ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Depth perception from focus measurements on an embedded platform

Semester Project

Jonathan Müller

Tuesday, May 2, 2017

Advisors: Prof. Tobi Delbrück, Dr. Yulia Sandamirskaya, Julien N.P. Martel
Institute of Neuroinformatics, University of Zürich and ETH Zürich

Abstract

Depth perception is the problem of determining the distance of objects from the observer. Useful sample applications of depth data are 3D reconstruction and robotic navigation. Depth is challenging to acquire because it cannot be measured directly, unlike for example light intensity. Often, the desired output of such a system is a depth map: a digital image where each pixel encodes a distance. This can be used as a basis for further processing. Different approaches for measuring and reconstructing depth include include multi-vision, motion parallax, structured light, and time of flight.

This project considers depth from focus. A stack of images at different focus settings are taken of a scene. By determining at which setting a pixel was maximally in focus, depth can be reconstructed. These calculations are directly done on a neuromorphic image processor which contains a small mixed-signal computer directly in each pixel. Only the reconstructed depth data is transferred for further processing. Preprocessing and reducing data directly at the source in a highly parallel manner makes the system energy efficient and fast.

This setup can only reconstruct data where there is texture or edges in the scene. We add an algorithmic post processing step step for inpainting and noise removal. The class of algorithms explored are highly parallelizable and thus also interesting for implementation on neuromorphic hardware.

Our system is capable of delivering sparse depth frames of 32 levels at 25 fps, or 15 fps with 64 levels. Implementation on a GPU allows up to 36 fps for the algorithmic post processing step on a recent consumer laptop, and 5 fps on an embedded computing platform.

Contents

Chapter 1

Introduction

Depth perception is a classical problem in computer vision and has important applications in robotics. It is useful for orientation in a 3D space and allows robots to map, plan, navigate and react to their surroundings. Whilst humans largely depend on stereo vision, a multitude of other cues can be found in nature and technology. They can be classified into various categories such as binocular vision, monocular vision and non-visual. A non-exhaustive list is given in the following:

Stereo vision: Stereo vision uses two 2D images of a scene captured from two different poses. By estimating disparity, e.g. from certain key points, it is possible to reconstruct a 3D image through triangulation from its projections to 2D ("Stereopsis", [1]). The main disadvantage is the necessity of at least two visual sensors or eyes separated by some distance, or the need of motion between two frame captures, known as Motion Parallax.

Motion parallax: When an observer moves in a scene, the relative motion of objects in his view differs depending on distance to the observer ("Motion parallax", [2, p. 419]): Objects farther away will move slower. This can be exploited to estimate their distance to the observer. Similar to stereo vision, multiple images of the same scene are needed. It is difficult to implement in a non-static scene and also requires the observer to move.

Structured light: In this approach, a known pattern of light is projected onto the scene from the observer. By analyzing the deformation of the pattern from observer perspective, depth structure can be inferred. A popularly known example using this technique is the first version of the Microsoft Kinect Sensor [3]. The obvious disadvantage of this approach is the need for a light projecting appliance. Further problems lie in the interference with other light sources.

Time of flight: The most direct approach for measuring distance. Light (LIDAR), sound (SONAR) or radio waves (RADAR) are emitted. The time

until a reflected signal can be detected ins then measured. Additional cues like frequency shifts caused by the Doppler effect can be picked up. Systems of this kind can deliver high accuracy. Some advantages and disadvantages are shared with the structured light approach, like independence from lighting conditions and sensibility to inference.

Depth from focus: By varying the focal length and determining which objects are in focus, depth can be reconstructed. Similar to stereo vision and motion parallax, an important restrictions is that this method only works on textured surfaces. There is a physical limitation for the maximum measurable distance which depends on the optical system ("Hyperfocal distance").

1.1 Problem Statement

An important issue in visual processing is the limited bandwidth between the sensors and processing. If fine grained computations are done directly on the optical sensor, high level computations can still occur externally, but on reduced data. This overcomes the bottleneck. For example, the theoretical data rate for 32 depth levels at 25 fps and 256×256 pixels resolution as currently achieved by our system is 8.2 Mbit/s, versus 419.4 Mbit/s for transferring the stack of 32 greyscale 8 bit images per frame for external processing.

The goal of this project is to construct a system for real-time depth from focus perception, using a **focus tunable lens** [4] placed in front of the **SCAMP-5 programmable vision chip** [5]. The lens contains a liquid and can be focused very fast using an electric current (hundreds of steps a second). The SCAMP-5 chip is a neuromorphic vision sensor, which includes a small mixed-signal ALU with digital and analog registers in each pixel. A single control unit dispatches instructions to the pixel ALUs in parallel, implementing SIMD (single instruction multiple data) processing per pixel. Also, communication mechanisms for signal propagation between neighboring pixels are included. This allows low latency, low power implementations of highly parallel vision algorithms. The simple ALU and limited memory require thinking about adequate algorithm design.

The described lens and vision chip in combination allow for fast depth reconstruction. As much as possible of the processing should be done directly on the chip for low latency and reduced bandwidth between the chip and the data consumer. For computational tasks which are not feasible on SCAMP-5, implementation on a GPU (Graphical Processing Unit) should be considered for real time processing speed.

1.2 Starting point

At INI, a first version of the described lens/sensor system has been developed and implemented on SCAMP-5. The working principle is described in the following.

1. Sweep the optical power of the lens in a given range.
2. During the sweep, capture as many image frames as points of depth resolution are desired.
3. By analyzing focus in each frame and associating them to their depth, construct a sparse depth map. This means the map only contains data where there are edges and textures in the field of view.

Because the depth map is sparse and the depth levels are discrete, an additional algorithmic step for inpainting and densification can be useful. This enables the system to deliver dense depth maps, comparable to e.g. a Kinect. A mathematical model has already been developed, and implemented using a commercial Solver on an external computer. It is however not yet directly integrated with the SCAMP-5 system and has a run time in the order of tens of seconds per frame.

1.3 Objective and Outline

This will serve as a base for this thesis. The existing system should be improved and reach real-time processing capabilities.

In particular, parallelizable optimization models for densification, inpainting and denoising have to be found. This will allow for faster performance and better results.

The system should also be able to run on a mobile computing platform like the Parallela Board (https://www.parallella.org/board/) allowing placement on a robot. This will pose the challenge of reduced available computing power as compared to a Laptop or Desktop computer.

From a more general point of view, this project should explore a class of visual algorithms that can benefit from using mostly local, parallel updates. This is suitable for neuromorphic hardware, which generally keeps state distributed locally among the processing units, and allows for highly parallel computations.

1.4 Definitions

This section defines and explains certain terms used throughout this report. Some of those terms can have different meanings in different contexts.

Depth frame A single image frame of depth values.

Focus frame An image taken at a certain focus value. N focus-frames at different focus levels are required for constructing a single depth-frame.

Inpainting In an (digital) image, reconstructing missing areas from the data in known areas. In our case, this means inferring depth data for parts of the image which were not measurable because of a lack of texture.

Densification Given a depth image of discrete levels, infer depth for pixels on a continuous scale (e.g. by smoothing).

Denoising Removing noise from measurements, for example by averaging over multiple data points.

Chapter 2

Optical Model

This chapter introduces the basic model required to understand the depth from focus approach. It also contains a description of our system and its operation.

2.1 Focus model

Most of the discussion and calculations mentioned in this section have already been done in [6]. We reproduce them here as a convenience to the reader. Note that the optical model in this chapter is simplified, and detailed modeling and simulation would be required to calibrate the system in detail. For example, we neglect lens distortion effects.

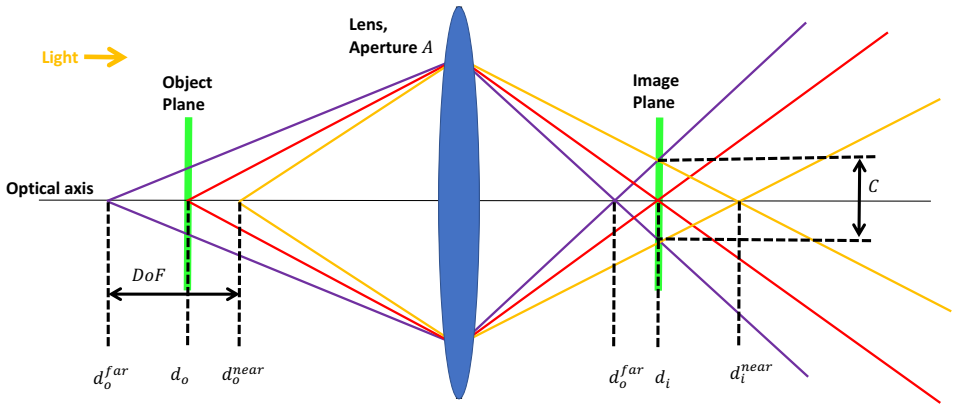


Figure 2.1: Depth of field effect. This illustration shows the relation of aperture diameter A, DoF and circle of confusion C. Graphic by J. Martel.

Our model is set up around the *optical axis*. Light rays pass through the system

parallel to or with small deviations from this axis. Parallel rays, for example sent from an object in infinite distance, will converge on a point behind the lens in distance f, called the *focal length*. Points on the object plane, a plane perpendicular to the optical axis situated in distance d_o before the lens, will form an image on the *image plane* in distance d_i behind the lens as described by

$$\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f} \tag{2.1}$$

where $\delta = 1/f$ is called the *optical power* of the lens. Higher optical power means that the rays get diverted more strongly by the lens, and the image plane moves closer to the lens.

If an optical sensor is placed behind the lens with distance d_i perpendicular to the optical axis (that is, on the image plane), the system is *in focus* for an object point in front of the lens with distance d_o. When the object distance d_o changes, the image plane will translate according to Equation 2.1. If the sensor distance remains constant at the same time, this will cause the projection on the sensor plane to appear blurred.

For a known f and d_i, the distance of an object d_o can be determined according to Equation 2.1 if it appears in focus. To be able to detect objects at different distances, one of the three variables in the equation needs to be varied. Shifting d_o means moving the optical system in the scene along the optical axis, whereas changing d_i means moving the sensor plane. We are interested in the third possibility: changing the optical power δ and thus the focal length $f = 1/\delta$ of the optical system.

So far, we implicitly assumed an infinite aperture and infinite resolution of our imaging sensor. Real systems however have finite aperture, and discrete sensor cell (pixel) size. We thus introduce the *aperture diameter* A, and the pixel diameter as C. C is also named *circle of confusion*, as it is not possible to determine resolution below this diameter. Given those physical limitations, points in a range $[d_o^{\text{far}}, d_o^{\text{near}}]$ will all appear in focus because of the maximal resolution given by C. The distances are given by

$$d_o^{\text{near/far}} = \frac{d_o A f^2}{A f^2 \pm f C (d_o - f)} \tag{2.2}$$

and the length of this range, called *depth of field* (DoF), is given by

$$\text{DoF} = \frac{2 d_o C A f (d_o - f)}{A^2 f^2 - (d_o - f)^2 C^2}. \tag{2.3}$$

2.2 Acquiring Depth Maps

The tunable lens optical power is continuously swept up and down with a frequency v, where the sweeps are linear. The current as well as the optical power thus change over time in the form of a triangular wave, as shown in Figure 2.2. For each *depth frame*, we sweep the optical power from the maximum to the minimum in the first half-wave. The object plane thus moves from close to the sensor to further away. To acquire a *depth map* of N levels, we take N *focus frames* during this half-wave ("down-sweep"). For each focus frame t, a Laplacian of Gaussian (LoG) filter is run. During the N iterations, we then store for each pixel the t at which the LoG result was maximized. This is the index of the depth frame at which the pixel was maximally in focus. After each LoG computation, we wait for a fixed delay ΔT so that N iterations fit into a down-sweep evenly distributed.

During the up-sweep which resets the optical system to the state at the beginning of a new depth frame iteration, we transfer the sparse depth map computed in this way on SCAMP-5 to the embedded computing platform via USB. No processing work is done on the image sensor during that time.

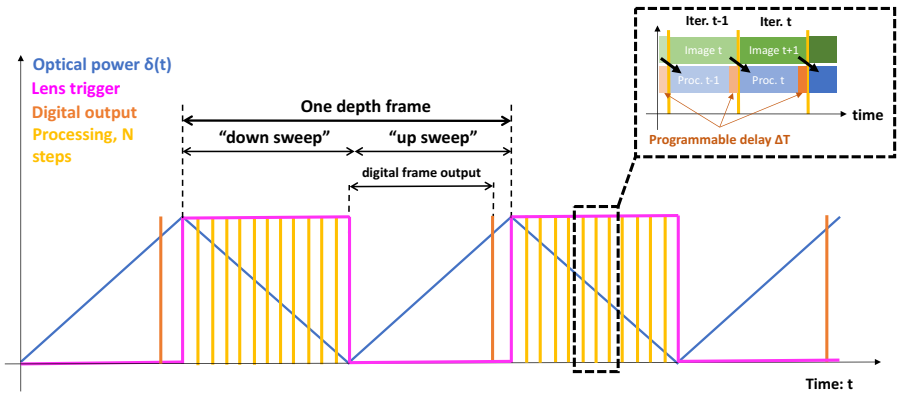


Figure 2.2: Acquiring depth frames. Graphic by J. Martel.

2.3 System description

The assembled system is shown in Figure 2.3. A more detailed description is given in the following:

1. The *object plane* is a plane perpendicular to the optical axis of the system (which is shown as a red arrow).

2. By changing the optical power of the *tunable lens*, the distance of the

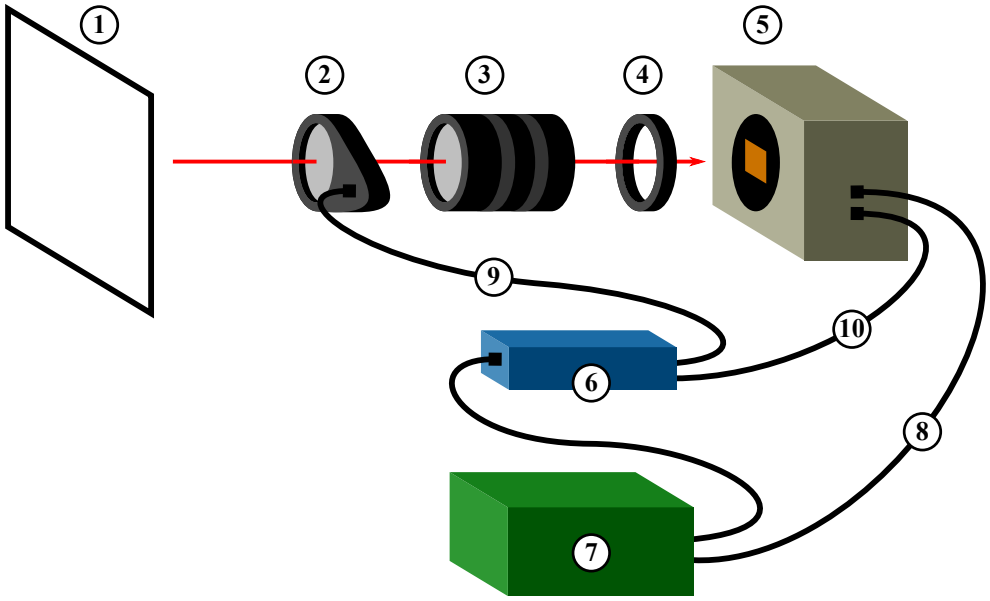


Figure 2.3: Schema of the assembled system. (1) object plane, (2) focus tunable lens, (3) fixed back lens, (4) distance ring, (5) SCAMP-5 sensor and case, where the object plane is projected to the image place, (6) lens current driver, (7) embedded computer running Windows 10, (8) USB-Cable between SCAMP-5 and computer, (9) lens cable, (10) frame trigger cable.

object plane to the sensor and image plane can be changed. The optical power is tunable by changing an electric current, and can thus be controlled from a computer.

3. A *fixed lens* is needed to focus light onto the small sensor surface. It also serves to hand-tune the base optical power of the whole system, and allows to change the aperture size.

4. The *distance ring* ensures correct distance to put the SCAMP-5 sensor into the focal plane.

5. A case contains and protects the *optical sensor* itself as well as the electronics needed to support it.

6. The lens *current driver* contains a high precision controllable current source and can be operated via a serial connection from the computer. It also emits a lens trigger signal once per depth-frame at the start of the optical power down-sweep, to signal the SCAMP-5 sensor to start collecting focus-frames. The current can be set from $-191\,\mathrm{mA}$ to $191\,\mathrm{mA}$, which maps almost linearly to an optical power of $-2\,\mathrm{dpt}$ to $2.3\,\mathrm{dpt}$ [7]. Higher (positive) current means higher optical power, thus bringing objects closer to the sensor in focus. By changing lower current limit, the

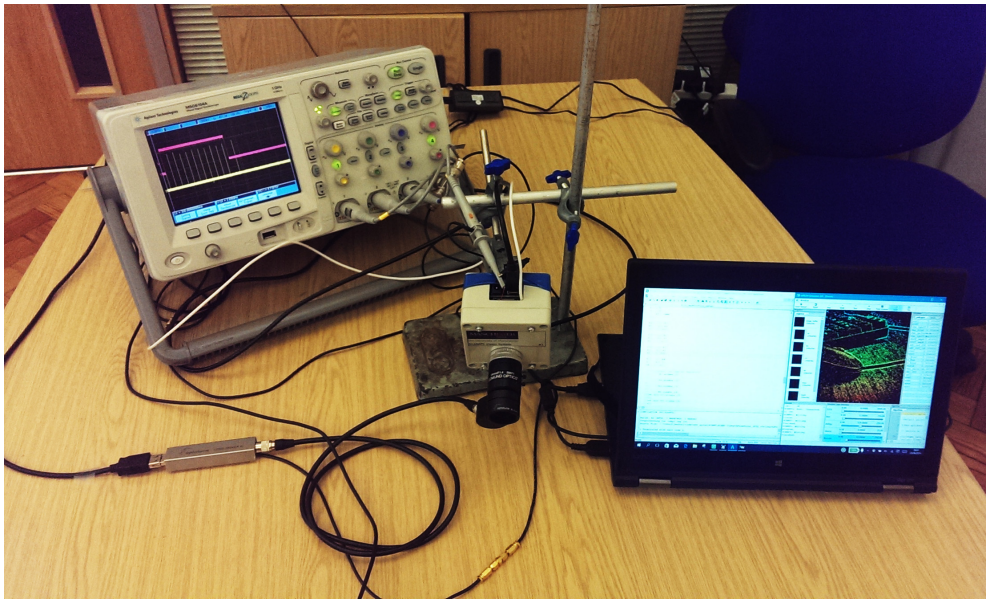


Figure 2.4: Photo of the system setup. The grey box in the center of the image is the case containing the imaging sensor. Screwed onto it are the fixed and the tunable lens. Laying on the table bottom left in the picture is the lens driver. The laptop to the right replaces the embedded computing system in this experimental setup. The Oscilloscope is needed to synchronize the delay between frame captures with the frame trigger. Photo by J. Martel.

far distance limit of the system can be changed, and vice versa for the upper current limit.

7. An *embedded Windows computer* (LattePanda) with a power consumption of approximately **10 W** controls the lens driver and downloads the sparse focus data from SCAMP-5. It then runs the inpainting and densification algorithm on its builtin GPU via OpenCL.
8. Through the USB *sensor cable*, firmware and instructions are uploaded to the optical sensor and its controlling FPGA, and image data is downloaded to the embedded platform.
9. The *lens cable* connects the lens to the lens current driver.
10. The *lens trigger cable* brings the trigger signal from the lens driver to SCAMP-5.

2.4 Physical Limitations and Tradeoffs

Understanding a systems underlying physical limitations is necessary to effectively modeling it, and knowing its strengths and weaknesses. In the following,

we discuss limiting properties of our system and their effects.

2.4.1 Depth of Field, Hyperfocal Distance

This sections presents two important limitations visible in **??**.

First, DoF will increase as d_o increases. This means that we have higher depth resolution at close distances, and that we cannot distinguish depth anymore from a certain distance. This effect is known in photography: when a photo lens is tuned to "infinity", this means $f = d_i$ and all objects past the *hyperfocal distance* will appear in focus.

The second effect is that we cannot distinguish the depth distance of pixels in the same depth of field interval. We can determine the d_o around which the DoF interval is placed, but not where in this range a pixel actually lies. Pixels assigned to the same level could be located anywhere in that range. As the aperture diameter A gets bigger, DoF gets smaller and thus our possible resolution bigger.

2.4.2 Exposure Time

To gather ν depth frames at N levels per second, the system has to record $N_f = \nu N$ frames per second in total. Because only half of $T = \frac{1}{\nu}$ is spent recording, the exposure time per focus frame is at most $\frac{q}{2\nu N}$. For $N = 32$ at $25\,\mathrm{fps}$, this gives $0.625\,\mathrm{m}$.

Because of the complex processing circuitry on each pixel, the SCAMP-5 sensor has a very low fill factor. Each pixel has a surface area of $32.26\,\mu\mathrm{m} \times 32.26\,\mu\mathrm{m}$, of which only approximately 6% are dedicated to the photo diode [8]. Typical CMOS sensors used in modern digital cameras reach fill factors in the range of 30% to 90% [9, p. 383]. This means that our system might be limited by light sensitivity in low light scenes, making collection of depth frames at high frame count impossible. On the other hand, the system can be tuned to work at very brightly illuminated scenes. Light sources like the sun, which can be a source of interference for other depth measurement methods, are only benefiting the performance of our system.

Likewise to the depth of field effect, a bigger aperture diameter is beneficial to our model in terms of exposure time. Exposure time required will scale with $1/A^2$, because the bigger aperture area will allow through more light.

2.4.3 Texture

Recovering depth from focus only works where there is texture: an evenly illuminated plain white wall will not change its appearance out of focus. Here, texture can mean edges between different objects, actual texture on single objects, or light and shadow patterns projected on a scene.

This is a problem because we consequently can only get sparse data. Is is inherent to all methods which exclusively use optical cues without actively projecting texture onto the scene. Contrary, the structured light method is not affected by this problem, neither are methods which directly measure distances using signal propagation times (Radar, Lidar).

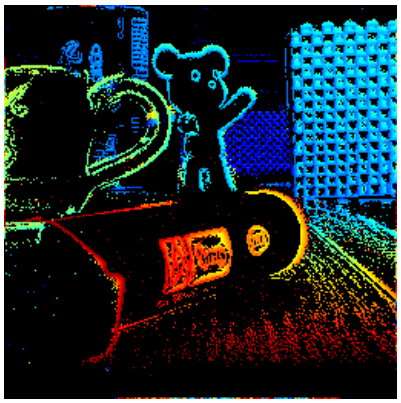
The depth of "plain objects" can however be inferred from contrast on their edges. It can be assumed that most objects without texture are also flat. Most untextured objects made by humans are flat (walls, tabletops, floors, ...), and there is almost no object without texture in nature. Inferring depth for the unknown parts of the image is described in detail in the next chapter.

Another approach to reduce this problem would be to simply add texture into the scene. This could for example be done by projecting a light pattern into the scene, similar to a structured light system. We however do not have to know the light pattern, and any pattern containing sharp edges will help.

Chapter 3

Models for Inpainting, Densification and Denoising

In this chapter, we discuss how sparse depth data as provided by our system can be enhanced by inpainting, densification and denoising. We will present a discrete model and the reasoning behind it. This model is then reformulated in the variational framework, which will allow the application of certain fast solution algorithms. Those algorithms are presented towards the end of this chapter. Implementation and results for all the models are presented in the next chapter.



(a) Depth data

(b) Color photo

Figure 3.1: Sample of retrieved sparse data. Red means closer, dark blue more distant. In the first sub figure, the depth map is shown. In the second image, we show the scene from the same point of view using a conventional camera.

3.1 Problem

As already described in section 2.4, a depth from focus system typically collects data only where there is texture, as visible in Figure 3.1. Texture means spatial variation in the image intensity, for example object edges or patterns on individual objects. Additionally, we can only get limited discrete depth levels because of the depth of field effect. The depth of field effect causes all objects in a certain distance range to appear in focus, as opposed to just objects in a single exact distance plane. It is caused by the finite aperture of our systems, as explained in chapter 2. Our system relies on having a depth of field as narrow as possible, however we cannot resolve the exact depth within the depth of field. Finally, we also get outlier measurements (noise) as visible for example on the cap of the pen in the sample image.

We present a mathematical model that attempts at inpainting, denoising and densifying the depth data:

1. Continuous depth values should be found from the measured discrete levels ("Densification").
2. Find values for the unknown pixels in the depth map ("Inpainting").
3. Remove outlier measurements ("Denoising").

Let the *image domain* $\Omega \subset \mathbb{R}^2$ be an open set. A continuous domain, continuous value *greyscale image* or *depth map* is a function $f : \Omega \longmapsto \mathbb{R}$. In the following, the symbols f and g will always denote functions of this type.

To define discrete domain, continuous value images (corresponding to a digital image of floating point valued pixels), let

$$\Omega_1 = \{(i, j) : 1 \leq i \leq m, 1 \leq j \leq n\} \tag{3.1}$$

be the regular Cartesian grid of size $m \times n$, with (i, j) representing the pixel indices. Furthermore, let $X = \mathbb{R}^{m \times n}$ be a finite dimensional vector space with standard scalar product

$$\langle u, v \rangle = \sum_{i,j} u_{i,j} v_{i,j} \quad \forall u, v \in X \tag{3.2}$$

and induced norm

$$\|u\| = \sqrt{\langle u, u \rangle} \quad \forall u \in X. \tag{3.3}$$

Unless otherwise mentioned, $u, v \in X$ will always hold in the following.

Finally, we define the vector space $Y = X \times X$ which we will need for definition of the differential operator.

3.2 Discrete Model

This model was developed at INI in together with J. Martel. It tries to formulate constrains which embed our understanding on the physical properties and constraints of the system.

We assume $q \in \{\varnothing, 1, ..., N\}^{n \times m}$ to be a $n \times m$ pixels sparse depth map of N depth levels retrieved as described in chapter 2. Its elements are q_k with indices $k = (i, j) \in \Omega_1$, and $\varnothing$ indicates a missing data element. u as defined before denotes the depth map we want to reconstruct. Note that q lives in the same domain as u, but only takes discrete values.

We find $\hat{u}$ by solving

$$\hat{u} = \underset{u_k, \xi_k^{+,-}, \chi_{k,l}^{+,-}}{\operatorname{argmin}} \sum_k w_k \left(\xi_k^+ + \xi_k^-\right) + \lambda \sum_{k,l} \chi_{k,l}^+ + \chi_{k,l}^- \tag{3.4}$$

under the constraints

$$\left.\begin{aligned} u_k + \xi_k^+ - \xi_k^- &\geq q_k - \beta \\ u_k + \xi_k^+ - \xi_k^- &\leq q_k + \beta \\ \xi_k^+ &\geq 0 \\ \xi_k^- &\geq 0 \end{aligned}\right\} \forall k \in \Omega_1 : q_k \neq \varnothing \tag{3.5}$$

$$\left.\begin{aligned} u_k + \chi_{k,l}^+ - \chi_{k,l}^- &\geq u_l - \alpha \\ u_k + \chi_{k,l}^+ - \chi_{k,l}^- &\leq u_l + \alpha \\ \chi_{k,l}^+ &\geq 0 \\ \chi_{k,l}^- &\geq 0 \end{aligned}\right\} \forall k, l : u_k, u_l \text{ neighboring pixels} \tag{3.6}$$

where ξ_k^+, ξ_k^- are slack variables for deviations of u from the data q and $\chi_{k,l}^+, \chi_{k,l}^-$ for fronto parallel deviations between neighboring pixels. The cost functions were chosen to be similar to L_1 regularization because this will rejects Laplacian "shot noise", as compared to L_2 regularization which corresponds to a Gaussian noise model. The model uses the following parameters:

α allowing small discontinuities between neighboring u_k without penalization,

β allowing placement of u_k values in a depth of field range without penalization, and

w_k a data parameter indicating a certain ratio of available data in the neighborhood of q_k.

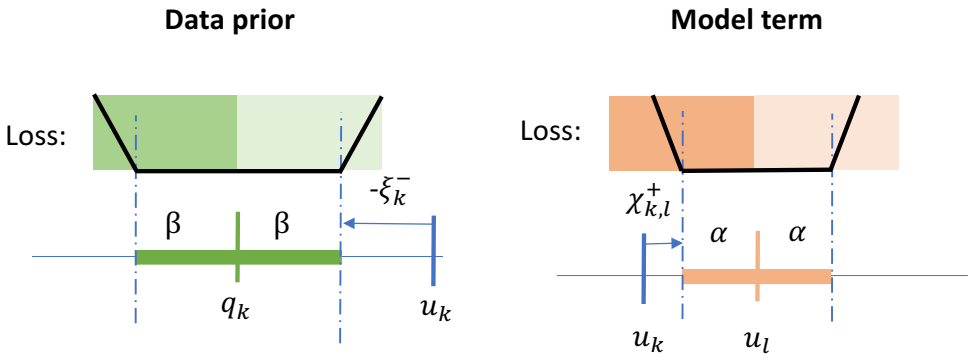


Figure 3.2: This figure illustrates the intuition behind the double hinge loss functions, and their decomposition into positive and negative slack variables. Deviations smaller than β and α do not get punished. In the data prior, deviations $u_k > q_k + \beta$ are contained in ξ_k^-, analog for negative deviations and the model term. is Graphic by J. Martel.

This is a convex optimization problem. The decomposed version of the problem presented above using slack variables gives us a linear objective. This makes it solvable by the dual simplex algorithm, e.g. using a commercial solver like Gurobi [10]. The dual simplex algorithm solves linear optimization problems by following the edges of the polyhedron defined by the constraints of the problem, searching for a minimal solution.

The original problem features for both the data prior and the model prior a loss function we call "double hinge loss". This name is chosen because the function looks like the hinge loss function common in machine learning mirrored at the $u = 0$ axis, as Figure 3.2 shows. The two loss functions are defined as:

$$C_1(u_k, q_k; \alpha) = \max\{|u_k - q_k| - \alpha, 0\} \tag{3.7}$$

and

$$C_2(u_k, u_l; \beta) = \max\{|u_k - u_l| - \beta, 0\}. \tag{3.8}$$

For understanding the model, those formulations are better suited than the decomposed version.

An illustration of the model is given in Figure 3.3. It shows the grid of latent u_k values and two measured q_k levels. Results for this model will be shown in chapter 4.

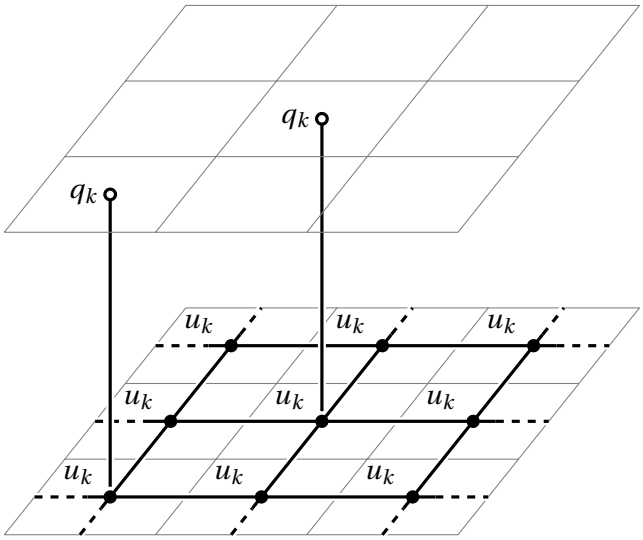


Figure 3.3: Illustration of the developed model. u_k are the latent piecewise continuous depth values, q_k sparse measured discrete depth levels.

3.3 Variational Model

While the discrete view on the problem is a good first approach to our problem, it is also somewhat limited. We will take a step back and change into the continuous world. This will allow us to formulate constraints at a more generalized and higher level, and leverage tools of continuous optimization and PDE solving.

Variational methods employ a continuous view: Images are viewed as functions, and the solution to a variational problem formulation is the minimizer to some energy functional. An extensive introduction is given in [11]. The prototypical application is image reconstruction using the total variation regularizer as first proposed by Rudin, Osher and Fatemi [12]. It is commonly named the ROF model after its authors, and is formulated as

$$\hat{f} = \arg\min_f \int_\Omega \|\nabla f\|_1 \, dx + L(f; g) \tag{3.9}$$

where

$$\|\nabla f\|_1 = \sqrt{f_x^2 + f_y^2} \tag{3.10}$$

is the discretized total variation norm and

$$L(f; g) = \frac{\lambda}{2} \int_\Omega (f - g)^2 \, dx \tag{3.11}$$

a data term loss function tying the solution to measured data g. The norm allows sharp discontinuities in the image, whilst discouraging high frequency changes (noise). Note that this problem is convex in the variable f. This guarantees a global optimum, and allows using the tools of convex optimization. Specifically, we will make use of the algorithms proposed in [13]. First however, we adapt the ROF model to our problem.

We replace the quadratic data term by the double hinge loss $C_1(f,g)$ from Equation 3.7. As already mentioned in the section before, this is to model the depth of field effect and Laplacian noise. To get inpainting, we introduce a spatially varying $\lambda(\cdot)$ ("mask") which is λ_0 where we have measured data and zero otherwise.

Our problem is then

$$\hat{f} = \arg\min_f \int_\Omega \|\nabla f\|_1 dx + \int_\Omega \frac{\lambda}{2} \max\{|f-g| - \alpha, 0\}\, dx. \tag{3.12}$$

Note that the function arguments x (location) have been omitted for f, g and λ. As compared to the discrete model, $\|\nabla\cdot\|_1$ now takes the role of Equation 3.8.

We will also consider two variants of the problem with modified data loss functions. First, the original ROF problem extended by inpainting

$$\hat{f} = \arg\min_f \int_\Omega \|\nabla f\|_1 + \frac{\lambda}{2}(f-g)^2\, dx \tag{3.13}$$

and second, with L_1 loss function:

$$\hat{f} = \arg\min_f \int_\Omega \|\nabla f\|_1 + \frac{\lambda}{2}|f-g|\, dx. \tag{3.14}$$

3.4 Algorithms and Hyperparameters

Before introducing the algorithmic framework, some definitions are needed. They are reproduced from [14] and [13]. We define a function $F : X \to \{-\infty, +\infty\}$ for the following definitions.

Definition 3.1 *The* domain *of a function F is*

$$\operatorname{dom} F = \{u \in X : F(u) < +\infty\}. \tag{3.15}$$

Definition 3.2 *The* epigraph *of a function F is*

$$\operatorname{epi} F = \{(u,v) \in X \times X : F(u) \le v\}. \tag{3.16}$$

In an intuitive way, it is the area above the function graph for the plot of a one-dimensional real function taking one argument.

Definition 3.3 *A function is* closed proper convex *if its epigraph is a nonempty closed convex set.*

Definition 3.4 *The* proximal operator $\mathbf{prox}_{\tau F} : X \to X$ with parameter τ *of a function* F *is*

$$\mathbf{prox}_{\tau F}(u) = \underset{v}{\arg\min}\, F(v) + \frac{1}{2\tau}\|v - u\|_2^2 \tag{3.17}$$

where $\tau > 0$.

Definition 3.5 *The* convex conjugate F^* *of a function* F *is*

$$F^*(u) = \max_{v \in X} \langle u, v \rangle - F(v) \quad \forall u \in X \tag{3.18}$$

where $\langle u, v \rangle$ is a bilinear form which happens to be the inner product in our case.

Definition 3.6 *Let* $K : X \to Y$ *be a continuous linear operator. The* adjoint operator $K^* : Y \to X$ *is then given as the continuous linear operator fulfilling*

$$\langle Kx, y \rangle_X = \langle x, K^* y \rangle_Y \quad \forall x \in X, y \in Y. \tag{3.19}$$

With those definitions, we can now introduce the general problem framework for the algorithms from [13].

Consider X, Y as defined in section 3.1. Let $K : X \to Y$ be a continuous linear operator with norm

$$\|K\| = \max\{\|Kx\| : x \in X \text{ and } \|x\| \leq 1\}. \tag{3.20}$$

The optimization problem is

$$\hat{x} = \underset{x \in X}{\arg\min}\, F(Kx) + G(x) \tag{3.21}$$

with $G : X \to [0, +\infty]$ and $F : Y \to [0, +\infty]$ "simple" functions, where simple means that $\mathbf{prox}_{\tau G}$ and $\mathbf{prox}_{\sigma F}$ are easily computable (ideally by a closed form solution).

The algorithms will make use of the proximal forms $\mathbf{prox}_{\tau G}$ and $\mathbf{prox}_{\sigma F}$, as well as the adjoint operator K^*.

Algorithm 1 Algorithm 1

1: ▷ $(\tau, \sigma) > 0, \theta \in [0,1]$ are hyperparameters
2: ▷ $(x_1, y_1) \in X \times Y$ initial fields
3: **procedure** Algorithm1$(\tau, \sigma, \theta, x_1, y_1)$
4: $\quad \bar{x}_1 \leftarrow x_1$
5: $\quad$ **for** $n \leftarrow 1,$ niter **do**
6: $\quad\quad y_{n+1} \leftarrow \text{prox}_{\sigma F^*}(y_n + \sigma K \bar{x}_n)$
7: $\quad\quad x_{n+1} \leftarrow \text{prox}_{\tau G}(x_n - \tau K^* y_{n+1})$
8: $\quad\quad \bar{x}_{n+1} \leftarrow x_{n+1} + \theta(x_{n+1} - x_n)$
9: $\quad$ **return** x_{n+1}

3.4.1 Algorithms

Algorithm 1 has been shown to have $1/N$ convergence rate in [13].

Definition 3.7 *The* modulus of convexity γ *of a function* $F(u)$ *is*

$$\gamma_F(t) = \inf\left\{\Delta = \frac{F(u)+F(v)}{2} - F\left(\frac{u+v}{2}\right) : \|u-v\| = t,\ u, v \in \text{dom}\, F\right\}. \tag{3.22}$$

As shown in Figure 3.4, the modulus of convexity is a measure for the convexity of a function, expressing the "minimum curvedness" of the function on all intervals of length t.

Definition 3.8 *A function* F *is* uniformly convex *(u.c.) if*

$$\gamma_F(u) > 0 \quad \forall u > 0. \tag{3.23}$$

For example, the function $F(x) = \frac{\lambda}{2}x^2$ is u.c. with modulus λ.

Algorithm 2 extends Algorithm 1 for the case that G or F^* is u.c. with modulus γ, achieving $1/N^2$ convergence.

3.4.2 Casting into the Framework

The discrete versions of Equation 3.12, 3.13 and 3.14 are

$$\hat{u} = \underset{u}{\arg\min} \|\nabla u\|_1 + \sum_{i,j} \frac{\lambda_{i,j}}{2} \max\{|u_{i,j} - v_{i,j}| - \alpha, 0\} \tag{3.24}$$

$$\hat{u} = \underset{u}{\arg\min} \|\nabla u\|_1 + \sum_{i,j} \frac{\lambda_{i,j}}{2} (u_{i,j} - v_{i,j})^2 \tag{3.25}$$

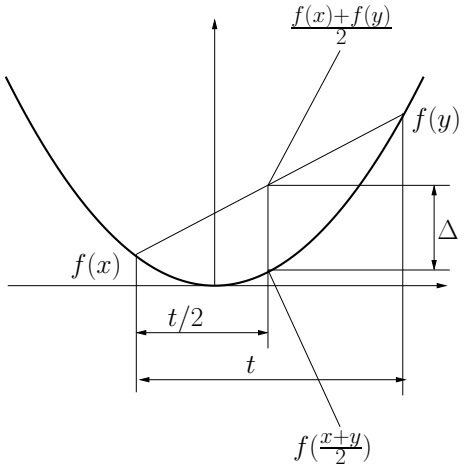


Figure 3.4: Modulus of convexity illustrated for a function $f(x)$. Graphic taken from [15].

Algorithm 2 Algorithm 2

1: ▷ $(\tau_1, \sigma_1) > 0$ with $\tau_1 \sigma_1 \|K\|^2 \leq 1$ are hyperparameters
2: ▷ $(x_1, y_1) \in X \times Y$ initial fields
3: **procedure** ALGORITHM2($\tau_1, \sigma_1, x_1, y_1$)
4: $\bar{x}_1 \leftarrow x_1$
5: **for** $n \leftarrow 1$, niter **do**
6: $y_{n+1} \leftarrow \text{prox}_{\sigma_n F^*}(y_n + \sigma_n K \bar{x}_n)$
7: $x_{n+1} \leftarrow \text{prox}_{\tau_n G}(x_n - \tau_n K^* y_{n+1})$
8: $\theta_n \leftarrow 1/\sqrt{1 + 2\gamma\tau_n}$
9: $\tau_{n+1} \leftarrow \theta_n \tau_n$
10: $\sigma_{n+1} \leftarrow \sigma_n / \theta_n$
11: $\bar{x}_{n+1} \leftarrow x_{n+1} + \theta_n (x_{n+1} - x_n)$
12: **return** x_{n+1}

$$\hat{u} = \underset{u}{\arg\min} \|\nabla u\|_1 + \sum_{i,j} \frac{\lambda_{i,j}}{2} |u_{i,j} - v_{i,j}| \tag{3.26}$$

with

$$\|\nabla u\|_1 = \sum_{i,j} \|\nabla u_{i,j}\|_2. \tag{3.27}$$

Inserting our problem into the aforementioned general framework Equation 3.21, we have

$$K = \nabla, \quad F = \|\cdot\|_1 \tag{3.28}$$

and three variants of G for the double hinge, L_2 and L_1 loss functions

$$G_1 = \sum_{i,j} \frac{\lambda_{i,j}}{2} \max\{|\cdot_{i,j} - q_{i,j}| - \alpha, 0\} \tag{3.29}$$

$$G_2 = \sum_{i,j} \frac{\lambda_{i,j}}{2} (\cdot_{i,j} - q_{i,j})^2 \tag{3.30}$$

$$G_3 = \sum_{i,j} \frac{\lambda_{i,j}}{2} |u_{i,j} - q_{i,j}| \tag{3.31}$$

where q is the measured depth level map. For implementation of both Algorithm 1 and 2 we will need the adjoint operator K^* as well as the proximal operators $\mathrm{prox}_{\sigma F^*}$ and $\mathrm{prox}_{\tau G_{1,2,3}}$.

The adjoint operator to $K = \nabla$ is $K^* = -div$.

The convex conjugate $F^* : Y \to [0, +\infty]$ can be found as

$$F^*(y) = \delta_{\max_{i,j} \|y_{i,j}\|_2 \leq 1}(y) \tag{3.32}$$

meaning it is infinite if the euclidean norm of any element is greater than 1. Its proximal operator is the point wise projection

$$\mathrm{prox}_{\sigma F^*}(y_{i,j})_{i,j} = \frac{y_{i,j}}{\max(1, |y_{i,j}|)}. \tag{3.33}$$

The proximal operators of $G_{1,2,3}$ likewise decompose into element wise operations:

$$\mathrm{prox}_{\tau G_1}(u_{i,j})_{i,j} = \begin{cases} u_{i,j} & \text{if} \quad |u_{i,j} - q_{i,j}| \leq \alpha \\ q_{i,j} + \alpha \,\mathrm{sign}(u - q) & \text{else if} \quad |u_{i,j} - q_{i,j}| \leq \alpha + \frac{\lambda\tau}{2} \\ u_{i,j} - \frac{\lambda\tau}{2} \,\mathrm{sign}(u - q) & \text{otherwise} \end{cases} \tag{3.34}$$

$$\mathrm{prox}_{\tau G_2}(u_{i,j})_{i,j} = \frac{u_{i,j} + \tau\lambda q_{i,j}}{1 + \tau\lambda} \tag{3.35}$$

$$\mathrm{prox}_{\tau G_3}(u_{i,j})_{i,j} = \begin{cases} u_{i,j} - \tau\lambda_{i,j} & \text{if} \quad u_{i,j} - q_{i,j} > \tau\lambda_{i,j} \\ u_{i,j} + \tau\lambda_{i,j} & \text{if} \quad u_{i,j} - q_{i,j} < -\tau\lambda_{i,j} \\ q_{i,j} & \text{if} \quad |u_{i,j} - q_{i,j}| \leq \tau\lambda_{i,j} \end{cases} \tag{3.36}$$

3.4.3 Discretizing the differential operators

For numerical implementation, we have to discretize the differential operator ∇ and its adjoint $-\mathbf{div}$ whilst keeping $\mathbf{div}\nabla = \Delta$ (Laplace operator) sensible. To keep the notation simple, we will use the notation

$$u_C = u_{i,j},\ u_E = u_{i+1,j},\ u_S = u_{i,j+1},\ u_W = u_{i-1,j},\ u_S = i, j-1 \tag{3.37}$$

as illustrated in Figure 3.5. We choose forward differentiation for ∇

$$\nabla u_C = \begin{pmatrix} u_E - u_C \\ u_S - u_C \end{pmatrix} \tag{3.38}$$

and backward differentiation for $\mathbf{div}$

$$\mathbf{div}\,\mathbf{p} = p_E^{(1)} - p_C^{(1)} + p_S^{(2)} - p_C^{(2)} \tag{3.39}$$

which for Δ gives

$$\begin{aligned} \Delta u &= \mathbf{div}\nabla u \\ &= (u_E - u_C)_C - (u_E - u_C)_W + (u_S - u_C)_C - (u_S - u_C)_N \\ &= u_E + u_S + u_W + u_N - 4u_C. \end{aligned} \tag{3.40}$$

For the boundaries, we set $\nabla u_{i,j} = 0$ and $\mathbf{div}\, u_{i,j} = 0$ everywhere their invocation would access elements outside the $N \times M$ grid.

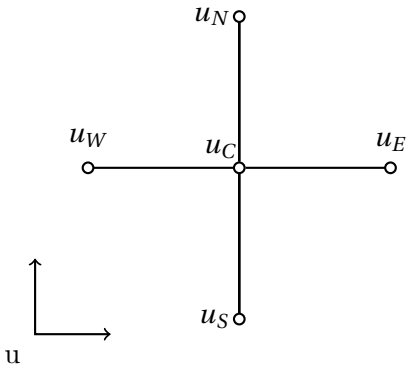


Figure 3.5: North-East-South-West notation for discretization of differential operators.

Chapter 4

Implementation and Results

After having introduced the models, their implementation end evaluation is described in this chapter. To explore and tune the models, we start with a flexible, easily modifiable prototype. We then work our way towards a highly optimized real-time implementation.

Unless stated otherwise, all performance measurements have been done on a Macbook with a 2.3 GHz Intel i7-4850HQ quad core processor, 16 GiB memory, and NVIDIA GeForce GT 750M dedicated graphics card.

The embedded computing platform "Lattepanda" features a 1.8 GHz Intel Z8350 quad core processor, 4 GiB of memory and an Intel HD Graphics GPU integrated on the CPU silicon. It runs Microsoft Windows 10. Note that the energy-saving Intel Cherry Trail CPU architecture upon which the Lattepanda platform is built delivers a lot less computing performance compared to an i7 processor, despite having a similar clock rate and the same number of cores. Because the software used to interface the SCAMP-5 sensor only runs on Windows, a Windows embedded platform had to be chosen. An additional criterion was the availability of a GPU core.

4.1 Discrete Model - Gurobi

Gurobi [10] is a commercial solver software package for convex optimization. It allows a black-box approach: The problem is fed to the software, and it then on its own chooses a method to solve the problem. The developer does not have to care about *how* the solution is reached. This makes prototyping and implementation very simple. The disadvantages are the dependency on a commercial software package (which is however available free of charge for academic purposes), and slow execution time compared to specialized implementations.

The discrete model described in section 3.2 can be directly input to Gurobi.

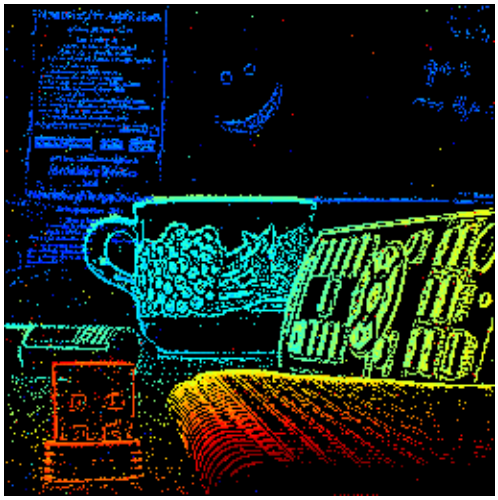


Figure 4.1: The sample sparse depth level image used for the experiments throughout this chapter.

Each measured depth level yields a constraint of form Equation 3.5. Every element of the latent field (except for pixels on the western and southern image border) yields two constraints of form Equation 3.6 each, for the connection to its southern and western neighbor. Equation 3.4 is then set as objective.

To avoid unnecessary obstruction of the development and prototyping process, this first implementation was not made capable of communicating with the optical system and sensor directly. It takes pre-recorded depth level frames as input. The open source computer vision and imaging library OpenCV [16] is used for in- and output as well as for necessary pre- and post-editing steps. C++ was chosen as programming language because both Gurobi and OpenCV offer easy to use interfaces to it.

As expected, this implementation does not perform well in terms of execution time. Multiple seconds are needed for the processing of a single frame. Gurobi chooses the dual simplex algorithm for optimization of this problem. This algorithm does work, but does not exploit the structure of the problem optimally.

Figure 4.2 shows the sample depth scene from Figure 4.1 inpainted with this implementation.

4.2 Variational Model

To work towards real-time processing, we want to make use of the fast algorithms presented in section 3.4. For this, we need the model reformulated in the variational framework. A first implementation will show their usefulness

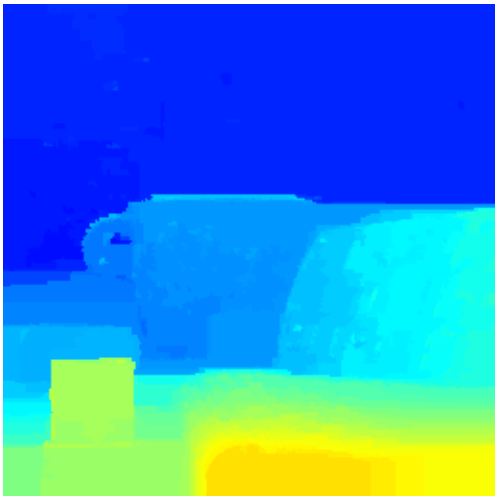


Figure 4.2: Sample depth scene inpainted using the discrete model, solved with Gurobi.

and allows tuning of parameters, without a focus on speed. A second, faster implementation then makes use of the experience gained from the first.

Unless mentioned otherwise, the parameters shown in Table 4.6 and hyperparameters from Table 4.1 are used in the following.

4.2.1 Theano

Theano [17] was chosen as prototyping platform. In short, Theano is an optimizing computation graph compiler: it allows to define a computation graph through an API exposed in the Python programming language. The Theano compiler can run various optimizations on this graph, and output and execute code for conventional CPUs as well as Graphics Processing Units (GPU). Results are then delivered back through the Python interface. This approach allows relatively high performance without the developer being required to write low level programming code.

The double hinge loss (DHL), L_2 and L_1 versions of the model where all implemented using both Algorithm 1 and 2. Note that only G_2 (L_2) is uniformly convex. The other two cost functions are non-smooth and thus Algorithm 2 theoretically is not suitable for solving them. However, it still delivered usable results.

For both algorithms, the initial field x_1 was initialized to the measured depth level image, and set to zero where no data was measured. Because of its high flexibility, this implementation was used for all parameter and algorithm tuning.

Algorithm 1				Algorithm 2			
Model:	L_2	L_1	DHL	Model:	L_2	L_1	DHL
$\tau =$	0.1	0.1	0.05	$\tau_0 =$	0.5	0.5	0.5
$\sigma =$	3.0	3.0	3.0	$\sigma_0 =$	0.3	0.3	0.3
$\theta =$	0.2	0.2	0.2	$\gamma =$	0.02	0.02	0.02

Table 4.1: Hyperparameters chosen for different algorithm and model combinations.

Through the OpenCV Python bindings, most of the boilerplate code to handle digital images could be easily ported from the Gurobi C++ implementation. Additionally to just processing single images for development, a version which can read video files and show inpainted frames as processing is done was added.

Both algorithms have three tunable hyperparameters. To chose them, all algorithm and model configurations were run with various combinations of different hyperparameters for a fixed number of iterations. The ones shown in Table 4.1 were then picked by comparing the results visually. The choice of model parameters is discussed in subsection 4.2.5.

4.2.2 CUDA and OpenCL

Both the differential and proximal operators shown in chapter 3 consist only of elementwise operations. This makes them highly parallelizable by running updates on multiple CPU cores, or a Graphics Processing Unit (GPU). Modern graphics processing units consist of a large number of small processing units. For example, the GeForce GT 750M used in this project offers 384 cores. Those cores are individually much less complex and capable than a modern conventional CPU, but enable large scale parallelization through their numbers. When talking about general purpose programming on GPUs, the terms "host" and "host memory" refer to the CPU and system memory of a computer, while "device" and "device memory" refer to the GPU and its dedicated graphics memory. The GPU cores can only access graphics memory, so data has to be transferred from host to device memory before computation can be done on them.

The Theano implementation mentioned in subsection 4.2.1 was not capable to exploit the parallel nature of the problem. Forcing Theano to execute code on multiple CPU cores or the GPU leaded to much slower execution time. According to its documentation, Theanos strength lies in parallelizing vector and matrix operations well over the size of our problem. Too much time was probably lost to memory copying and synchronization when using multicore

processing. For example, data was copied back and forth between host and device memory for every algorithm iteration. However, data transfer is actually only needed before the first and after the last iteration. Inbetween, the current state can reside in device memory. To overcome those issues, a fully custom implementation as described here was built.

CUDA [18] is a software development kit and driver package developed and offered by the GPU vendor NVIDIA. It offers a framework and SDK for general purpose programming on GPUs. In CUDAs terms, so called kernels can be executed in massively parallel manner on GPU cores. In this context, a kernel is a small program working on a subproblem of the whole task. For example, instead of iterating over an array sequentially and executing an operation on each element, using CUDA a single kernel is instantiated for each array element in parallel manner. Kernels are to CUDA and general purpose programming on GPUs what shader programs are to OpenGL.

Additionally to parallelism, we get the advantage that the GPU can work independently from the host system. Once data and program code are sent to the GPU, it does not need further supervision until we fetch the results. To exploit this, the CPU can already read and preprocess input data for the next frame while computation for the last one is still going on the GPU.

Both the single image and video processing programs were ported from Theano / Python to CUDA / C++. Algorithm 1 and Algorithm 2 were both implemented. Only the double hinge loss model was considered, because it delivered the best results in comparision of the models. Implementing the other model versions would be easy though, as only the proximal operators need to be replaced.

The embedded platform does offer a GPU directly integrated within the CPU chip (Intel HD Graphics). It cannot be leveraged using CUDA, because CUDA only supports NVIDIA GPUs. It is however supperted by OpenCL. OpenCL is an SDK very similar to CUDA, the main difference being that it is openly standardized and supported by multiple GPU vendors (among them Intel, NVIDIA and AMD). Our implementation was consequently extended to also support OpenCL.

The CUDA and OpenCL implementations are semantically identical to each other and the Theano implementation. Given the same input and parameters, they will produce the same results. Consequently, parameter tuning and algorithm development done on the flexible Theano implementation can be directly used for the GPU implementation.

4.2.3 APRON Plugin

APRON [19] is the software package used to program and interface the SCAMP-5 sensor from the host computer. The easiest way to integrate our inpaint-

ing/densification implementation was to write a plugin which is loadable by APRON. This was easily possible with the C++ / OpenCL implementation. APRON loads the inpainting and densification module as a DLL (dynamic loadable library), and passes the data to the module as an array of float numbers. After processing, the inpainted frame can then be retrieved the same way and is displayed in APRON.

4.2.4 Convergence

Realtime applications require a guaranteed upper boundary of processing time for each frame. This lead us to iterate the algorithms a constant number of time for each frame. Choosing the number of iterations per frame means trading off between processing time and quality of the result.

Using the sample depth scene shown in Figure 4.1, convergence of both algorithms on all three models was tested. For each model, Algorithm 1 was first run for a high number of iterations (100'000). The results are shown in Figure 4.3. This served to establish ground truth in terms of convergence. Next, both algorithms were run for 5'000 iterations on all models and after each step, the error distance $\varepsilon(\hat{x})$ to the ground truth was computed according to

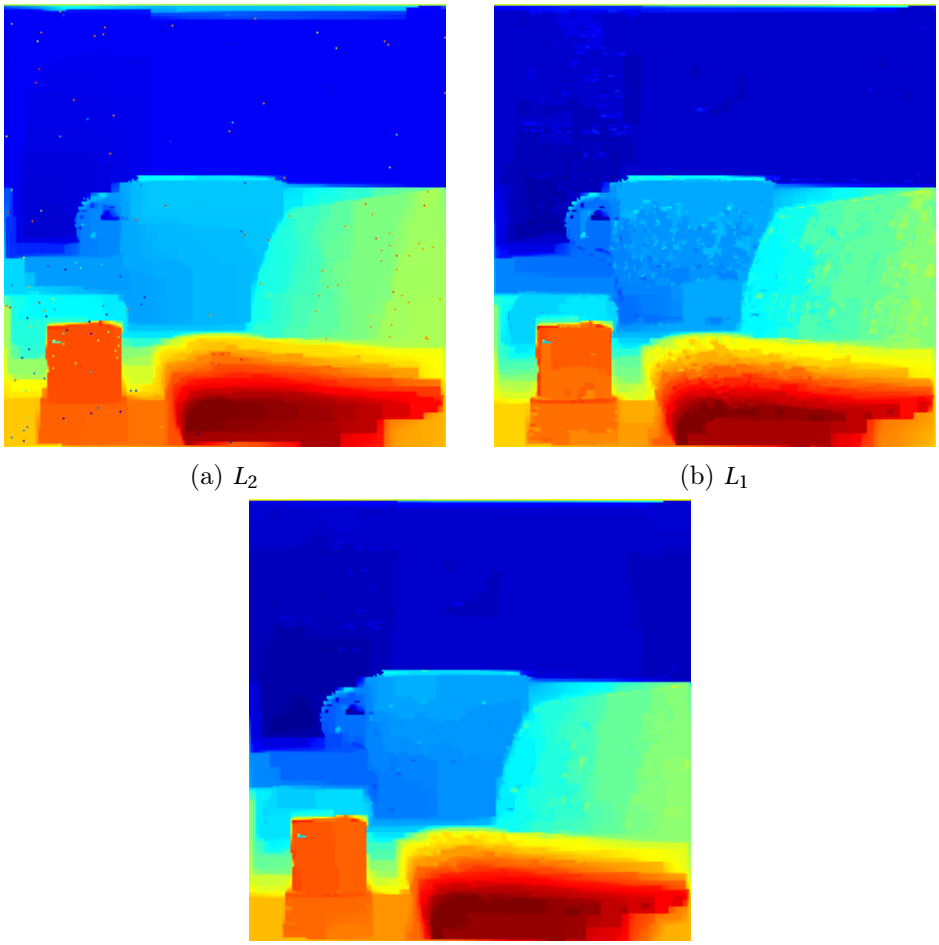
$$\varepsilon(\hat{x}) = \frac{1}{nm} \sum_{i,j} |x_{i,j} - \hat{x}_{i,j}| \tag{4.1}$$

where $\hat{x}$ is the reconstructed depth map, x ground truth, and $n \times m$ its dimensions.

The rates of convergence according to this method are plotted in Figure 4.4. The number of 5'000 iterations was chosen because more than several hundred or thousand iterations are not computable fast enough for a realtime implementation. Also, the error stops its initial steep descent approximately after the 2'000 first iterations. The rate of convergence above 5'000 iterations is thus not interesting for our application.

From the acquired data, we can see how many iterations were required for a certain algorithm and model to reach given values of ε. The numbers are shown in Table 4.2. To be able to interpret the numbers, we must know how a reconstructed map with some given ε actually looks like. This will allow us to estimate the number of iterations needed for a good trade off between quality and speed. Visualizations for different ε are given in Table 4.3.

The pictures show very visible differences between $\varepsilon = 0.01, \varepsilon = 0.005$ and $\varepsilon = 0.002$. After that however, the differences become much less apparent. This confirms that more than several thousend iterations of either algorithm will not be necessary in our case. Often, $\varepsilon = 0.01$ should already be enough. Figure 4.4 shows that it makes sense to prefer Algorithm 1 over Algorithm 2 in



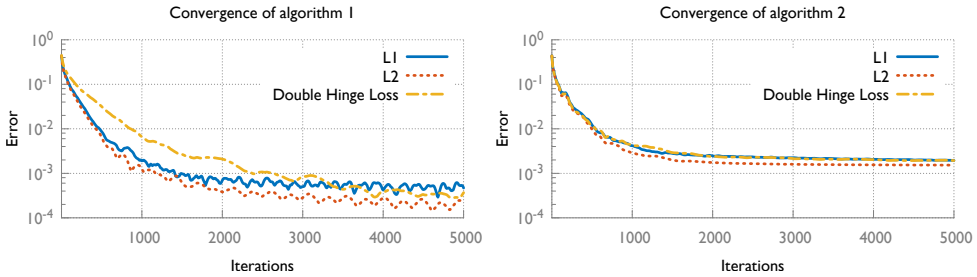
(a) L_2 (b) L_1

(c) DHL

Figure 4.3: The three models after 100'000 iterations of Algorithm 1.

almost all cases, except for the DHL model and less than around 1'300 iterations, which corresponds to $\varepsilon \approx 0.0035$. Because this is exactly our mode of application, we stick with the choice of Algorithm 2 for the DHL model in our fast implementation.

Under the assumption that subsequent depth frames are highly correlated, we can make use of partial convergence: instead of running the inpainting algorithm from reset state for every new frame, we leave the current state of the reconstructed field untouched and only update the data prior. We can then run the algorithm for only a fraction of the number of iterations required for full convergence, for example 200 or 500 iterations instead of 1200. Over the course of 3-10 frames, we then have the same total number of iterations.



(a) Algorithm 1 (b) Algorithm 2

Figure 4.4: Development of error distance for both algorithms, depending on the number of iterations.

	Algorithm 1				Algorithm 2			
	Model:	L_2	L_1	DHL	Model:	L_2	L_1	DHL
$\varepsilon = 0.01$	$n =$	428	473	859	$n =$	496	542	542
$\varepsilon = 0.005$	$n =$	527	612	1170	$n =$	658	876	905
$\varepsilon = 0.002$	$n =$	817	964	2036	$n =$	1489	4493	4080
$\varepsilon = 0.001$	$n =$	1168	1410	2372	$n =$	-	-	-
$\varepsilon = 0.0001$	$n =$	6471	28407	30585	$n =$	-	-	-

Table 4.2: The number of iterations needed to reach a certain error ε computed according to Equation 4.1. A dash means that the given ε was not reached during 100'000 iterations at this configuration.

4.2.5 Parameters

The results achievable using our models depends on a good choice of parameters. All of the models contain the λ parameter, regularizing the tradeoff between smoothing the resulting field and closeness to the data prior. Higher lambda means that higher weight for the data term, as can be seen in Table 4.4 and Table 4.5. The α parameter in the DHL model steers the allowed deviation of the recovered depth map from the measured depth levels. It directly corresponds to the depth of field effect. Table 4.4 shows how larger values of α lead to a higher deviation from the data prior, thus leading to less variation in the resulting field.

To choose parameters, all three variational models were run with a set of different parameter combinations. Through visual inspection of the resulting fields, the parameters shown in Table 4.6 were selected.

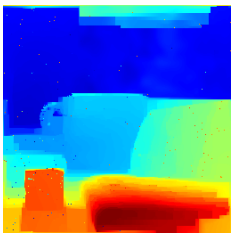
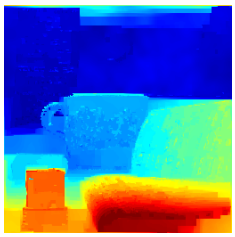
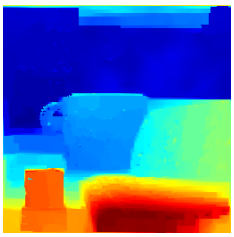
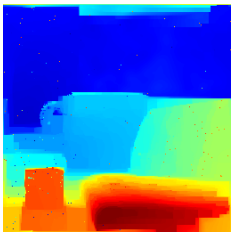
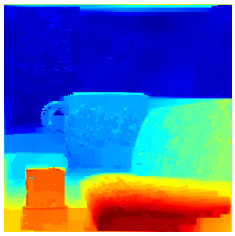
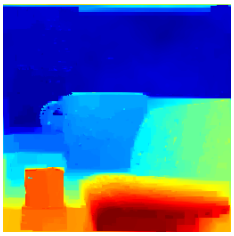
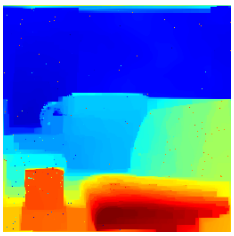
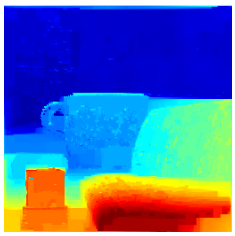
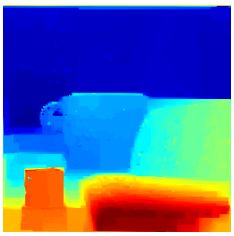
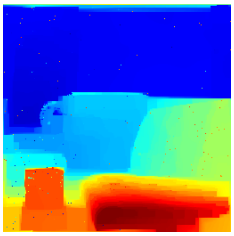
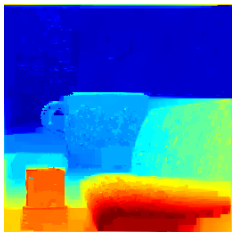
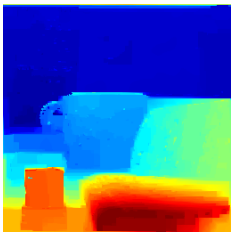
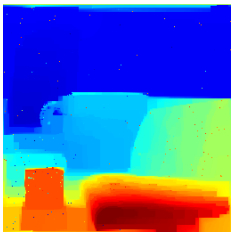
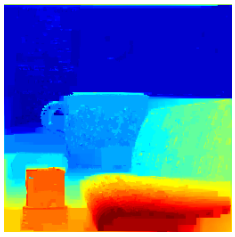
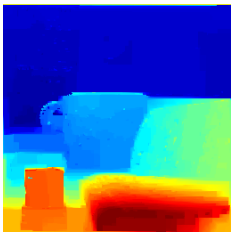
Model:	L_2	L_1	DHL
$\varepsilon = 0.01$			
$\varepsilon = 0.005$			
$\varepsilon = 0.002$			
$\varepsilon = 0.001$			
$\varepsilon = 0.0001$			

Table 4.3: Resulting reconstructed depth maps for different models and ε. For each map, Algorithm 1 was run until the given ε was reached. The corresponding number of iterations can be seen in Table 4.2.

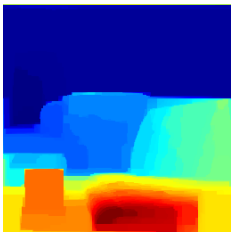
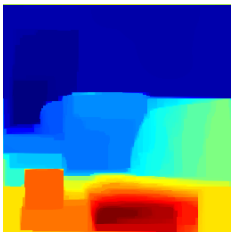
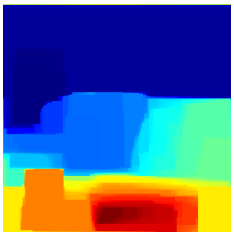
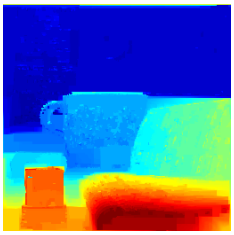
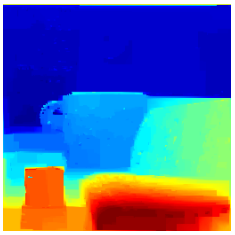
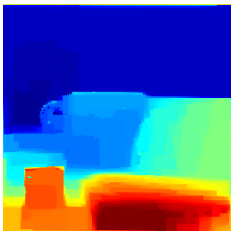
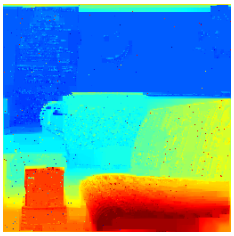
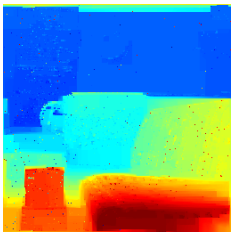
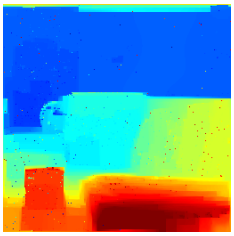
Model: DHL	$\alpha = 0.001$	$\alpha = 0.02$	$\alpha = 0.04$
$\lambda = 1.0$			
$\lambda = 2.75$			
$\lambda = 4.0$			

Table 4.4: Reconstructed depth maps for different parameter choices on the DHL model. All maps were computed using 8000 iterations of Algorithm 1.

4.3 Comparison of models

Figure 4.3 contains the best results we achieved with every model. They were made with the parameters we selected before, and are the result of running Algorithm 1 for 100'000 iterations.

It can be seen that L_2 yields a smooth image, but cannot remove individual noisy pixels. L_1 removes the noise, but makes for a much more blocky appearance. Double Hinge Loss gives a smoother looking image than L_1, while removing the noisy pixels visible in L_2. Those differences make sense considering that the L_2 model is, from a bayesian viewpoint, a MAP estimate assuming gaussian noise and a TV prior [20]. The L_1 and DHL model on the other hand assume laplacian noise, which is closer to reality in our data.

In our opinion, the double hinge loss model offers the best compromise between noise removal and smoothness.

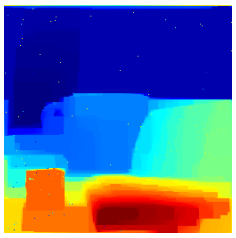
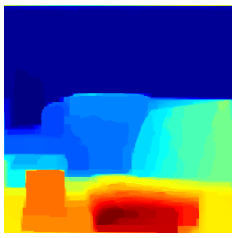
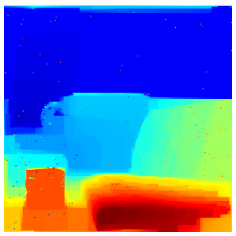
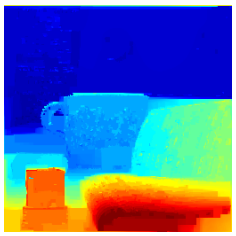
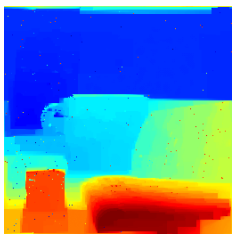
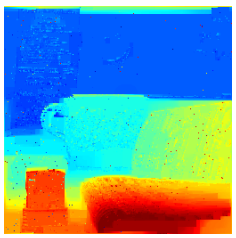
Model:	L_2	L_1
$\lambda = 15$		
$\lambda = 1.00$		
$\lambda = 30$		
$\lambda = 2.75$		
$\lambda = 50$		
$\lambda = 4.00$		

Table 4.5: Reconstructed depth maps for different choices of the λ parameter on the L_2 and L_2 models. All maps were computed using 8000 iterations of Algorithm 1.

Model:	L_2	L_1	DHL
$\lambda =$	30	2.75	2.75
$\alpha =$	-	-	0.02

Table 4.6: Final parameters chosen for the different models. A dash means that the model does not use the parameter.

4.4 Frame Rate

Execution time of the different implementations was measured on on both Macbook and the embedded platform. The timing results are shown in Table 4.7. The Gurobi implementation does not have the number of iterations listed, because they are not comparable to the number of iterations in Algorithm 1 or 2. Directly comparable however are the δt_{frame} and frames per second numbers. Running Algorithm 2 for 1200 iterations is roughly equivalent to running the Gurobi implementation for one frame. This means that the Theano implementation delivers results of equal quality at around double the frame rate the Gurobi implementation does, and OpenCL on Macbook at more than 37 times the frame rate.

When looking at the timing results for 200 and 500 iterations using partial convergence, we see that we can inpaint frames at up to 36.5 fps on the Macbook, and up to 5 fps on the embedded platform.

On the Macbook, the OpenCL implementation can make use of both the Iris Pro graphics processor integrated with the CPU, or the dedicated NVIDIA graphics card. CUDA on the other hand can only use the NVIDIA GPU. As can be seen in the results, the OpenCL is slightly faster than CUDA. The effect is bigger when using less iterations per frame. This might seem surprising at first, because the dedicated graphics card is supposed to be more powerful. It however makes sense if we accept that our problem is still memory bound: the integrated Iris Pro graphics chip uses system memory instead of separate graphics memory. When using it, data does thus not have to be copied around between two different physical locations.

4.5 File Formats

The code running on SCAMP-5 currently outputs only a single grayscale image per frame. Pixels without measured data are set to 0 and are not distinguishable from pixels with measured data of value 0. Our inpainting/densification implementation simply assumes every pixel from SCAMP-5 with value 0 to be not measured. In the APRON plugin for our implementation, the sparse depth data is received as a memory chunk of float numbers ranging between 0.0 and 255.0 (even though the values are discrete, e.g. in 32 levels). If sparse depth data has to be saved for later processing, it is simply stored as greyscale PNG image with the same assumption made above.

For storing videos however, there is a workaround needed. There was no lossless video format with support on all needed platforms available. The lossy compression formats however caused pixels to change from 0 to very low values > 0 (compression artifacts). This made working with the above solution impossible. The workaround is to save the depth values in the videos green and

Implementation	Platform	Iterations	δt_{frame}	δt_{iter}	fps
Discrete model, Gurobi	Macbook	n.a.	3896ms	n.a.	0.3
DHL, alg. 2, Theano	Macbook	200	309.6ms	1.548ms	3.2
DHL, alg. 2, Theano	Macbook	500	746.3ms	1.493ms	1.3
DHL, alg. 2, Theano	Macbook	1200	1793.4ms	1.495ms	0.6
DHL, alg. 2, CUDA	Macbook	200	28.7ms	0.143ms	34.9
DHL, alg. 2, CUDA	Macbook	500	70.0ms	0.140ms	14.3
DHL, alg. 2, CUDA	Macbook	1200	158.9ms	0.132ms	6.3
DHL, alg. 2, OpenCL	Macbook	200	27.4ms	0.137ms	36.5
DHL, alg. 2, OpenCL	Macbook	500	46.0ms	0.092ms	21.8
DHL, alg. 2, OpenCL	Macbook	1200	104.5ms	0.087ms	9.6
DHL, alg. 2, OpenCL	Lattepanda	200	201.7ms	1.009ms	5.0
DHL, alg. 2, OpenCL	Lattepanda	500	467.3ms	0.935ms	2.1
DHL, alg. 2, OpenCL	Lattepanda	1200	1169.1	0.974ms	0.9

Table 4.7: Timing measurements. All but the first entry in the table were measured using the double hinge loss model and Algorithm 2. In the Gurobi implementation, the number of iterations could not be set and the average of 5 subsequent runs is shown. For the other implementations, the average of processing each frame of a video clip containing 84 frames in total was computed. All measurements do not include time for loading and initialization. Iterations means the number of times the algorithm was looped per video frame. Frames per second is a theoretical value calculated as $\text{fps} = 1/\delta t_{\text{frame}}$.

red channel, and extract the image mask into the blue channel with discrete values of $\{0, 255\}$.

Chapter 5

Conclusion and Outlook

5.1 Conclusion

In this project we have developed a pipeline for collecting depth maps from focus. Building upon an already existing optical and sensoric system, our main contribution is a system which computes dense and inpainted depth maps from sparse depth level data retrieved by the sensor. Exploring different models and their implementation, we show that it is possible to do this close to real time (5 fps) on an embedded platform, or at up to 36.5 fps a recent consumer mobile graphics card. The computational work necessary to collect sparse depth maps is completely offloaded onto the neuromorphic vision sensor, which makes the system fast and energy efficient.

The presented system works in a purely passive manner and does not require emission of energy. Because a liquid lens which is quickly tunable using an electric current is used, there are no mechanical movable parts in the system. The system is currently physically limited by the maximum size of the aperture (DoF resolution), and the limited light sensitivity of the sensor used (frame rate). Computationally, we are limited by the speed at which we can inpaint sparse frames.

From a broader point of view, the project explores the class of variational algorithms which could be highly suitable for implementation on neuromorphic hardware. We show that our problem, with a global objective given, can be solved using local state and updates. This highly suits the distributed memory and computational elements inherent to neuromorphic systems.

5.2 Future Work

The system presented offers several approaches for improvement.

Physical limitations Several of the mentioned physical limitations could be overcome by better hardware. Using a tunable lens with bigger aperture would allow us to get a higher depth resolution, because the depth of field would be reduced. A sensor with higher light sensibility would allow for faster collection of depth frames. Using a telecentric tunable focus lens could remove zooming artifacts currently present in depth frames.

Computational power Using more powerful or higher clock rate GPU would serve to make inpainting and densification models run faster. Also possible is implementation on an FPGA.

Preprocessing on SCAMP-5 Some of the processing needed for inpainting and densification could be offloaded directly to the neuromorphic sensor. For example, anisotropic diffusion is a well suited pre-processing step. This is currently not possible because SCAMP-5 can currently not differentiate between measured 0 values and missing data.

Integrated segmentation The variational models used for inpainting and densification could be extended to automatically implement simple segmentation, for example detection of obstacles versus free space.

Using additional sensors For example, an inertial measurement unit could be leveraged to detect when the field of view is static, and the convergence schema and number of iterations for inpainting could be changed accordingly.

Improvement of models The current models for inpainting and densification could be improved, for example by adding an additional term for smoothing the resulting field. This would prevent some of the staircasing effects visible in the current results.

Bibliography

[1] U. R. Dhond and J. K. Aggarwal, "Structure from stereo-a review", *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 6, pp. 1489–1510+, 1989.

[2] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001, ISBN: 0130307963.

[3] Z. Zhang, "Microsoft kinect sensor and its effect", Tech. Rep., Apr. 2012, pp. 4–12. [Online]. Available: https://www.microsoft.com/en-us/research/publication/microsoft-kinect-sensor-and-its-effect/.

[4] M. Blum, M. Büeler, C. Grätzel, and M. Aschwanden, *Compact optical design solutions using focus tunable lenses*, 2011. DOI: 10.1117/12.897608. [Online]. Available: http://dx.doi.org/10.1117/12.897608.

[5] S. J. Carey, A. Lopich, D. R. W. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535 gops/w 256x256 simd processor array", in *2013 Symposium on VLSI Circuits*, Jun. 2013, pp. C182–C183.

[6] J. N. P. Martel, L. K. Müller, S. J. Carey, and P. Dudek, "High-speed depth from focus on a programmable vision chip using a focus tunable lens", in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, Submitted.

[7] *Electrically tunable large aperture lens*, EL-16-40-TC, Update: 10.01.2017, Optotune Switzerland AG, Jan. 2017.

[8] S. J. Carey, D. R. W. Barr, B. Wang, A. Lopich, and P. Dudek, "Mixed signal simd processor array vision chip for real-time image processing", *Analog Integrated Circuits and Signal Processing*, vol. 77, no. 3, pp. 385–399, 2013, ISSN: 1573-1979. DOI: 10.1007/s10470-013-0192-x. [Online]. Available: http://dx.doi.org/10.1007/s10470-013-0192-x.

[9] B. G. Batchelor, Ed., *Machine Vision Handbook*, ser. Springer Reference. Springer-Verlag London Ltd, 2012, vol. 2, ISBN: 978-1-84996-168-4. DOI: 10.1007/978-1-84996-169-1. [Online]. Available: http://dx.doi.org/10.1007/978-1-84996-169-1.

[10] I. Gurobi Optimization, *Gurobi optimizer reference manual*, 2016. [Online]. Available: http://www.gurobi.com.

[11] A. Chambolle, V. Caselles, M. Novaga, D. Cremers, and T. Pock, "An introduction to total variation for image analysis", working paper or preprint, Nov. 2009, [Online]. Available: https://hal.archives-ouvertes.fr/hal-00437581.

[12] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms", *Phys. D*, vol. 60, no. 1-4, pp. 259–268, Nov. 1992, ISSN: 0167-2789. DOI: 10.1016/0167-2789(92)90242-F. [Online]. Available: http://dx.doi.org/10.1016/0167-2789(92)90242-F.

[13] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging", working paper or preprint, Jun. 2010, [Online]. Available: https://hal.archives-ouvertes.fr/hal-00490826.

[14] N. Parikh and S. Boyd, "Proximal algorithms", *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014, ISSN: 2167-3888. DOI: 10.1561/2400000003. [Online]. Available: http://dx.doi.org/10.1561/2400000003.

[15] D. Cremers, B. Goldlücke, and T. Pock, *Variational methods in computer vision*, Sep. 2010. [Online]. Available: http://vision.in.tum.de/tutorials/eccv2010 (visited on 01/20/2017).

[16] G. Bradski, *Dr. Dobb's Journal of Software Tools*, 2000.

[17] Theano Development Team, "Theano: a Python framework for fast computation of mathematical expressions", *ArXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688.

[18] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda", *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008, ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500.

[19] D. R. W. Barr and P. Dudek, "A cellular processor array simulation and hardware prototyping tool", in *2008 11th International Workshop on Cellular Neural Networks and Their Applications*, Jul. 2008, pp. 213–218. DOI: 10.1109/CNNA.2008.4588680.

[20] P. Getreuer, "Rudin-osher-fatemi total variation denoising using split bregman", *Image Processing On Line*, vol. 2, pp. 74–95, 2012. DOI: 10.5201/ipol.2012.g-tvd.

List of Figures

List of Tables

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.