKF. With loose coupling, the vision-based pose estimate is calculated separately, after which a second filter is used to fuse it with the IMU measurement. Tight coupling produces more accurate results, but loose coupling might be easier to implement with existing autopilot filters. A second use of the IMU is in feature tracking. The IMU can be used to predict the next position of keypoints; this estimate can reduce the search space for tracking. An example of this principle can be found in [46].

An overview of VO methods is presented in Table 2.3. UAV obstacle avoidance applications should prefer methods that use stereo or RGB-D input together with the IMU.

2.1.3. Performance evaluation

Once a Collision Avoidance System (CAS) has been implemented, its performance should be evaluated. The literature review on this subject can be kept brief: hardly any literature exists on this topic. Most articles on obstacle avoidance demonstrate their method in an example application, but there is no common benchmark on which they can be compared.

A first step towards such a benchmark was taken in [39]. One of the core ideas of this paper is that the obstacle avoidance task can be split into smaller subtasks that can be evaluated independently. For instance, the accuracy of obstacle detection can be evaluated independently from the UAV's motion planning algorithm or state estimator.

The main difficulty in the development of a benchmark is to find suitable metrics to describe the avoidance problem: the metrics should be chosen such that different environments with the same metrics (e.g. obstacle density, typical obstacle size) result in the same behavior. It should be possible to predict the performance of an obstacle avoidance system when all relevant metrics of the target environment are available. Such a benchmark would be extremely valuable both for UAVs and other types of robots.

Table 2.3: Overview of VO algorithms.

Method	Camera	IMU	S/D	Descriptor	Code available	Platform
DSO [12]	Mono	–	Sparse	Direct	`https://github.com/JakobEngel/dso`	CPU, laptop 5× real-time
SVO2 [14]	Mono (incl. fish-eye, catadioptric), stereo	Yes (optional)	Sparse	Direct	Binary only, incl. armhf. `http://rpg.ifi.uzh.ch/svo2.html`	Laptop, smartphone (100fps), MAV
ORB-SLAM2 [36]	Mono, stereo, RGB-D	–	Sparse	ORB	`https://github.com/raulmur/ORB_SLAM2`	Core i7 real-time
Usenko et al. [57]	Stereo	Yes	Semi	Direct	–	–
OKVIS [28]	Mono, stereo	Yes	Sparse	BRISK	`https://github.com/ethz-asl/okvis`	Real-time (platform not mentioned)
ORB-SLAM [37]	Mono	–	Sparse	ORB	`https://github.com/raulmur/ORB_SLAM`	–
ROVIO [3]	Mono	Yes	Sparse	Direct	`https://github.com/ethz-asl/rovio`	CPU 1-core 33fps
SVO [13]	Mono	No	Sparse	Direct	`https://github.com/uzh-rpg/rpg_svo`	Embedded on MAV 55fps, Laptop 300fps
Schmid et al. [51]	Stereo	Yes	Sparse	Direct	–	MAV
LSD-SLAM [10, 11]	Mono (incl. fish-eye, catadioptric), stereo	No	Dense	n/a	`https://github.com/tum-vision/lsd_slam` (without stereo support)	CPU 40× real-time
eVO [46]	Stereo	Gyro only	Sparse	Direct	–	CPU 1.86 MHz 2-core 12–56 ms per frame
DVO [25]	RGB-D	Yes (optional)	Dense	n/a	`https://github.com/tum-vision/dvo_slam`	CPU 1-core 30fps
MSCKF 2.0 [29]	Mono, stereo	Yes	Sparse	Direct	3rd-party implementations on github	CPU i7 2.66 GHz 1-core 10 ms per frame
DTAM [38]	Mono	–	Dense	n/a	openDTAM	GPU
PTAM [26]	Mono	–	Sparse	Direct	`http://www.robots.ox.ac.uk/~gk/PTAM/`	–

A similar lack of benchmarks exists for robot navigation. A proposal for a navigation benchmark is found in [54], perhaps this paper can also serve as inspiration for an obstacle avoidance benchmark.

2.2. Deep Learning for depth perception

Because of strict weight constraints, UAV obstacle detection is strongly dependent on vision. While earlier vision algorithms had to be designed and tuned by hand, the arrival of Deep Learning allows depth estimation to be learned from large datasets. This section presents an overview of recent literature and developments in the field of depth perception. Since the first application of a Convolutional Neural Network (CNN) for depth perception in 2014 [9] this field has been rapidly evolving. This is also illustrated by the articles cited in this section, as the majority of them were uploaded to ArXiv between June 2018 and now. Each month, roughly ten new relevant papers appear on ArXiv.

Section 2.2.1 describes different depth perception tasks. Section 2.2.2 will discuss the training of these networks including a brief overview of commonly used datasets. Finally, section 2.2.3 presents some works on the analysis of networks after they have been trained.

2.2.1. Problems in depth perception

While the goal of depth perception is clear – the estimation of a depth map from input images – there are a few ways this problem is formulated in literature. The most common problem that is solved in literature is *depth prediction*: generating a depth map using only one or more input images. A second problem in literature is that of *depth completion*. In this case, a partial depth map is already available, such as the depth towards VO or SLAM keypoints. The goal of the neural network is then to fill in the missing parts of the depth map. Finally, recent literature has shifted towards the *combination of depth perception with other tasks* in the same network. For instance, a single network performs both depth estimation and object segmentation. The next subsections look more closely at these problems.

2.2.1.1. Depth prediction

The goal of depth prediction is the estimation of a depth map given only one or more RGB images. The field of *monocular depth prediction* uses only one image for its depth estimate. The first CNN for monocular depth prediction was presented in 2014 by Eigen et al. [9]. This network was trained on images labeled with true depth maps. Because these maps are hard to obtain, Garg et al. [16] developed the first network that used unsupervised learning. Training is performed by predicting the *other* image from a stereo pair; it is no longer necessary to collect true depth maps. Godard et al. [20] proposed further improvements to this technique. The recently published *PyD-Net* (July 2018) can run at ~2 Hz on a Raspberry Pi 3 CPU and still produce competitive results [44].

While methods [16, 20] do not require true depth labels, they still need to use a stereo camera to collect training data for monocular vision. As a result, these methods cannot be used for on-board training of monocular vision. An alternative to these approaches is to train on monocular image sequences. Examples of this approach are [24, 59, 65].

While it may seem redundant at first, it is also possible to perform depth prediction on stereo images. The advantage of this over ‘normal’ stereo vision methods such as SGM is that the neural network can also learn to include appearance cues. These provide additional depth information that is not provided by just the disparities. An example of deep learning for stereo vision is found in [64], where Self-Supervised Learning (SSL) is used to learn stereo vision from scratch. After training, the network can compete with existing state-of-the-art algorithms.

Compared to monocular vision, stereo vision has the advantage that a reliable reference distance is available: the baseline between the two cameras. As a result, depth estimates from stereo vision are more accurate than those from monocular vision. This point is strongly argued by Smolyanskiy et al., who state that any application that relies on accurate depth estimates and that can carry more than one camera should do so [53]. The use of a stereo camera should be possible on all UAVs as even the ~20 g DelFly can carry a small stereo camera. The only reason the preliminary work in chapter 3 still looks at monocular vision is that this allows appearance cues to be examined in isolation from disparity or flow cues.

2.2.1.2. Depth completion

Where depth prediction uses only RGB images, *depth completion* assumes that some sparse depth information is available. This information can come, for instance, from the depth of VO keypoints. In literature, LIDAR is also commonly mentioned as a source of sparse depth measurements.

Ma and Karaman [30] implement a network that uses sparse depth information and then compare its performance to monocular depth estimation. They come to the interesting conclusion that even a depth map generated from only 20 sparse depth measurements *without RGB images* already has a higher accuracy than the monocular depth estimation networks of [8, 27]. Note that this comparison is based on scale-aware metrics; the scale-invariant error [9] is not reported so it is not possible to say whether the relative distances are incorrect or that the monocular methods only suffer from a scaling error. Nevertheless, the experiments show that sparse depth measurements can be a valuable addition for depth estimation. The authors also check whether the use of RGB images in addition to sparse depth estimates leads to further accuracy improvements: this is primarily the case for low numbers of depth measurements, at higher numbers there is also an increase in accuracy but it is small. The work of Ma and Karaman is continued in [31]; other recent examples of depth completion are [6, 23, 60]. No examples were found where sparse depth completion is combined with or compared to stereo vision.

The good results from depth completion lead to an interesting design choice: is it better to perform depth prediction and use the results for VO, or to use VO to collect sparse measurements and use these to estimate a depth map? A third option has also appeared in recent literature: use a single network to predict both depth and pose from image sequences.

2.2.1.3. Combined tasks: depth, pose, flow, segmentation, ...

Recently a growing number of articles is appearing on networks that combine depth estimation with other tasks. Common combinations are depth with pose, segmentation and/or optical flow. There are a few potential advantages to combining these techniques in a single network: if filters can be shared between tasks, this might lead to a lower total number of parameters. Secondly, combining multiple tasks can potentially improve learning as the depth estimation is encouraged to use the (intermediate) results of, for instance, object segmentation and vice versa. A review of these networks is left for future work; it is therefore not possible to confirm these advantages in this report.

2.2.2. Training

Training is an essential component of Deep Learning. For depth estimation, two types of training are common in literature: supervised and unsupervised (also called self-supervised). Earlier examples of monocular depth estimation (e.g. [9]) rely on *supervised learning*. The network is trained to replicate a true depth map that belongs to the input image. This depth map is typically obtained using a LIDAR sensor or an RGB-D sensor (e.g. Microsoft's Kinect). An advantage of supervised learning is that in most cases a true depth value is available for every pixel. The major disadvantage, however, is that it requires an additional sensor to capture the true depth of the scene. For this reason, supervised learning cannot be used on-board a UAV; all training has to be performed offline.

In *unsupervised learning*, the true depth map is not available. Instead, unsupervised learning often depends on a reconstruction error, where for instance the other images in a stereo pair are predicted and compared to the true images (e.g. [16]). The advantage of unsupervised learning is that the training data is easier to collect. Since no additional sensor is required, learning can also be performed online, allowing the UAV to adapt to its environment during operation. 'Unsupervised learning' is a bit of a misnomer as the methods primarily rely on supervised training methods. The argument to call them unsupervised is that no labeled data has to be provided from an external source. Unsupervised learning is the same as *Self-Supervised Learning (SSL)*, but this term does not introduce ambiguity about the learning method, while it still makes it sufficiently clear that the supervision is already provided by the input data. Therefore, only 'SSL' will be used in this report.

Recent articles have started to use Generative Adversarial Networks (GANs) for depth perception (e.g. [5, 42]). In the GAN framework, a second network (the *discriminator*) is trained to distinguish the network's output from the training label. The depth perception network (the *generator*) and discriminator are trained in alternation. In this framework the discriminator essentially replaces the loss function, but unlike the loss function it is trained specifically for the (last version of) the generator network and can therefore provide a more precise measure of its performance.

GANs can be used in both supervised and self-supervised settings. In the former ([5]), it compares the generated and true depth maps; in the latter ([42]) it compares reconstructed and true images. In both papers the accuracy exceeds that of common benchmark papers.

The (off-line) training of a neural network requires an appropriate dataset. The use of publicly available dataset also allows a quantitative comparison between methods. Commonly used datasets are the KITTI stereo dataset[7] and the NYUv2 dataset[8] [52]. The KITTI dataset is aimed at automotive applications; the images are obtained from a stereo camera and LIDAR mounted on the front of a car. The NYUv2 dataset contains RGB-D images captured in indoor environments. Other frequently-used datasets are Make3D[9] [47–49] and the Cityscapes dataset[10] [7].

Instead of using data captured in the real world, it is also possible to generate these from a simulation. Examples of generated datasets are vKITTI[11] [15] and Synthia[12]. Training data can also be generated during closed-loop simulation. An example of this is Microsoft's AirSim[13] for UAVs and autonomous cars. An advantage of simulation is that the actual depth of all pixels is directly available. The disadvantage is that the generated images differ from those captured in the real world – the *reality gap*. In [63] Zheng et al. propose to use a GAN to reduce the difference between real and simulated images. The resulting network can outperform [9] but not [16, 20] when subsequently evaluated on real datasets.

2.2.3. Analysis of trained networks

While there are many articles on deep learning for depth perception, no articles were found on *how* the trained networks perform this task. There is a small number of articles that focus on the analysis of CNNs in general. In [62] Zeiler and Fergus use unpooling and deconvolution operations to map neuron activities back to the input space. Given an input image, this technique produces an image that highlights the regions that cause a strong activation of a selected neuron. This technique focuses on single neurons, but note that [55] argues that it is the space spanned by multiple neurons can be more informative than individual neuron activations. In a more recent paper Olah et al. [40] present a highly detailed (interactive) overview of visualization techniques. This article provides a good starting point for further research into neural network visualization.

The cited papers examine generic CNNs at a rather low level. No articles were found that examine the high-level behavior of networks for depth perception. How exactly do these networks estimate depth? This information is essential in order to predict the behavior of these networks on other platforms – UAVs in this case. Therefore, chapter 3 presents the first steps towards a high-level understanding of these networks.

---

[7] `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo`

[8] `https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html`

[9] `http://make3d.cs.cornell.edu/data.html`

[10] `https://www.cityscapes-dataset.com/`

[11] `http://www.europe.naverlabs.com/Research/Computer-Vision/Proxy-Virtual-Worlds`

[12] `http://synthia-dataset.net/`

[13] `https://github.com/Microsoft/AirSim`

3

Preliminary results

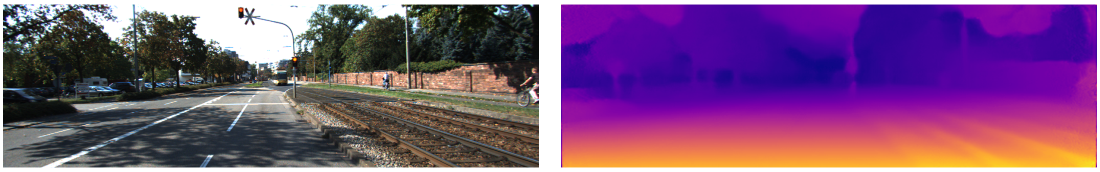


Figure 3.1: Monocular depth estimation with *MonoDepth* [20]. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

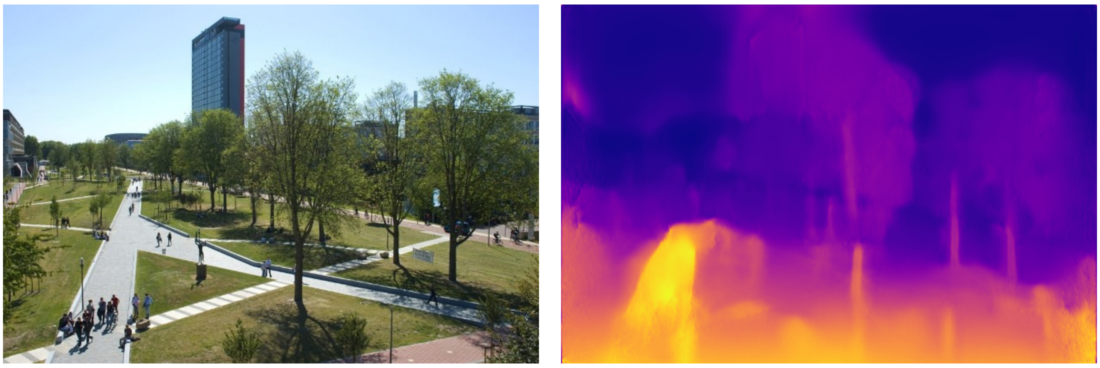


Figure 3.2: The MonoDepth network does not transfer well to different viewpoints. The road on the left is seen as a nearby obstacle. The high-rise building of EWI appears closer than the trees in front of it. Input image: *Aanzicht Mekelpark met studenten* by D. Brinkman, © Delft University of Technology, `https://repository.tudelft.nl/view/MMP/uuid:cb952de1-34e8-4db3-a577-9177a62581ed?fullscreen=1`.

3.1. Monocular depth perception

The goal of this research is to use SSL to improve obstacle avoidance on UAVs. SSL will be used for depth estimation as the need to use vision to sense the environment sets UAVs apart from other vehicles. The first question to be asked is whether SSL-based depth estimation can actually be used on a UAV. At first sight this may seem obvious: why would it not work on a UAV? However, results indicate that this might not be as simple as it appears.

This chapter presents experiments performed on the *MonoDepth* network [20]. MonoDepth is a Self-Supervised monocular depth estimation network that is trained on the KITTI stereo vision dataset. The network predicts disparities such that these minimize a reconstruction error between two images of a stereo pair. On images in the KITTI dataset the network performs quite well (Figure 3.1). However, when the network is used on images taken from a different viewpoint (Figure 3.2), the accuracy of the depth map quickly degrades.

Clearly, a network trained on a dataset of automotive images cannot be transferred directly to a UAV. Most likely it is possible to get MonoDepth to work on a UAV by training it on a suitable dataset. However, that does not explain why the network trained on KITTI fails. The results on KITTI images show that the network *can* estimate depth, but apparently it does so using image features that do not work on UAVs. To guarantee correct behavior it is important to know what these features are and under what circumstances they are learned.

While there is a large and increasing number of articles on monocular depth perception, there is not a single paper that analyses what these networks actually learn. This experiment is a first step towards an *understanding* of monocular depth perception as learned by neural networks. The goals of this experiment are:

- Provide insight into monocular vision. While useful for UAVs, this insight will also be extremely valuable for automotive applications. With an understanding of the inner workings of monocular depth perception, it becomes easier to predict its behavior and to make guarantees about its correctness.
- Provide insight into the use of monocular vision on UAVs. The results should explain why the network trained on KITTI does not transfer well. The same experiments can then be performed on a network trained on a UAV dataset, and the differences can be compared.
- The features learned by MonoDepth might be replicated using simpler, lightweight algorithms. This would enable the use of monocular vision on embedded hardware and tiny UAVs, perhaps even the DelFly.

The problem with neural networks like MonoDepth is that they are black boxes. It is difficult to analyze what is going on inside them. In that respect there is some overlap with human depth perception, which is also difficult to take apart piece-by-piece. For neural networks there are techniques to visualize the functions of individual neurons, but this is a very low-level form of analysis. Instead, the experiments presented here will analyze the behavior of the entire network using the time-tested scientific method: coming up with *hypotheses* on how the network could estimate depth, then designing experiments that *test* whether these hypotheses are true.

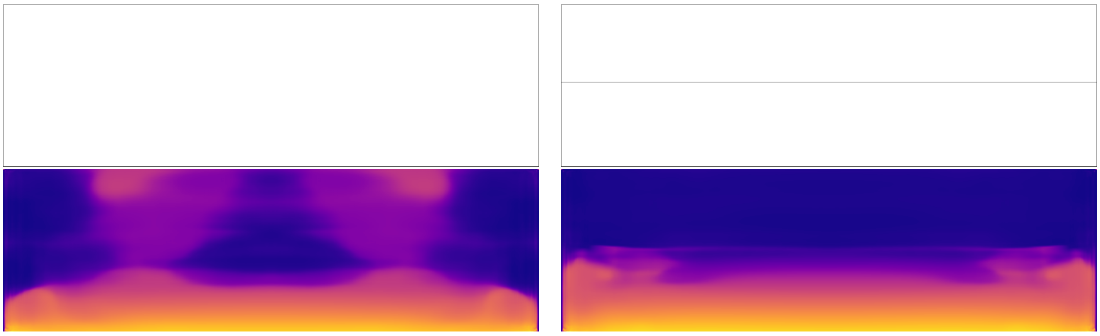
So what would be a good hypothesis on the inner workings of MonoDepth? The experiments performed in this chapter test the following points:

- MonoDepth assumes there is a flat ground in front of the vehicle.
- MonoDepth uses color to detect the sky.
- MonoDepth estimates the distance towards obstacles using one or more of the following features:
  - Apparent size of known obstacles.
  - Vertical position in the image.

MonoDepth is trained on the KITTI dataset, which contains images taken from a forward-facing camera on a car. The camera has a fixed attitude and height and in the majority of the images there is a free section of road in front of the car. Rather than detecting the road, MonoDepth can just assume it is there as this will be true for nearly all images in the training set. It is hypothesized that MonoDepth assumes the presence of a flat ground rather than detecting it. The ground's depth estimate will be 'overwritten' by any obstacles it detects.

The upper half of the image consists of sky and obstacles. The sky seems easy to detect using its color or brightness and would result in a large number of correct pixels; it is therefore assumed that MonoDepth does just that, possibly combined with a prior expectation of seeing the sky in the upper half of the image.

With the ground and sky detected, the rest of the depth map consists of obstacles. Following the list of appearance cues in subsubsection 2.1.1.3, likely candidates for depth estimation towards obstacles are their apparent size and vertical position as both seem relatively easy to measure, although the former also requires knowledge about the true size of various types of obstacles.



(a) MonoDepth's response to a completely white input image. Although there is a hint of a ground surface, the depth map (especially the top half) contains a lot of garbage.

(b) Addition of a single thin line is enough to make MonoDepth detect a floor and sky, or at least assume they are there.

Figure 3.3: MonoDepth has strong prior expectations about the presence of a flat ground in the lower half of the image and sky in the upper half. (Outlines added for visibility, these are not part of the input images.)

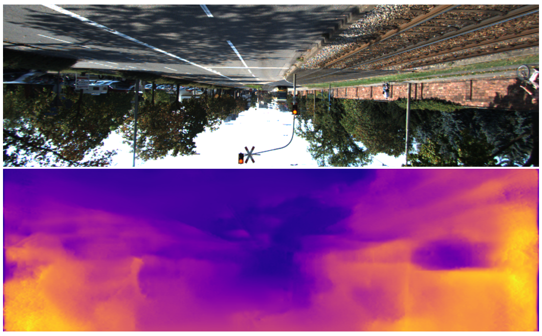
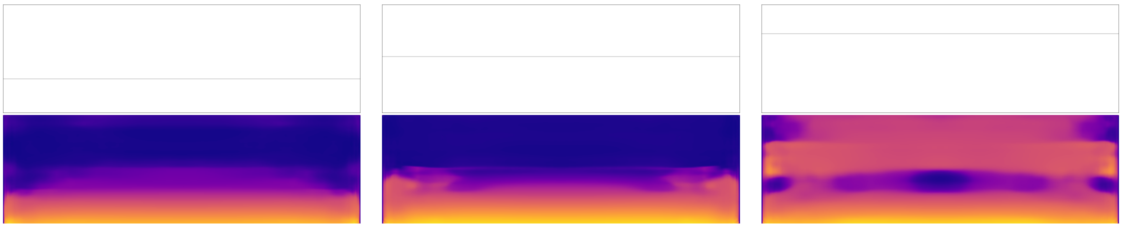


Figure 3.4: When the image is flipped vertically, the trees and sky in the lower half of the image are assumed to be closer than the road in the upper half. If MonoDepth did not have prior expectations about a flat ground, the disparity map would also have flipped vertically. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

If MonoDepth indeed assumes the presence of a flat ground, it should be possible to make it 'see' a ground surface even if it isn't actually there. This is attempted in Figure 3.3. First, MonoDepth is presented with a completely blank input image to see if it will guess the presence of ground and sky. This is not entirely the case. However, when a thin horizon line is added to the image, MonoDepth is suddenly able to detect a floor and sky. Since the input image contains no features other than a single line, these can only come from MonoDepth's prior expectations.

This prior expectation can also be demonstrated on real photographs, as shown in Figure 3.4. In this figure, a vertically flipped image from the KITTI dataset is presented to MonoDepth. If MonoDepth would not assign a high prior probability to the presence and location of the ground and sky, the resulting depth map would also be a flipped version of the original depth map. This is, however, not the case. Instead, the resulting depth map still assumes that the lower half of the image is closer than the upper half, confirming the presence of this bias in the estimation.

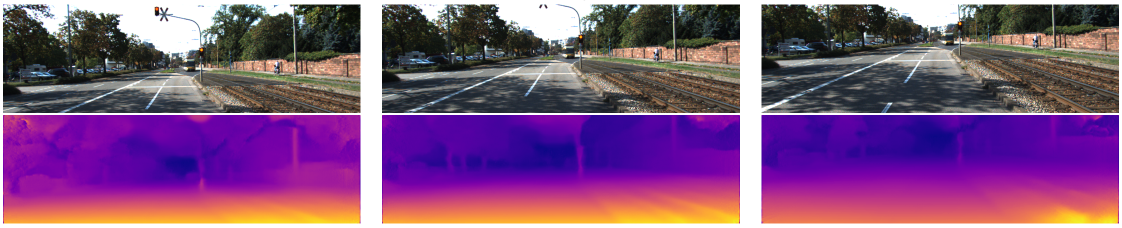
It appears that MonoDepth indeed assumes the presence of a flat ground in the lower half of the image, but does it assume a fixed depth map for the ground or does it detect the horizon in the input image? The latter would allow the network to correct changes in the pitch of the camera. To test this, the network is first presented with artificial images based on Figure 3.3, but with the horizon line shifted to different positions. The result is shown in Figure 3.5 a-c. In these artificial images it seems that MonoDepth does not use the position of the line for the ground plane, although it does have some influence on the depth map. This is also tested on real images by cropping different regions from one of the KITTI images, see Figure 3.5 d-f. The result is different; now the ground plane in the depth maps ends at the horizon in the input image. Apparently MonoDepth does estimate the position of the horizon. At the time of writing it is not yet clear how this works. Note that MonoDepth's correction to the pitch is not perfect: the depth towards obstacles appears to have changed, especially in Figure 3.5 d.



(a) Lower horizon line.

(b) The original horizon line, aligned with the horizon in the KITTI images.

(c) Higher horizon line.

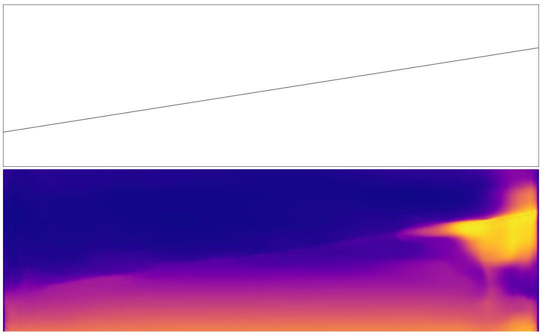


(d) Lower horizon line.

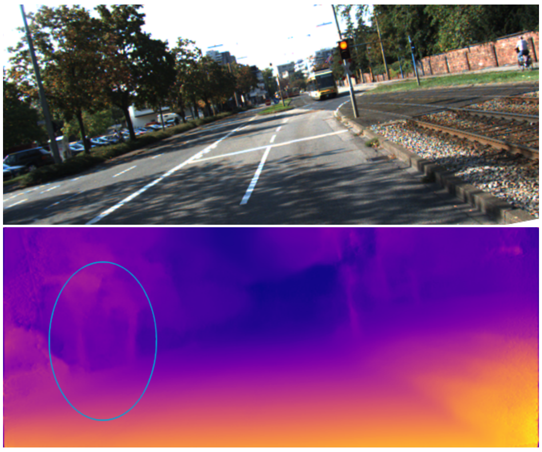
(e) Original horizon line.

(f) Higher horizon line.

Figure 3.5: *a-c*: The vertical position of the horizon line does not change the ground plane. While it has some influence on the depth map, the ground plane still seems to end at the same height in the image. *d-f*: In real images, the extent of the ground plane matches the position of the horizon. This suggests that MonoDepth has a mechanism to detect the horizon that is not triggered in images *a-c*; it is not yet understood how this works. Input images d-f: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

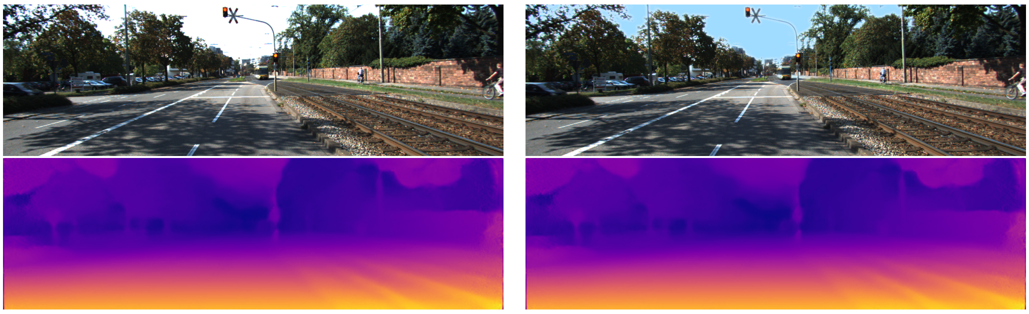


(a) When the artificial horizon line is tilted, the ground surface remains flat in the depth map. There are some artefacts where the line is above the assumed horizon.



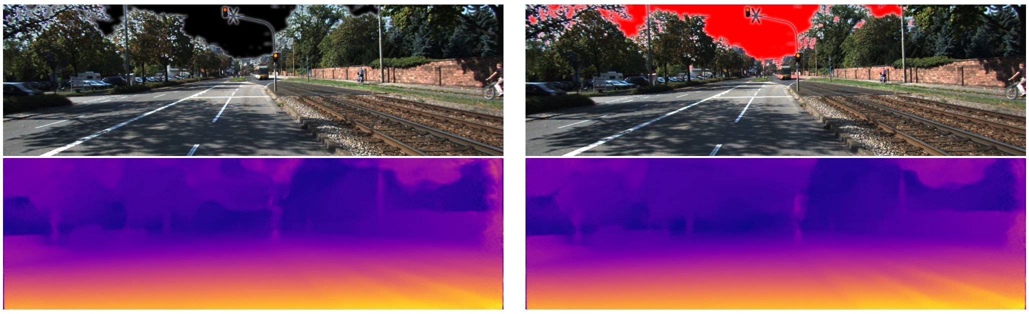
(b) Even though the input image is tilted, the road surface appears more-or-less flat in the disparity map. Also notice the trees that are vertical in the disparity map but clearly tilted in the input image.

Figure 3.6: Both in artificial and real images MonoDepth does not appear to detect roll motions and will still assume a flat road surface and vertically oriented obstacles. Input image b: KITTI stereo vision dataset (CC BY-NC-SA 3.0)



(a) Original image.

(b) Blue sky. No difference in the depth map.



(c) Black sky. The sky appears further away, there are some artifacts around the traffic light.

(d) Red sky. The sky appears further away.

Figure 3.7: Sensitivity to sky color. There is no perceivable difference between the depth maps for white and blue sky, colors that naturally occur in the training dataset. There is some difference when the sky is made black or red, but the effect remains relatively small. Input image: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

Does MonoDepth also detect roll angles? This is tested by rotating the input images, the result is shown in Figure 3.6. It appears that MonoDepth does not detect the roll angle of the camera: the ground surface still appears flat. Also note the tree trunks that appear vertical in the depth map even though they are clearly tilted in the input image. This seems another example of MonoDepth assuming the presence of certain features rather than actually observing them.

The second hypothesis is that MonoDepth detects the sky using color segmentation. While this sounds plausible, Figure 3.1 already hints that this is not entirely true: in the depth map the sky appears closer than the trees that occlude it. Figure 3.4 also shows that there is a prior expectation of the position of the sky in the upper half of the image; the sky color in the bottom half of the image does not result in an infinite depth (although its observed depth is further than that of the trees).

In Figure 3.7 the sky is replaced with different colors. The figure shows that unnatural colors have some effect on the depth estimate, but do not cause large disturbances such as objects appearing at close distance. These results show that MonoDepth does not detect the sky using (only) color segmentation. What mechanism it uses instead remains to be found in further research.

The final hypothesis on the workings of MonoDepth concerns the depth estimation of obstacles. Two options appear likely: the obstacle's size is used, or its vertical position in the image.

Figure 3.8 provides a first clue. The image shows that MonoDepth's perception of obstacles is easy to trigger: black rectangles are already shown as obstacles in the depth map. More importantly, however, is how it determined the distance towards the rectangles: the rectangles are the same size in the image but at different vertical positions. They are placed at different depths, which suggests that the vertical position had a strong influence on this estimate.

Further evidence towards this conclusion is presented in Figure 3.9. In this figure, three scenarios are presented to the network: the real-life scenario in which one car is smaller and higher in the image, one where the car is only smaller, and one where it is only placed at a higher position. In all cases, the vertical position of the car appears to control the depth estimate.

A final clue can be found back in Figure 3.5 d where the camera pitch was changed. When the camera is pitched up, the obstacles move downwards in the image. In the resulting depth map, the obstacles appear closer.

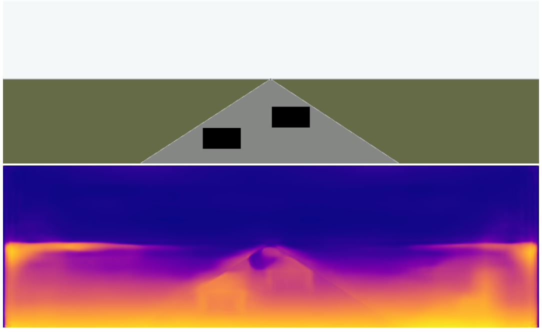


Figure 3.8: MonoDepth's depth perception can easily be triggered using simple visual features. Notice how the right rectangle appears to be further away than the left, even though they are the same size.

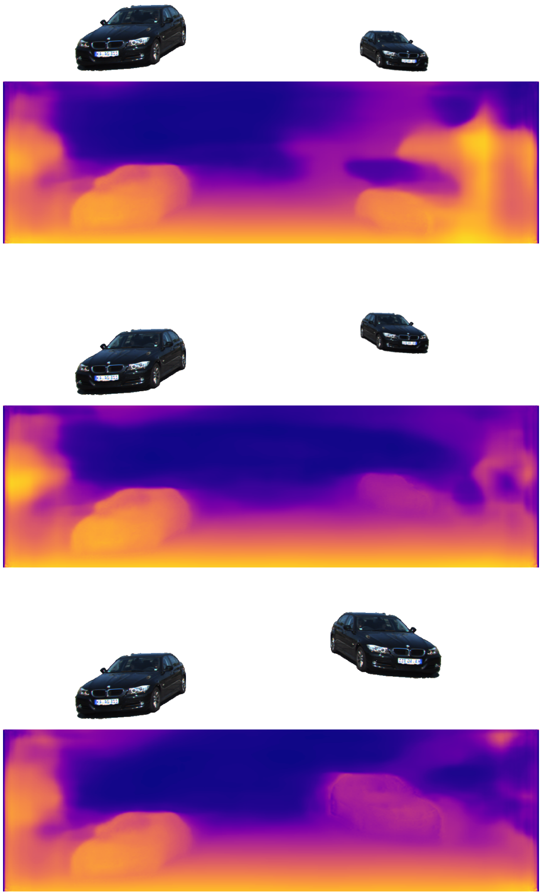


Figure 3.9: MonoDepth's depth estimation depends on the vertical position of objects in the image, not their apparent size. *Top*: the cars are assigned the same disparity, even though the car on the right has a smaller apparent size. *Middle*: the right car is smaller and positioned higher in the image (as would be the case in real images), it is correctly estimated to be further away. *Bottom*: the car on the right has the same apparent size as the one on the left. Still, it is indicated to be further away as its vertical position in the image is higher. Cars: KITTI stereo vision dataset (CC BY-NC-SA 3.0)

There are a few possible reasons why the vertical position is used as the main cue for depth. First of all, it might be easier for a CNN to measure the vertical position than the scale of obstacles because the convolution operation is translation-invariant but not scale invariant. Secondly, since the camera is fixed at a nearly constant height and attitude, distance estimates based on the vertical image position may be more accurate than estimates based on the scale as the latter depend on the real-world dimensions of the obstacle which can contain large variations.

What do these results mean for the use of MonoDepth on a UAV? The strong assumption of a flat ground in front of the camera is not compatible with the large pitch and roll angles expected on a UAV. The same holds for the use of vertical image position to estimate the depth towards obstacles: this assumes a constant height above the ground and a constant pitch angle. In conclusion, MonoDepth trained on the KITTI dataset can not be directly transferred to a UAV. Training on a suitable dataset for UAVs should reveal whether these problems come from the use of the KITTI dataset for training or whether they are more fundamental.

The ideas in this section were developed together with Guido de Croon. In 2019 we published a conference paper on this topic at the International Conference on Computer Vision:

> Tom van Dijk and Guido de Croon. How Do Neural Networks See Depth in Single Images? In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. `http://openaccess.thecvf.com/content_ICCV_2019/html/van_Dijk_How_Do_Neural_Networks_See_Depth_in_Single_Images_ICCV_2019_paper.html`

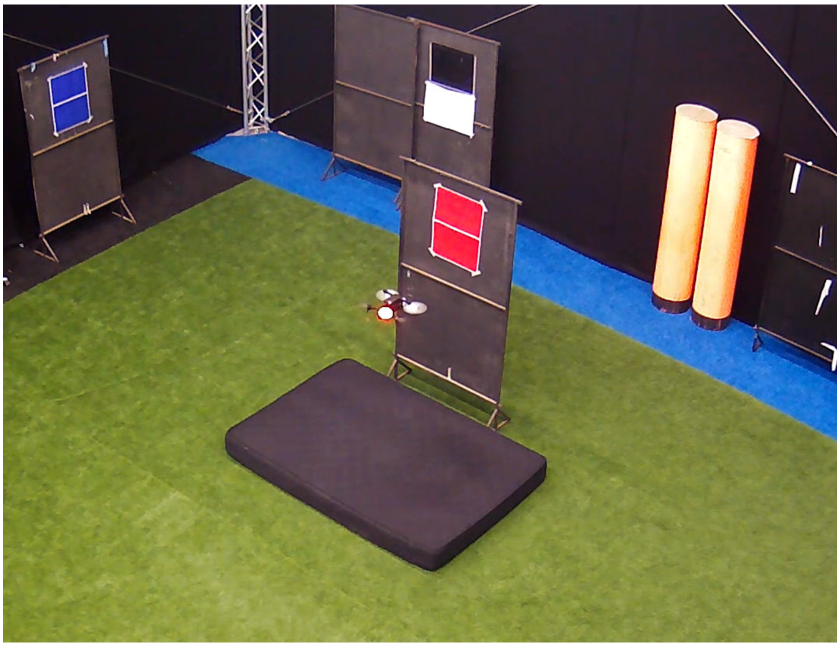


Figure 3.10: Parrot Bebop 2 with SLAMDunk stopping in front of an obstacle using SGBM and OptiTrack. (March 2018.)

3.2. Flight tests

Flight tests were performed to get more practical experience with visual obstacle avoidance. The system uses a combination of stereo vision and visual odometry for obstacle avoidance, even in GPS-less environments. These flight tests were performed as part of the *Percevite* project (`www.percevite.org`).

3.2.1. Stop-before-obstacle with OptiTrack

The first task towards obstacle avoidance was the implementation of visual obstacle detection on the Parrot SLAMDunk. This work started with a review on stereo vision and optical flow (subsection 2.1.1). The review showed that SGBM [22] still belongs to the best-performing algorithms. An implementation of SGBM was already present on the SLAMDunk and performed better than the OpenCV implementation due to its use of the GPU.

The depth map obtained from SGBM is then used as follows: a Region of Interest (ROI) is cropped from the center of the image where the view is not occluded by rotors or other parts of the drone. Of this region, the 5th percentile of depth values is calculated. The use of the 5th percentile instead of the minimum value adds some robustness to noise, at the cost of missing tiny obstacles (although this has not been a problem in practice). The 5th-percentile distance is sent to the autopilot as the distance to the nearest obstacle in front of the UAV. Additionally, the number of valid pixels (pixels for which a disparity can be found without ambiguity) is sent to the autopilot; if this value is too low the UAV is not allowed to move forward.

The movement logic is implemented as a Paparazzi[1] autopilot module (but general enough to be ported to other autopilots). The movement of the UAV is controlled using a waypoint; this waypoint can only be moved within the region that is observed to be free of obstacles with a sufficient safety margin. Since the waypoint is always in a safe region, this should prevent collisions when used in a static environment, provided that the drone can maintain its position without drift. At this stage, the OptiTrack system inside the TU Delft *Cyberzoo* was used for position feedback, so there was no drift present.

The full system was successfully demonstrated in March 2018 in the *Cyberzoo* (Figure 3.10). The code developed for the SLAMDunk/ROS is published at `https://github.com/tomvand/percevite_slamdunk`, the Paparazzi module is published at `https://github.com/tomvand/paparazzi/tree/percevite`.

---

[1] `http://wiki.paparazziuav.org/wiki/Main_Page`

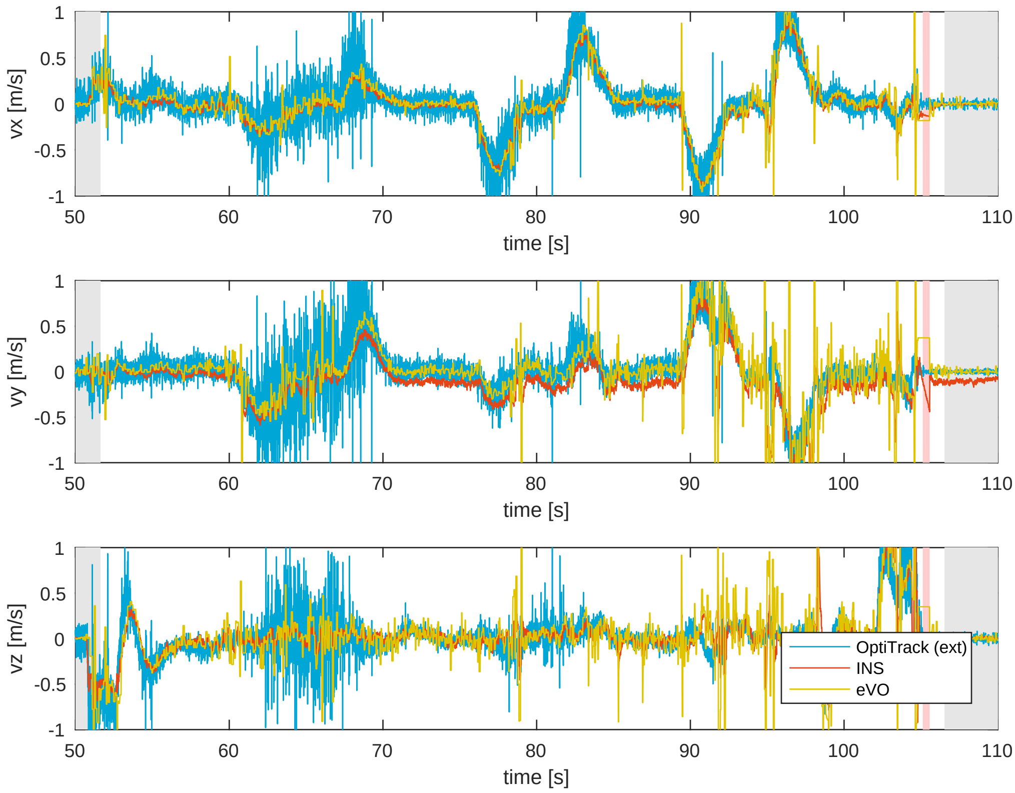


Figure 3.11: Body-frame velocities estimated using embedded Visual Odometer (eVO) compared to the ground-truth from OptiTrack. The figure shows both the raw estimates (eVO) and the velocity estimates after fusion with the accelerometer readings (INS). In general the estimate of eVO corresponds well to the OptiTrack measurements and appears to be less noisy (this may depend on the OptiTrack calibration and lighting conditions in the *Cyberzoo*). The INS estimates have a slight bias (especially in the y axis) that comes from the accelerometer. Possibly the SLAMDunk is not placed exactly above the Center-of-Gravity (CoG) of the drone.

3.2.2. Implementation of ‘embedded Visual Odometry’

While the system of March 2018 worked, its dependency on OptiTrack was a strong limitation. Work on VO started with a review of existing methods (subsubsection 2.1.2.3). Out of the reviewed methods, the following were evaluated on the SLAMDunk: ORB-SLAM2[2] [36], OKVIS[3] [28] and SVO2[4] [14]. However, all of these packages had performance issues either in terms of run-time (ORB-SLAM2, OKVIS) or drift (SVO2). The optical flow algorithm of the Paparazzi autopilot was also evaluated but was found to produce poor velocity estimates or cause segmentation faults, leading to a crash of the autopilot.

Since none of the readily available packages was suitable for use on the SLAMDunk, there was no other choice but to implement a lightweight alternative. Encouraged by the results of [32], the *embedded Visual Odometer (eVO)* algorithm by Sanfourche et al. [46] was selected. The algorithm is a relatively straightforward implementation of VO using the Perspective-3-Point (P3P) algorithm. P3P is used to compute the UAV’s pose relative to a single keyframe of 3D points; these 3D points are obtained using the depth map of SGBM. The run-time of eVO is reduced by using a fixed position for the points in the keyframe; these are not further refined when new measurements arrive. Secondly, the gyroscope is used to predict the next positions of the keypoints, thereby lowering the search region for the optical flow algorithm. My implementation of eVO is published at `https://github.com/tomvand/openevo` (and `https://github.com/tomvand/openevo-ros` for the ROS wrapper). This implementation does not reach the run-times reported in [46], but still runs at a rate of $\sim 5-10$ Hz on the SLAMDunk.

---

[2] `https://github.com/raulmur/ORB_SLAM2`

[3] `https://github.com/ethz-asl/okvis_ros`

[4] `http://rpg.ifi.uzh.ch/svo2.html`

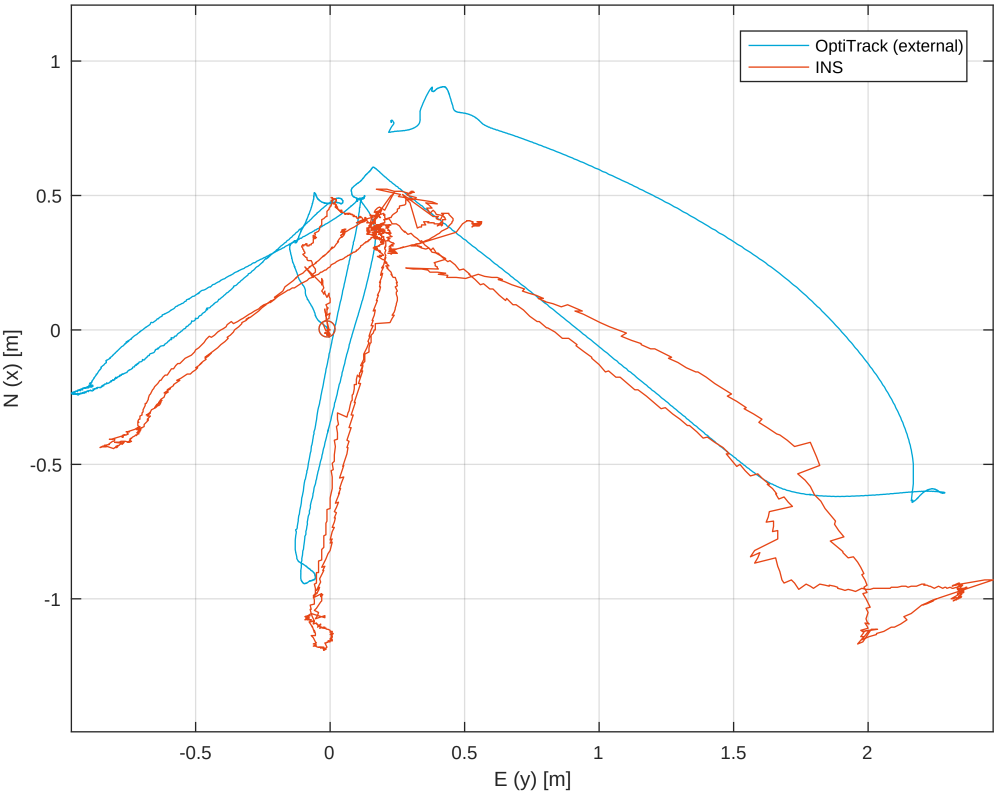


Figure 3.12: UAV trajectory estimated using eVO as input to the INS, compared to the ground truth captured using the OptiTrack system. The starting position of the UAV is indicated by the circle at $(0, 0)$. The final position error was 50 cm after a flight of 60 s.

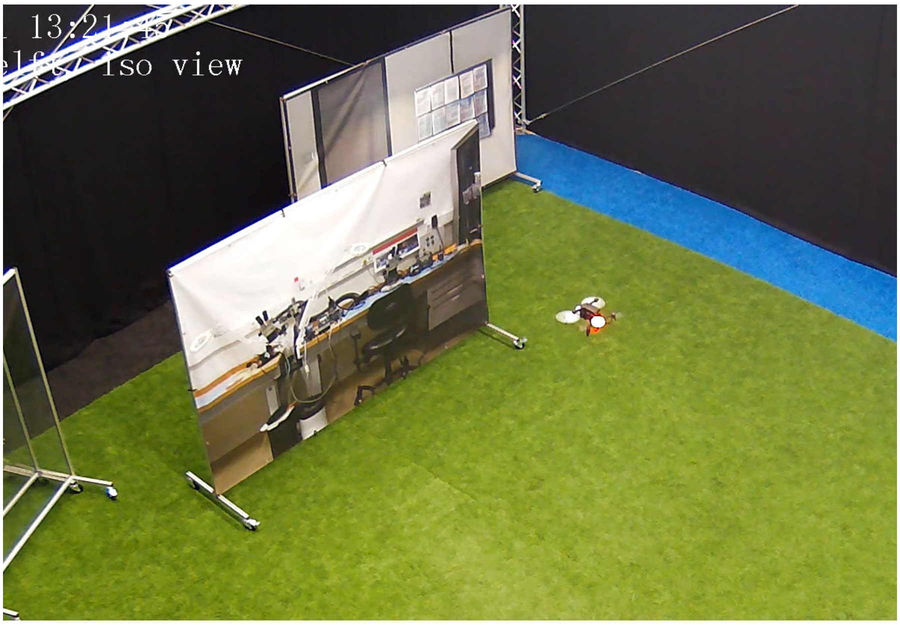


Figure 3.13: Stop-before-obstacle using eVO. (July 2018.)

The velocity estimates from eVO are sent to Paparazzi, where the horizontal EKF combines the velocities with accelerometer measurements to get a final estimate of the UAV's velocity and position. The world position and angular rates – also estimated by eVO – are currently unused. The eVO algorithm was tested inside the *Cyberzoo* and found to be surprisingly robust: it was able to follow the drone's trajectory even when no textured panels were placed around the edge of the *Cyberzoo*. Test flight results are shown in Figure 3.11 and 3.12. The drone could successfully navigate between waypoints and stop in front of obstacles (Figure 3.13).

3.2.3. Outdoor test flights

Outdoor test flights allowed eVO and SGBM to be tested in real outdoor environments with natural obstacles and lighting and under windy conditions. The velocity estimate of eVO was found accurate enough for the UAV to maintain its position under windy conditions. SGBM was able to reliably detect obstacles, although it produced a few false positive detections when facing the sun. Tests of the full obstacle avoidance system were performed by sending the drone towards obstacles. In most of the cases the drone stopped successfully in front of the obstacle (Figure 3.14).

A second goal of the outdoor flights was to add GPS to the position estimate of the drone. The use of GPS allows the UAV to follow pre-programmed trajectories and is more in line with the target applications. Fusion of GPS with eVO velocity estimates was successfully demonstrated during short test flights. Longer GPS trajectories were however not tested because of battery limitations.

During the outdoor tests the ground-truth position of the drone could not be recorded, most test flights are instead recorded as videos of the UAV combined with logs of its internal estimates. The outdoor tests were very successful; during the week the UAV only crashed once, possibly as the result of human error instead of an algorithm failure.

The collision avoidance system described here was used in the International Micro Air Vehicle (IMAV) competition 2018, where the ENAC/TU DELFT Team Paparazzi reached first place in the outdoor competition. `http://www.imavs.org/imav-2018-awards/`

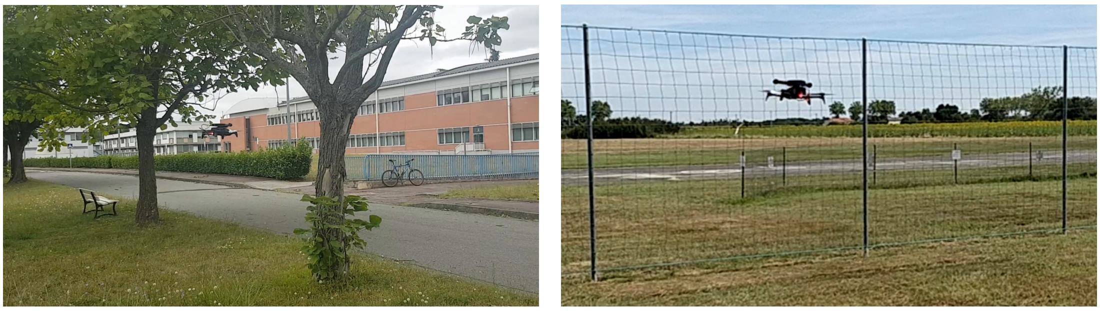


Figure 3.14: Outdoor stop-before-obstacle tests at ENAC. (August 2018.)

Bibliography

[1] B. M. Albaker and N. A. Rahim. A Survey of Collision Avoidance Approaches for Unmanned Aerial Vehicles. In *Technical Postgraduates (TECHPOS), 2009 International Conference for*, 2009. doi: 10.1109/TECHPOS.2009.5412074.

[2] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011. ISSN 09205691. doi: 10.1007/s11263-010-0390-2. Data CC BY-NC 2.0.

[3] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust Visual Inertial Odometry Using a Direct EKF-Based Approach. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 298–304. IEEE, 2015. ISBN 9781479999941.

[4] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High Accuracy Optical Flow Estimation Based on a Theory for Warping. *Computer Vision - ECCV 2004*, 3024:25–36, 2004. ISSN 03029743. doi: 10.1007/978-3-540-24673-2_3.

[5] Richard Chen, Faisal Mahmood, Alan Yuille, and Nicholas J Durr. Rethinking Monocular Depth Estimation with Adversarial Training. *arXiv preprint arXiv:1808.07528*, 2018.

[6] Xinjing Cheng, Peng Wang, and Ruigang Yang. Depth Estimation via Affinity Learned with Convolutional Spatial Propagation Network. *arXiv preprint arXiv:1808.00150*, 2018. URL `http://arxiv.org/abs/1808.00150`.

[7] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.350.

[8] David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[9] 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 27*, pages 2366–2374. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5539-depth-map-prediction-from-a-single-image-using-a-multi-scale-deep-network.pdf`.

[10] Jakob Engel, Thomas Schops, and Daniel Cremers. LSD-SLAM: Large-Scale Direct monocular SLAM. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8690 LNCS(PART 2):834–849, 2014. ISSN 16113349. doi: 10.1007/978-3-319-10605-2_54.

[11] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-Scale Direct SLAM with Stereo Cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1935–1942. IEEE, 2015. ISBN 9781479999941. doi: 10.1109/IROS.2015.7353631.

[12] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. doi: 10.1109/TPAMI.2017.2658577.

[13] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 15–22, 2014. ISSN 10504729. doi: 10.1109/ICRA.2014.6906584. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6906584`.

[14] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017. doi: 10.1109/TRO.2016.2623335.

[15] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4340–4349, 2016.

[16] Ravi Garg, Vijay B.G. Kumar, Gustavo Carneiro, and Ian Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision*, pages 740–756, Cham, 2016. Springer International Publishing. ISBN 9783319464848. doi: 10.1007/978-3-319-46484-8.

[17] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient Large-Scale Stereo Matching. In *Computer Vision – ACCV 2010*, pages 25–38, 2011. ISBN 978-3-642-19314-9. doi: 10.1007/978-3-642-19315-6_3.

[18] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. *Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. doi: 10.1109/CVPR.2012.6248074. Data CC BY-NC-SA 3.0.

[19] James J. Gibson. *The perception of the visual world*. Houghton Mifflin, Oxford, England, 1950.

[20] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] C. Goerzen, Z. Kong, and B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*, volume 57. 2010. ISBN 0921-0296. doi: 10.1007/s10846-009-9383-1.

[22] Heiko Hirschmüller. Stereo Processing by Semiglobal Matching and Mutual Information ¨. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008. doi: 10.1109/TPAMI.2007.1166.

[23] Maximilian Jaritz, Raoul de Charette, Emilie Wirbel, Xavier Perrotton, and Fawzi Nashashibi. Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation. *arXiv preprint arXiv:1808.00769*, aug 2018.

[24] Huaizu Jiang, Erik Learned-Miller, Gustav Larsson, Michael Maire, and Greg Shakhnarovich. Self-Supervised Relative Depth Learning for Urban Scene Understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–35, 2018. URL `http://arxiv.org/abs/1712.04850`.

[25] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Robust odometry estimation for RGB-D cameras. In *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754. IEEE, may 2013. ISBN 978-1-4673-5643-5. doi: 10.1109/ICRA.2013.6631104. URL `http://ieeexplore.ieee.org/document/6631104/`.

[26] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007. ISSN 00472778. doi: 10.1109/ISMAR.2007.4538852.

[27] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pages 239–248, 2016. doi: 10.1109/3DV.2016.32.

[28] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015. doi: 10.1177/0278364914554813.

[29] Mingyang Li and Anastasios I Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013. doi: 10.1177/0278364913481251.

[30] Fangchang Ma and Sertac Karaman. Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image. *arXiv preprint arXiv:1709.07492*, 2017.

[31] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised Sparse-to-Dense: Self-supervised Depth Completion from LiDAR and Monocular Camera. *arXiv preprint arXiv:1807.00275*, 2018.

[32] Julien Marzat, Sylvain Bertrand, and Alexandre Eudes. Reactive MPC for Autonomous MAV Navigation in Indoor Cluttered Environments : Flight Experiments. 2017. ISSN 24058963. doi: 10.1016/j.ifacol.2017.08.1910.

[33] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, volume 9836, pages 3242–3249. IEEE, may 2014. ISBN 978-1-4799-3685-4. doi: 10.1109/ICRA.2014.6907325. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6907325`.

[34] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June: 3061–3070, 2015. ISSN 10636919. doi: 10.1109/CVPR.2015.7298925.

[35] Javier Minguez, Florant Lamiraux, and Jean-Paul Laumond. Motion Planning and Obstacle Avoidance. *Springer Handbook of Robotics*, pages 1177–1202, 2016. doi: 10.1007/978-3-319-32552-1_47. URL `http://link.springer.com/10.1007/978-3-319-32552-1{_}47`.

[36] R. Mur-Artal and J.D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[37] Raul Mur-Artal, JMM M M Montiel, and Juan D Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015. ISSN 1552-3098. doi: 10.1109/TRO.2015.2463671. URL `http://ieeexplore.ieee.org/document/7219438/`.

[38] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327. IEEE, nov 2011. ISBN 978-1-4577-1102-2. doi: 10.1109/ICCV.2011.6126513. URL `http://ieeexplore.ieee.org/document/6126513/`.

[39] Clint Nous, Roland Meertens, Christophe de Wagter, and Guido de Croon. Performance Evaluation in Obstacle Avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3614–3619, 2016. ISBN 9781509037629.

[40] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, (https://distill.pub/2017/feature-visualization), 2017. doi: 10.23915/distill.00007. URL `https://distill.pub/2017/feature-visualization`.

[41] Hung Pham, Scott A Smolka, Scott D Stoller, Dung Phan, and Junxing Yang. A survey on unmanned aerial vehicle collision avoidance systems. *arXiv preprint*, (arXiv:1508.07723), aug 2015. URL `http://arxiv.org/abs/{%}5Cnhttp://arxiv.org/abs/1508.07723http://arxiv.org/abs/1508.07723`.

[42] Andrea Pilzer, Dan Xu, Mihai Marian Puscas, Elisa Ricci, and Nicu Sebe. Unsupervised Adversarial Depth Estimation using Cycled Generative Networks. *arXiv preprint arXiv:1807.10915*, 2018.

[43] Peter Pinggera, David Pfeiffer, Uwe Franke, and Rudolf Mester. Know Your Limits: Accuracy of Long Range Stereoscopic Object Measurements in Practice. In *European Conference on Computer Vision*, pages 96–111. Springer, 2014. doi: 10.1007/978-3-319-10605-2_7.

[44] Matteo Poggi, Filippo Aleotti, Fabio Tosi, Stefano Mattoccia, and C V Jul. Towards real-time unsupervised monocular depth estimation on CPU. *arXiv preprint arXiv:1806.11430*, 2018.

[45] Anurag Ranjan and Michael J. Black. Optical Flow Estimation using a Spatial Pyramid Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.291. URL `http://arxiv.org/abs/1611.00850`.

[46] Martial Sanfourche, Vincent Vittori, and Guy Le Besnerais. eVO: a realtime embedded stereo odometry for MAV applications. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2107–2114. IEEE, 2013. ISBN 9781467363587.

[47] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning Depth from Single Monocular Images. *Advances in Neural Information Processing Systems*, 18:1161–1168, 2006. ISSN 0920-5691. doi: 10.1007/s11263-007-0071-y.

[48] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. 3-D Depth Reconstruction from a Single Still Image. *International Journal of Computer Visional of computer vision*, 76(1):53–69, 2007.

[49] Ashutosh Saxena, Min Sun, and A.Y. Ng. Make3D: Learning 3D Scene Structure from a Single Still Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):824–840, may 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.132. URL `http://ieeexplore.ieee.org/document/4531745/`.

[50] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1/3):7–42, 2002. ISSN 09205691. doi: 10.1023/A:1014573219977. URL `http://link.springer.com/10.1023/A:1014573219977`.

[51] Korbinian Schmid, Philipp Lutz, Teodor Tomić, Elmar Mair, and Heiko Hirschmüller. Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation. *Journal of Field Robotics*, 31 (4):537–570, 2014. doi: 10.1002/rob.21506.

[52] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 746–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[53] Nikolai Smolyanskiy, Alexey Kamenev, and Stan Birchfield. On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach. *arXiv preprint arXiv:1803.09719*, 2018.

[54] Christoph Sprunk, Gershon Parent, Luciano Spinello, Gian Diego Tipaldi, Wolfram Burgard, and Mihai Jalobeanu. An Experimental Protocol for Benchmarking Robotic Indoor Navigation. In *Experimental Robotics*, pages 487–504. Springer International Publishing, 2015. ISBN 978-3-319-23778-7. doi: 10.1007/978-3-319-23778-7_32.

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[56] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald. Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing*, 11(1):5–25, 2016. ISSN 18618200. doi: 10.1007/s11554-012-0313-2. URL `http://dx.doi.org/10.1007/s11554-012-0313-2`.

[57] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. Direct Visual-Inertial Odometry with Stereo Cameras. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1885–1892. IEEE, 2016. ISBN 9781467380263. doi: 10.1109/ICRA.2016.7487335.

[58] Kevin van Hecke, Guido C.H.E. de Croon, Daniel Hennes, Timothy P. Setterfield, Alvar Saenz-Otero, and Dario Izzo. Self-supervised learning as an enabling technology for future space exploration robots: Iss experiments on monocular distance learning. *Acta Astronautica*, 140:1–9, 2017. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2017.07.038. URL `https://www.sciencedirect.com/science/article/pii/S0094576517302862`.

[59] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. SfM-Net: Learning of Structure and Motion from Video. *CoRR*, abs/1704.0, 2017.

[60] Chamara Saroj Weerasekera, Thanuja Dharmasiri, Ravi Garg, Tom Drummond, and Ian Reid. Just-in-Time Reconstruction: Inpainting Sparse Maps using Single View Depth Predictors as Priors. *arXiv preprint arXiv:1805.04239*, 2018.

[61] Oliver J. Woodman. An Introduction to Inertial Navigation. Technical report, 2007. URL `http://www.ncbi.nlm.nih.gov/pubmed/19863683`.

[62] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[63] Chuanxia Zheng, Tat-jen Cham, and Jianfei Cai. T²Net: Synthetic-to-Realistic Translation for Solving Single-Image Depth Estimation Tasks. *arXiv preprint arXiv:1808.01454*, 2018.

[64] Yiran Zhong, Yuchao Dai, and Hongdong Li. Self-Supervised Learning for Stereo Matching with Self-Improving Ability. sep 2017. URL `http://arxiv.org/abs/1709.00930`.

[65] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised Learning of Depth and Ego-Motion from Video. In *CVPR*, page 7, 2017.

TU Delft OPEN

ISBN 978-94-6366-509-4

9 789463 665094