ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

DEPTH ESTIMATION IN STEREO BIOMEDICAL IMAGES VIA PROXY-SUPERVISED DEEP LEARNING

CANDIDATE

Claudio Bonetta

SUPERVISOR

Prof. Samuele Salti

CO-SUPERVISOR

Matteo Poggi, PhD

Anna Cesaratto, PhD

Academic year 2021-2022

Session 1st

To my beloved family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Depth estimation has long been an important enabling factor for many robotic-related tasks and many other applications involving SLAM-based pipelines. Many are the techniques that can be used to achieve this task: while some rely on depth sensors to acquire depth data, others make use of imaging systems that can either exploit stereo geometry by computing disparities (horizontal displacement of pixels between two hardware and software aligned cameras) or use temporal adjacent images in structure from motion (SfM) kind of algorithms.

Disparity map computation through stereo imaging systems is a commonly adopted solution in order to achieve depth estimation since it is very cost-effective and is able to reach a very high level of precision. A lot of bibliography focuses on the achievements in the field of autonomous navigation in both outdoor (such as autonomous driving) and indoor environments, but there is actually an humongous variety of tasks that are able to enjoy the benefits of a robust stereo-based disparity map computation pipeline.

This thesis will focus on disparity-map estimation in a 3D reconstruction pipeline in the biomedical field which, compared to others, suffers greatly from the presence of smooth and texture-less surfaces and reflective substances

(such as blood, saliva, mucus and other bodily fluids), which makes the common stereo matching based pipeline fail because of the absence of distinct key-points to be matched among the two cameras. Among all the applications that can be found in the biomedical domain we have decided to focus on dental imaging, concentrating our efforts on capturing synthetic targets of teeth.

In literature, it is not uncommon to see complementary procedures and/or sensors introduced alongside standard stereo matching whenever the task is deemed to be too hard to be faced by mean of pure vision-based approaches or whenever it is required an additional boost in performances. Such methodologies are usually referred to as *active stereo vision*. We can find among the most commonly deployed techniques/sensors LIDAR and ultrasonic sensors, which are often used in navigation based tasks (such as autonomous driving). Among the other techniques we can also find structured light projection, which consists in the projection of a light pattern onto the scene to create distinctive matching points in textureless areas (which are characterized by the lack of distinctive features). Structured light projection is the technique that will be used for image capturing in this thesis.

The main challenge that has to be faced in this particular context is the absence of precise labeled data: the commonly pursued approach of gathering ground truth data by relying on external sensors is not achievable in those cases were the working range is very small (in the order of the millimeters) and the required precision is very high (in the order of the micrometers). This is not an uncommon scenario in the biomedical domain where, in some of its applications, it is required to move around in narrow spaces, which constrains the movements to be very close to the surfaces. Small distances between camera and foreground can also be motivated by the necessity of capturing small structures in a more detailed manner. Since Machine Learning based approaches require a great amount and variety of examples in order to produce accurate results, a data gathering technique fit for this context is needed. The techniques adopted to generate ground truth data are reported in chapter 3.

The implementation of the stereo matching pipeline as been initially established through a baseline based on classical computer vision approaches. Thereafter, we have experimented and implemented a Deep Learning based stereo matching technique to attempt to gain an improvement over the baseline.

The main evaluation metric for this application is a 3D distance metric. This is motivated by the fact that we have access to high precision 3D models of the targets that are going to be scanned though our imaging system. This poses the necessity to produce as output not just disparities but also a 3D entity to compare with the 3D ground truth. This is achieved through a pre-built SLAM pipeline that takes as input the produced disparities and camera's intrinsic parameters and output a 3D mesh, which will be used for the comparison with the 3D ground truth.

1.1 Objectives

The objectives of the thesis can be summarized as follows:

- Implementation of a reliable technique to gather ground truth data in the absence of external sensors.
- Inspect and select a Deep Learning model capable of working in real-time and that is suitable to work with the imaging systems that is being used.
- Train the selected model with the gathered dataset.
- Introduce the trained model in the depth sensing part of the SLAM pipeline and evaluate its performance.

Chapter 2

Baseline

2.1 Standard Stereo Vision

The baseline for this project has been established using a non-learned disparity estimation technique. All standard stereo vision algorithms can be said to share more or less the same kind of pipeline. The pipeline is well defined in [23] and represented in Figure 2.1. It generally consists in a maximum of four steps (the actual number depends on the algorithm):

In the **Matching Cost Computation**, it is computed a distance function between the pixels of the left image and the right image's pixels that lie on the same row as the left's and that are within a predefined disparity range. Since a pixel-wise comparison would produce ambiguous results (both because of the redundancy typical of natural signals which might result in neighbouring pixels having very similar values and because of noises introduced by the imaging system), a commonly adopted approach is to compute the distance function over pixel-level features derived from a window centered on the said pixel. Being this a distance function, the smaller the result is, the more probable that the compared pixels match. The comparisons are performed over the scanline: pixels of the left image $I_L(x, y)$ are compared to the pixels on the

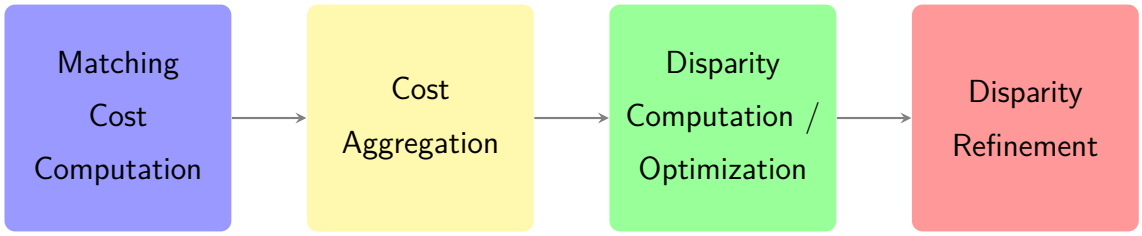


Figure 2.1: Standard Stereo Matching Pipeline.

right image $I_R(x, y)$ that lay on the same row:

$$C(x, y, d - D_{\min}) = f\left(I_L(x, y),\ I_R(x - d, y)\right) \ \text{ s.t. } \ d \in [D_{\min}, D_{\max}] \quad (2.1)$$

Where f is the distance function and D_{min} and D_{max} represent the search range over the scanline. This procedure produces as output a cost volume, also known as Disparity Space Image (DSI), which in the above formula is shown as $C(x, y, d)$. The cost volume is a data structure that holds, for each spacial location (x, y) and disparity value d, the cost needed to "approximate" $I_L(x, y)$ with $I_R(x - d, y)$ or, more formally, the distance between the considered pixels' features.

The **Cost Aggregation** phase is a step typical of global and semi-global algorithms that is not performed in local methods.
Since, with the exception of disparity discontinuities, disparities tend to show a smooth behaviour, introducing a smoothness constraint has been shown to be beneficial. The smoothness constraint is introduced in the form of an energy function that has to be minimized with respect to a disparity function $\boldsymbol{d}$:

$$E(\boldsymbol{d}) = E_{\text{data}}(\boldsymbol{d}) + \lambda E_{\text{smooth}}(\boldsymbol{d}) \quad (2.2)$$

Where E_{data} is the sum of all the pixels matching costs for the disparity $\boldsymbol{d}$:

$$E_{\text{data}} = \sum_{(x,y)} C(x, y, \boldsymbol{d}(x, y)) \quad (2.3)$$

And $E_{\text{smooth}}(\boldsymbol{d})$ encodes the smoothness assumption, whose value increases as the difference between the neighbourhood of a disparity's pixel increases:

$$E_{\text{smooth}} = \sum_{(x,y)} \rho(\boldsymbol{d}(x,y) - \boldsymbol{d}(x+1,y)) + \rho(\boldsymbol{d}(x,y) - \boldsymbol{d}(x,y+1)) \quad (2.4)$$

Where ρ is a monotonically increasing function. Selecting a suitable function is essential in order to assure the quality of the smoothness performed: for instance, quadratic functions tend to make the disparity $\boldsymbol{d}$ smooth everywhere. Functions that do not suffer from this issue are referred as *discontinuity-preserving*. After the definition of ρ comes the selection of the minimization algorithm. While many have been proposed (simulated annealing, etc.), thanks to their efficiency with respect to the other methods, *max-flow* and *graph-cut* became some of the most established global disparity estimation methods.
Independently of the selected global method, it can be proved that the 2D-optimization of the smoothness function is in practice an NP-hard problem. Another class of disparity estimation methods that proposes a solution to the high computational complexity of the global methods are the semi-global class of algorithms which, by means of dynamic programming, instead of 2D-minimizing the smoothness function, minimize its approximation by performing a 1D-minimization of scanlines over the cost volume. The aforementioned 1D-minimization done through dynamic programming is an operation that can be performed in polynomial time, thus solving the issues that the global methods commonly face.

The **disparity computation** is the process of extrapolating the disparity from the cost volume. This is normally achieved by means of winner takes all: since the cost volume contains a distance function among the matched pixels it is sufficient to take as disparity value $\boldsymbol{d}(x,y)$ the $\arg\min$ of the cost

curve at that spatial location:

$$\boldsymbol{d}(x, y) = \arg\min_{d} C(x, y, d) + D_{\min} \tag{2.5}$$

Since the result of winner takes all is the pixel-distance between the matched left and right images' pixels, the produced disparities are integer valued, which implies that a discretization happens in the process of forming the depth since it is formed by exploiting the inverse of the disparity:

$$Z = \frac{fb}{\boldsymbol{d}} \tag{2.6}$$

Where f is the focal length of the camera and b is the length of the stereo rig. This might not be a big issue for tasks such as robot navigation, but it is for reconstruction tasks which require a certain degree of precision.
In order to obtain real-valued disparities different techniques have been proposed such as iterative gradient descent and fitting a curve to the matching costs.

To the **disparity refinement** step belong all of the post-processing procedures that can be deployed in order to improve the raw disparity resulted from the previous steps.
Among the commonly deployed refinement techniques there is the *left-right consistency check* which allows, through the comparison between the left-to-right and the right-to-left disparities, to detect occluded and mismatched disparity values.
A median filter is generally applied to clean the disparity from the presence of spurious mismatches.

2.2 Semi-Global Matching

In this section we will introduce the stereo algorithm used to establish the baseline: Semi-Global Matching (SGM), highlighting the details regarding the parts of its implementation that differ from the original work [7]. SGM belongs to the class of semi-global approaches discussed in section 2.1 thus, as previously described, its cost aggregation step is very fast making it possible to use it in scenarios that require real-time computation capabilities.

2.2.1 Matching Cost Computation

As engineered by Heiko Hirschmüller [7], Semi-Global Matching uses as matching cost algorithm a pixel-wise matching cost calculation named Mutual Information. The usage of Mutual Information is motivated my the fact that, even thought robustness of matching increases with the size of the window considered, the implicit assumption done about constant disparity is violated at discontinuity hence, only the intensities at the considered points can be used to compute the matching cost. Mutual Information is defined by the entropy of two images as well by their joint entropy in the following way:

$$MI_{I_1,I_2} = H_{I_1} + H_{I_2} - H_{I_1,I_2} \tag{2.7}$$

Where for the continuous case:

$$H_I = \int_0^1 P_I(i) \log P_I(i) di \tag{2.8}$$

$$H_{I_1,I_2} = \int_0^1 \int_0^1 P_{I_1,I_2}(i_1, i_2) \log P_{I_1,I_2}(i_1, i_2) di_1 di_2 \tag{2.9}$$

Which can easily be rewritten for the discrete case as:

$$H_I = \sum_{i=0}^{255} P_I(i) \log P_I(i) \tag{2.10}$$

$$H_{I_1,I_2} = \sum_{i_1=0}^{255} \sum_{i_2=0}^{255} P_{I_1,I_2}(i_1, i_2) \log P_{I_1,I_2}(i_1, i_2) \tag{2.11}$$

For well rectified stereo pairs, the joint entropy is very low since the Mutual Information is high. As a pre-requisite in order to compute the joint entropy, one would need to warp one image over the other according to the disparity so that pixels are on the same location on both images. The fact that knowledge over the disparity is needed at cost compute time makes this method unusable. In order to avoid this issue an approximation of the joint entropy as sum over pixels using Taylor expansion, proposed by [13], has been used:

$$H_{I_1,I_2} = \sum_p h_{I_1,I_2}(I_{1p}, I_{2p}) \tag{2.12}$$

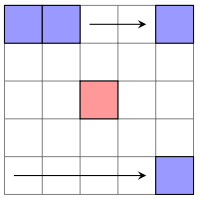
h_{I_1,I_2} is calculated from the joint probability distribution P_{I_1,I_2}. Convolution with a 2D Gaussian $g(i, j)$ effectively perform Parzen estimation.

$$h_{I_1,I_2}(i, k) = -\frac{1}{n} \log(P_{I_1,I_2}(i, k) \otimes g(i, k)) \otimes g(i, k) \tag{2.13}$$

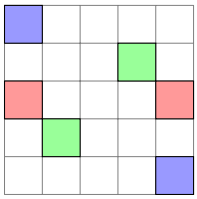
The operator $T[]$ defines the probability distribution of corresponding intensities: it is 1 if its argument is true, 0 otherwise.

$$P_{I_1,I_2}(i, k) = -\frac{1}{n} \sum_p T\left[(i, k) = (I_{1p}, I_{2p})\right] \tag{2.14}$$

However, Mutual Information is quite expensive to compute and not trivial to implement. Moreover, an extensive study on different matching costs [8] has shown in depth the limits of MI in difficult radiometric settings (such as local radiometric changes). The same study also elect as top performing cost computation method for stereo vision Census Transform [28][9], which is what we used for this project's baseline.



(a) Census Transform



(b) Center-Symmetric Census Transform

Figure 2.2: 2.2(a) **Standard Census Transform**: every element in the window is compared with the central one. 2.2(b) **Center-Symmetric Census Transform**: each element (x, y) is compared with the element $(W - x - 1, H - y - 1)$.

Census Transform

Census Transform [28] is an image operator used to build pixel-level features by comparing intensity values in a window centered on a given pixel.

In its classical form, the window's central pixel is compared with the others (either majority or minority comparison) and, for each comparison, it is outputted a Boolean, whose value depends on whether the comparison held positive or negative results.

The number of comparisons amount to $H \cdot W - 1$ where H and W are the height and width of the window respectively.

For performance reasons, the results of the comparisons are usually not stored in a boolean array, but as bits in a integer variable. The chosen window size is usually 9×7 so that the result of the 62 comparisons can be stored in a 64 bit integer.

In this work it has been used Center-Symmetric Census Transform [26] which, instead of performing comparisons against the central pixel, compares pixels at the opposite side of the window. This helps reducing the number of comparisons: $\lfloor \frac{H \cdot W}{2} \rfloor$ against the previous $H \cdot W - 1$. This also imply that the feature generated from a 9×7 window can now be stored in a 32 bit integer.

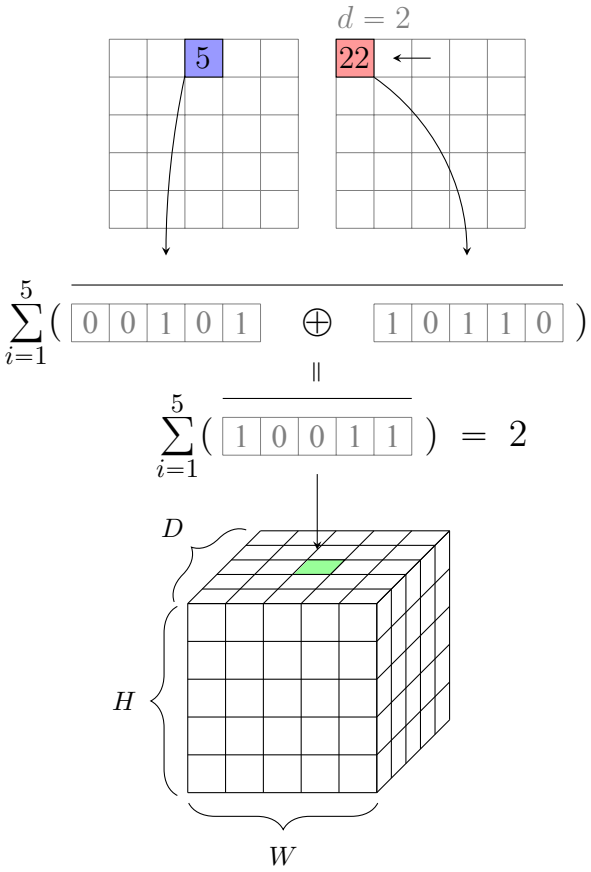


Figure 2.3: Visual representation of the Hamming Distance. W, H and D represent width, height and disparity range respectively where $D = W$. Being that the left and right feature represented in this example have only 5 bits (this is done for representational simplicity) we can expect that $C(x, y, d) \in \{k \mid 0 \leqslant k \leqslant 5,\ k \in \mathbb{N}\}$. Here the cost $C(0, 2, 2)$ is being computed.

Hamming Distance

Once the Census of the left and right image are computed, the feature distance between pixel's features lying over the same horizontal scanline is computed in order to build the unrefined cost volume.

The Hamming Distance between two Census' features is computed by mean of the *XNOR* operator and counting the 1 bits.

The description above can be formalized as follows:

$$C(x, y, d) = \sum_{i=1}^{31} (\overline{F_L(x, y) \ \oplus \ F_R(x - d, y)})_i \tag{2.15}$$

Where (x, y) is the spacial displacement, d is the disparity value, $C(x, y, d)$ are the initial costs, F_L and F_R are the left and right images' features generated by the Census Transform. Figure 2.3 shows a visual representation of how Hamming Distance is computed.

2.2.2 Cost Aggregation

Since pixel-wise matching costs are unreliable due to noise and other factors, the cost aggregation phase adds an additional smoothness constraint that penalize changes in neighbouring disparities. The pixel-wise cost and the smoothness constraint is expressed by the energy function $E(\boldsymbol{d})$ (that depends on the disparity $\boldsymbol{d}$).

$$E(\boldsymbol{d}) = \sum_{\boldsymbol{p}} \left(C\left(\boldsymbol{p}, \boldsymbol{d_p}\right) + \sum_{\boldsymbol{q} \in N_{\boldsymbol{p}}} P_1 T\left[|\boldsymbol{d_p} - \boldsymbol{d_q}| = 1\right] + \sum_{\boldsymbol{q} \in N_{\boldsymbol{p}}} P_2 T\left[|\boldsymbol{d_p} - \boldsymbol{d_q}| > 1\right] \right) \tag{2.16}$$

The first term is the sum of all the matching costs. The second term adds a small penalty for those pixels in the neighbourhood of $\boldsymbol{p}$ ($\boldsymbol{q} \in N_{\boldsymbol{p}}$) whose disparity values change a little bit with respect to $\boldsymbol{p}$ (a change of 1 in this case). The third term is similar to the second but adds a penalty for large disparity changes. Since higher disparity changes have to be penalized more than smaller ones it has always to be ensured that $P_2 \geqslant P_1$.

The problem of stereo matching can be now formulated as finding the disparity D that minimizes the energy function $E(D)$. Since this kind of 2D global minimization is, in many cases, an NP-complete problem, it is needed an approximation of the minimization process that would make the problem tractable. A solution can be found through the adoption of 1D minimization since it can be performed in polynomial time using Dynamic Programming. In order to avoid streaking effects given by the asymmetric combination of constraints having opposing directions, an equal aggregation of the costs from all the direction is needed. The aggregated cost $S(\boldsymbol{p}, d)$ for a pixel $\boldsymbol{p}$ and disparity

d is the summation of all the 1D minimum cost path that end up in pixel $\boldsymbol{p}$ at disparity d. While the path will be straight in the spacial dimension it is non-straight in the disparity dimension depending on the disparity changes. For the correctness of the aggregation only the aggregated cost information needs to be retained, while information about the followed path can be discarded.

The cost $L_{\boldsymbol{r}}^{'}(\boldsymbol{p}, d)$ along a path traversed in the direction $\boldsymbol{r}$ of the pixel $\boldsymbol{p}$ at disparity d is defined recursively as:

$$L_{\mathbf{r}}^{'}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min \begin{cases} L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d) \\ L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d-1) + P_1 \\ L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d+1) + P_1 \\ \min\limits_{i} L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ i) + P_2 \end{cases} \tag{2.17}$$

The cost $C(\boldsymbol{p}, d)$ adds the cost at position $\boldsymbol{p}$ while the rest of the equation adds to the current cost $L_{\boldsymbol{r}}^{'}(\boldsymbol{p}, d)$ the minimum cost of the previous pixel along the followed path $\boldsymbol{p} - \boldsymbol{r}$, including the appropriate penalties for discontinuities. Since by following the given definition of the recursive formula the values of the costs monotonically increase along the path, in order to keep the computation bounded in a range of values, it is possible to subtract the minimum path cost of the previous pixel.

$$L_{\mathbf{r}}^{'}(\mathbf{p}, d) = C(\mathbf{p}, d) - \min_{k} L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ k) + \min \begin{cases} L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d) \\ L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d-1) + P_1 \\ L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ d+1) + P_1 \\ \min\limits_{i} L_{\mathbf{r}}^{'}(\mathbf{p} - \mathbf{r},\ i) + P_2 \end{cases} \tag{2.18}$$

Adopting this definition will allow to limit the value of L to the upper bound: $L \leqslant C_{\max} + P_2$ without changing the path. The final cost value is determined by summing the results of the aggregation of all the paths:

$$S(\boldsymbol{p}, d) = \sum_{\boldsymbol{r}} L_{\boldsymbol{r}}(\boldsymbol{p}, d) \tag{2.19}$$

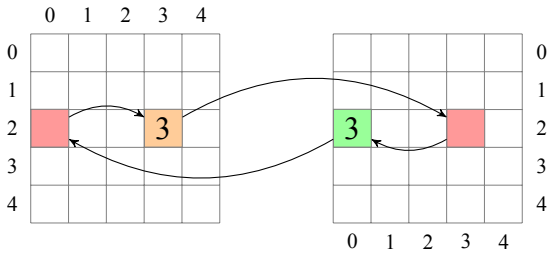


Figure 2.4: Visual representation of the Left-Right Consistency Check. When the matching is consistent (there are neither occlusions nor mismatches) it would be possible, in principle, to visualize it as a cycle between the left-to-right and right-to-left disparity.

The paths should be at least 8 or 16 to have a good coverage of the 2D image.

The final costs values are bounded by $S \leqslant \#\text{paths}\,(C_{\max} + P_2)$.

The computation of $L'_{\mathbf{r}}(\mathbf{p}, d)$ takes $O(D)$ at each pixel, where D is the disparity range. Since each pixel is visited $\#$paths times the total complexity is $O(WHD)$.

2.2.3 Disparity Computation

From the refined costs, in order to extract the disparities, it is used a Winner Takes All strategy.

In order to choose the disparity value $\boldsymbol{d}(\boldsymbol{p})$ the index of the minimum cost for $\boldsymbol{p}$ is taken. It can be mathematically expressed as:

$$\boldsymbol{d}(\boldsymbol{p}) = \arg\min_{d \in D} \left(C(\boldsymbol{p}, d)\right) \tag{2.20}$$

Simply choosing the costs' minimum index as disparity values would result in having an integer disparity, which is an undesirable property in certain scenarios where high precision is required (such as 3D reconstruction) since integers disparities ends up discretising the depth-map (as already mentioned in section 2.1).

In order to patch this issue it has been proposed a sub-pixel estimation

technique that consists in fitting a quadratic curve to the cost curve in correspondence of the minimum cost and its direct neighbours and taking the minimum of such curve as the selected disparity value.

From the very same DSI used to compute the left-right disparity it is also possible to extract the right-left disparity.

$$\begin{aligned} \boldsymbol{d}_R(x,y) &= \arg\min_d \{C(x,y,0), C(x+1,y,1), ..., C(x+D,y,D)\} \\ &= \arg\min_d \{C(x+d,y,d) \mid d \leqslant D,\ x+d < W\} \end{aligned} \tag{2.21}$$

Extracting this information might be useful for the refinement of the left-right disparity (more on this in subsection 2.2.4).

2.2.4 Disparity Refinement

In order to attenuate the issue of the presence of outliers a median filter is applied on both left and right disparity map.

After that, in order to filter out invalid disparity values, a Left-Right Consistency Check [4] is applied: each disparity value of the left disparity map $\boldsymbol{d}_L(x,y)$ is compared with its corresponding value on the right disparity map (present on the same row y at column $x - \boldsymbol{d}_L(x,y)$), which, in order to be consistent, should be the same as the left's (or at least different by at most a predefined threshold). This process enforces an uniqueness constraint by permitting one to one mappings only.

What stated above can be formalized in the following way:

$$\boldsymbol{d}_L(x,y) = \begin{cases} \boldsymbol{d}_L(x,y), & \text{if } |\boldsymbol{d}_L(x,y) - \boldsymbol{d}_R(x - \boldsymbol{d}_L(x,y), y)| < \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{2.22}$$

Chapter 3

Dataset Generation

In order for Supervised Deep Learning approaches to be able to function in a proper manner, they need the availability of a certain quantity of labeled examples. The actual quantity needed varies greatly depending on whether pre-trained weights for a given model are made available. Being able to work with pre-trained models allows to easily adapt them to the target domain by means of simple fine-tuning, which requires fewer examples with respect to a pre-training. Hence, being able to gather examples associated with precise ground truth data is a core activity in the construction and deployment of a functioning Deep Learning-based pipeline.

The context revolving around this work is very peculiar: the distances between the cameras and the foreground is very small (in the range of 10-20 mm) and, being the target precision of this application in the order of the μm, makes difficult finding among the commercially available sensors one that could fit these requirements.

The usage of sensors in order to gather ground truth data is the core approach used in the greatest majority of disparity estimation works.

The impossibility of gathering ground truth data through sensors constitute a major issue for this application and creates the need to find alternative techniques in order to collect them.

A throughout revision of the available literature highlighted mainly two ways

of securing ground truth data without the usage of sensors:

- Synthetic data generation.
- Proxy-labeling.

In this section, we are going to thoroughly examine the proposed data gathering techniques.
Moreover, a zero-shot approach has been attempted as well by leveraging the generalization capabilities of the model called *RAFT-Stereo* [16] (which will be the model that we are going to use in this work and that will be presented in section 4.1) without excellent results. Even though the zero-shot approach highlighted the great generalization capability of the said model by exhibiting results that show some degree of visual consistency with the results presented by SGM, it is still far off from being precise, and since predictions on temporally adjacent images do not present much consistency, the 3D reconstruction through the SLAM-based pipeline has not been made possible and, consequentially, neither the 3D evaluation of this approach (see chapter 5 for more details about the 3D evaluation). This result, though, was to be expected, since not only the domain of the image used to train the model is very different from the one used to test it (trained on Scene Flow [17] and used to make inferences over teeth images), but also the test time images, being captured using a projected pattern, can be said to go out from the natural images manifold.

3.1 Synthetic Data Generation

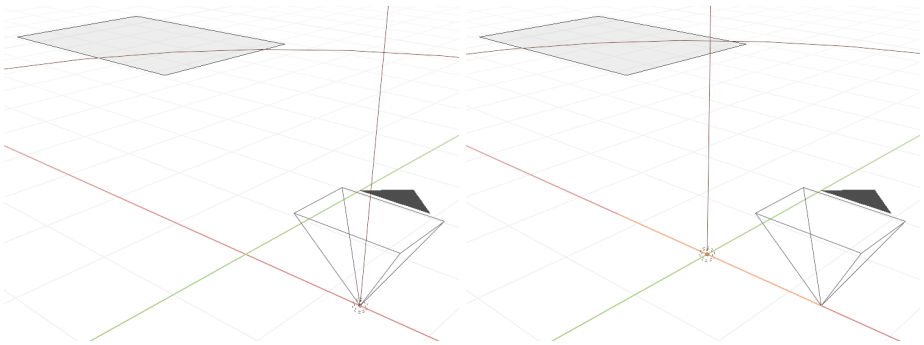
Following the observation made over the zero-shot performance of *RAFT-Stereo* [16] (more on it in Section 4.1) pre-trained over Scene Flow [17], it might have been reasonable to think that fine-tuning it over a dataset that resembles more the target domain could be beneficial to the task even without training it directly over the images belonging to the target domain. In order to follow this objective, it has been devised a pipeline that could generate a pair

of synthetic stereo images obtained through a rendering engine by digitally projecting over a mesh of teeth an artificially generated dotted pattern.
The rendering engine used for this work is Blender's Cycles [3] (path tracing based renderer).
The pipeline that allowed to generate the synthetic dataset is as follows: creation of the scene's elements (stereo rig, structured light projector, teeth's mesh, etc.), creation of an animation by moving the virtual stereo rig randomly in the scene and the rendering of the animated scene. The rest of this section will go into detail on all the aforementioned pipeline's steps.
It is worth to highlight that one of the greatest merits of adopting such a pipeline is the fact that the generated disparities are very precise and that they are not affected by the presence of occlusions since they are obtained from the bidimensional reprojection of the 3D view.

All of the steps needed to generate the synthetic dataset, from the setup of the virtual stereo rig to the generation of camera's random poses have been automated using Blender's Python API.

3.1.1 Stereo Rig Setup

In order to acquire the synthetic stereo images, it is necessary to setup the stereo rig with the left and right cameras and the structured light projector.
The first step in order to properly setup a camera in Blender is to specify the camera's intrinsic parameters: focal length and size of the sensor. Normally, real cameras would present also some form of lens distortion such as radial and/or tangential distortion but, since the aim is to simulate calibrated and rectified cameras, there is no need to simulate these properties as well.
Blender also allows to automatically generate another camera in order to simulate stereo vision without having to manually create a second camera object. The created camera is a phantom object, in the sense that it is not directly manageable in the object space but it is managed indirectly by acting on the



(a) First rig setup (b) Final rig setup

Figure 3.1: 3.1(a) is the first stereo rig setup where the light projector's position coincide with the camera's. 3.1(b) is the last configuration where the light projector is placed between the two cameras. The left camera is represented by the pyramidal object on the right while the right one by the rectangular light-gray object on the left. The light is represented by the spherical dotted object laying on the axis.

main camera (which is the left one in our case). This relieves the user both from handling a more complex rendering procedure and managing the relative positions of the two cameras in a way that allows their image planes to be co-planar. Moreover, it is also possible to specify the size of the baseline.

After the aforementioned steps comes the positioning of the structured light projector on the rig. Two configurations have been tried: in the first, the position the light is the same as the left camera. This has the advantage of being a particularly simple configuration since it is easy to constrain the light to have the same location and rotation as the left camera, but it has been observed distortions of the light captured by the right camera which is not coherent with the left view: this phenomena is easily explainable as a parallax effect.

In order to solve this issue it has been deemed necessary moving the structured light projector in a position in-between the left and right camera. Placing the structured light in this position is not as straightforward as it was in the previous solution since its location cannot be the same of the left camera anymore.

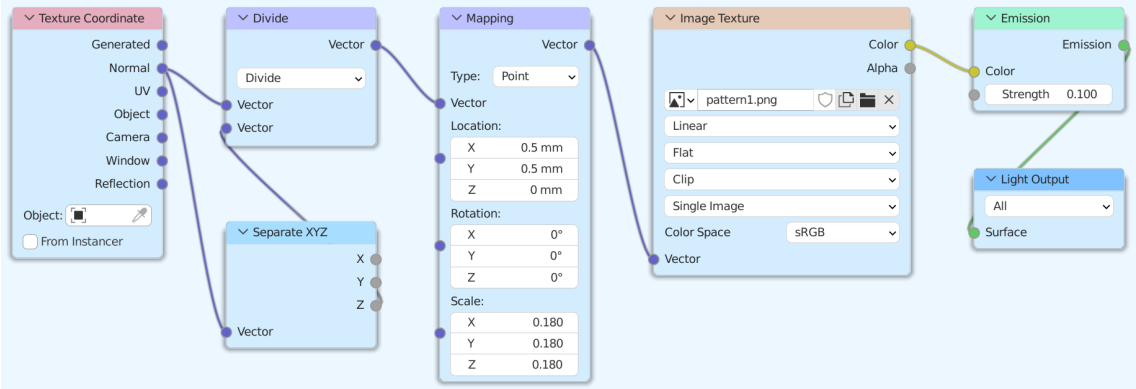


Figure 3.2: Structured light projector's shader nodes.

In order to automatically compute the structured light projector's position the following setup has been proposed: a segment composed of three vertices (a central one and two at the extremities) with the same length as the baseline has been added to the scene and the camera has been placed on one of its extremities, in this way the phantom right camera will be automatically placed at the other side of the segment. After that, it is sufficient to add the structured light projector in the segment's middle vertex and, through Blender's integrated constraint system, constraint both left camera and structured light projector to follow location and rotation of the segment's vertex they have been placed on (this has been implemented through Blender's *Child Of* constraint). It is now possible to move the newly composed stereo rig by simply moving the segment.

3.1.2 Structured Light Projector Setup

Shader Nodes

In Blender's Cycle it is possible to setup a projector by relying on a light object with *spot* type of light and by adding shader nodes to alter light properties. It is in fact possible, through the shader nodes, to apply an image as texture to the emitted light rays.

As it can be seen in Figure 3.2, the node tree (the set of shader nodes

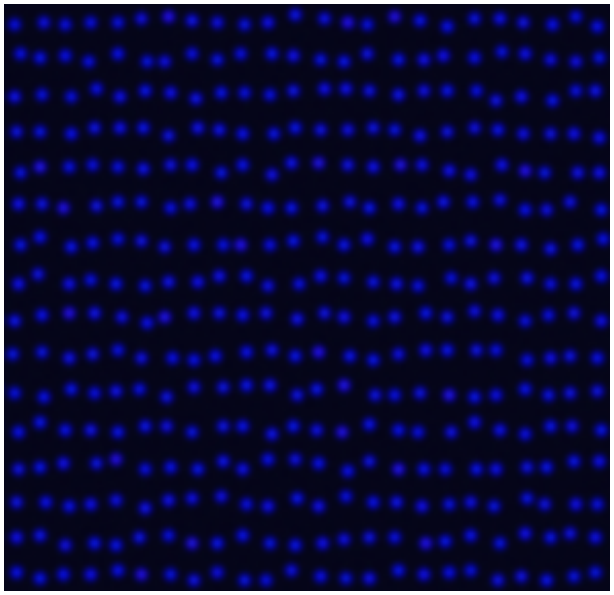
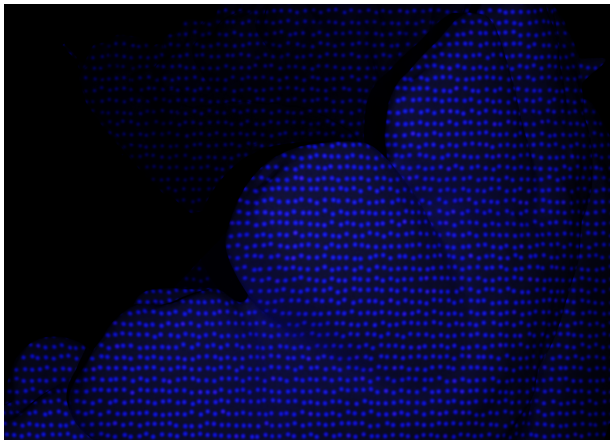


Figure 3.3: Synthetically generated pattern.

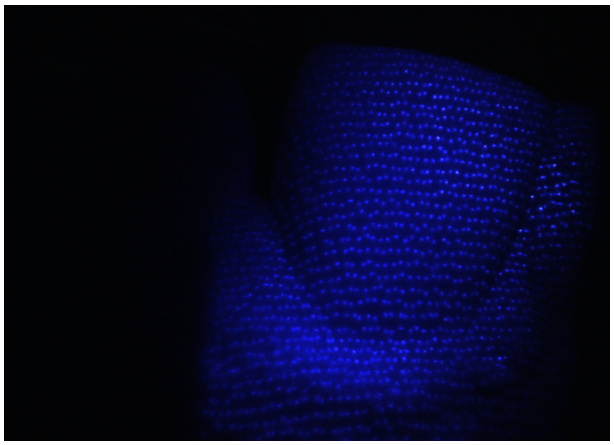
and the links among them) of the structured light projector is composed by a total of seven nodes. The input node is the *Texture Coordinate* node. Among *Texture Coordinate*'s outputs *Normal* is used. *Normal*, instead of returning the location of the shaded point, returns its normal: location on the unit sphere centered on the shaded point. The *Texture Coordinate*'s output is then divided by its z component so as to project the 3D vector to the bi-dimensional plane so that the projected pattern is only related to its xy components but not the z. The *Mapping* node adjusts the location of the projection, which is initially offset by 0.5 mm, and let us modulate the scale of the projected image. The *Image Texture* node is the projection pattern. The *Emission* node adds Lambertian luminous shader for light output. The last node is the *Light Output* which is responsible of projecting the image onto the scene.

Pattern Image Generation

The image of the pattern that has been loaded into the digital projector (the *Image Texture* node) has been synthetically generated through a Python script.
In order to generate a geometrically accurate synthetic version of the pattern, it is necessary to observe that, on the real structured light's pattern, the



(a) Synthetic image



(b) Real image

Figure 3.4: 3.4(a) is the synthetic image rendered with Blender using the digitally generated pattern of Figure 3.3. 3.4(b) is a real example of an image taken with the projected pattern.

blue dots are not arranged precisely in a grid but present elements of pseudo-randomicity. It is also worth to notice that the vertical distance between the dots is higher than the horizontal one. Reproducing this effect is quite straightforward as it is only necessary to sample regular grid indices with higher intensity on the horizontal axis and with less frequency in the vertical one. In order to implement the pseudo-randomicity, it has been added the result of the sampling on a Binomial distribution to the regularly-sampled points on both axis (to emulate randomicity in both directions). The Binomial distribution's number of Bernoulli trials, has been taken to be inversely proportional to the

axis-wise grid's points' sampling frequence: hence, the random offset is generally higher in the vertical axis with respect to the horizontal one.
Regarding the photorealism, reproducing the drooling effect has proven to be a major hurdle. We have decided to approach this issue by applying a Gaussian smoothing to the previously placed points to give them a more natural effect. In Figure 3.3 it can be observed the end result, while Figure 3.4 is the comparison between an image rendered with the synthetic pattern and one taken with the real projected pattern.

3.1.3 Random Poses Generation

The generation of camera poses is not straightforward as it present some constraints. The constraits can be summarized as follows:

- Constraints relative to the **rig's location**: the camera should not appear beneath the mesh or outside of the working range (which is between 10mm and 20mm).
- Constraints relative to the **rig's rotation**: the cameras have to always look at the teeth.

The first random poses generation approach relied on Blender's internal constraint system to shape the aforementioned requirements.
In order to setup the integrated constraints that shape the rig-mesh distance requirements, it must be created an *Empty* object that sticks to the surface of the mesh (through a *Shrinkwrap* constraint) and follows the rig as it moves around in the space (which can be done by copying into the Empty object's location the rig's location's drivers). Two *Limit Distance* constraints will then limit the maximum and minimum distance between the rig and the Empty object (and, thus, the mesh). The fact that the rig should not appear beneath the mesh is done by placing the mesh aligned with the Z axis and then a *Limit*

Location constraint is imposed to the rig so that it cannot go below certain values of Z (in this way the rig's location is limited only to the upper part of the mesh). The choice of the initial random location is done by generating a 3D random point with maximum and minimum for each axis dictated by the maximum and minimum location of the mesh' vertices offsetted by a value. Using an offset allow the rig to reach the maximum working distance at every point of the mesh. The initial location obtained through this procedure will be then adjusted by the integrated constraint system to obtain the final position.

The constraint relative to the rig's rotation are handled by creating a curve that passes through the teeth (as shown in Figure 3.5) and convert the said curve to a mesh (which are two distinct entities in Blender). After that, similarly to how it was done for the *Limit Distance* constraints, we need to create an *Empty* object and add a *Shrinkwrap* constraint to it so that it will stick to the created curve. The newly created Empty object needs, while remaining on the curve, to follow the movements of the rig as it moves around the scene: this can be done simply by copying into the Empty object's location the rig's location's drivers. Lastly, a *Track To* constraint needs to be added to the rig having as target the newly created Empty object, in this way the cameras will always be looking at the teeth.
While it is straightforward to leverage Blender's integrated constraint system in order to shape these constraints, it is not an advisable approach since the interaction of multiple constraints might negatively affect the final effective position (where final effective position means the position obtained as a result of applying the constraints to the initial random position). In fact, the results given from the first approach were not deemed appropriate, reason being that many frames had to be discarded because errors would occur with the choice of the camera's angle, resulting in the camera pointing to empty areas.

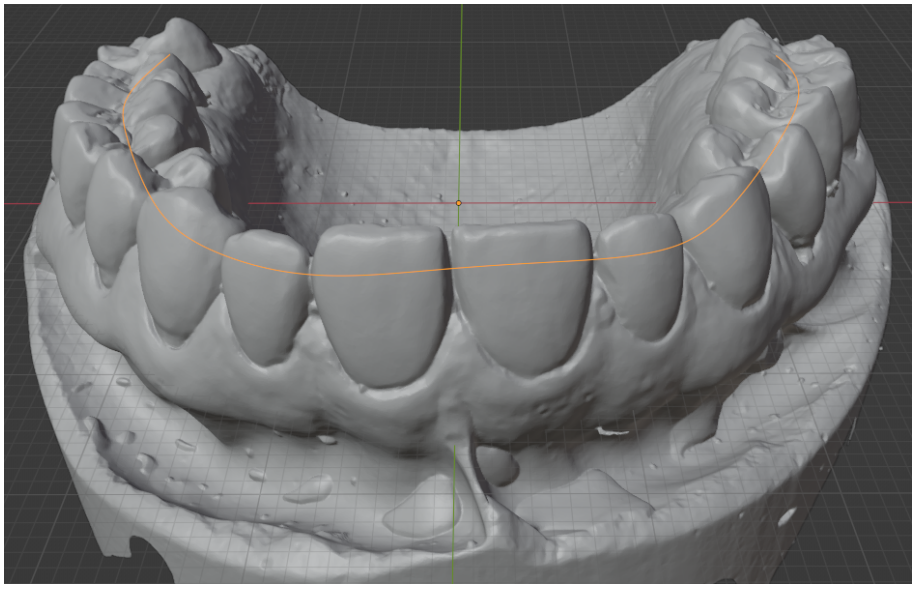


Figure 3.5: In yellow is the line used to orient the rig to face the teeth.

In the light of the results obtained with the first approach, it has been decided to create another random poses generation algorithm that could make less use of Blender's internal constraint system. While the considerations previously done about the rotation have been mostly retained, the generation of the rig's location has been changed altogether.
In order to generate a position that is not beneath the mesh, we take a random mesh' vertex and extract its location and normal vector. The rig is then placed in the vertex location and pushed with a certain intensity in the direction of the normal vector. The intensity of the push is random and its interval is determined according to the working range we wish to work with. At the end, a check on the euclidean distance with the closest mesh' vertex is performed and in case it is outside of the working range, the procedure is repeated (selection of the random vertex, etc.).
The procedure has been observed empirically to give good results and only few corrections to the generated poses were needed.

It is worth to mention that, instead of starting the rendering as the poses are being generated, it has been preferred to first generate all of the poses and then

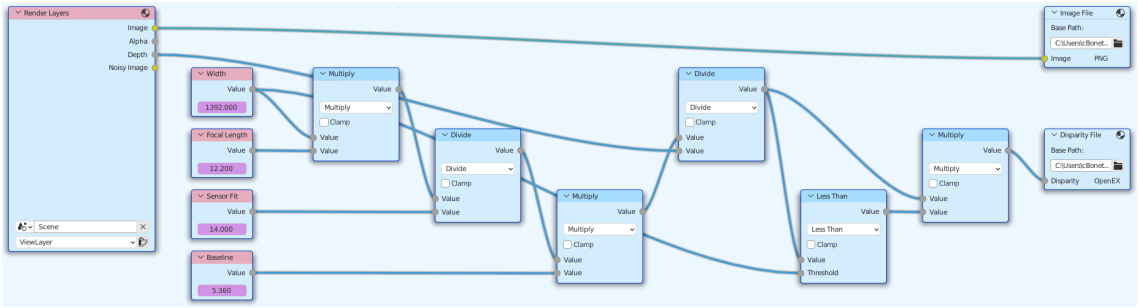


Figure 3.6: Compositor node tree.

leverage Blender's animation system to perform the rendering. This choice has been made mainly because during the execution of a Python script Blender's GUI remains unresponsive. Therefore, rendering immediately after the generation of the pose would result in long periods of non-responsiveness of the GUI, making it difficult to be aware of the state of progress of the rendering operations. Moreover, using the generated poses to create an animation enable the inspection of the generated poses before launching the rendering operation, allowing to skip (or at least lighten) the post-rendering data inspection (none of the generated examples have been discarded). Lastly, with animations, the naming of the rendering's output file is done automatically, whereas otherwise it would have had to be handled manually.

3.1.4 Rendering

The output of the rendering procedure are:

- The stereo images (the input of the model).
- The disparity (labels used to supervise the training).

The issue is that Blender does not have an integrated option that allows to return the disparity as a byproduct of the rendering procedure, but it does return the depth instead. By having information about the camera's intrinsic parameters, we are able to retrieve the disparity from the depth.

In order to retrieve the disparity from the rendering procedure we use the

Compositing Nodes, which are Blender's internal structures that allow to fiddle with the output of the rendering.

The first step would be the activation of the *Z pass* which allows to obtain the depth as output of the rendering. After that, we move on to the configuration of the *Compositor* node tree. Since we are now able to get the Z-depth relative to the camera position, we just need to replicate the formula that converts depth to disparity:

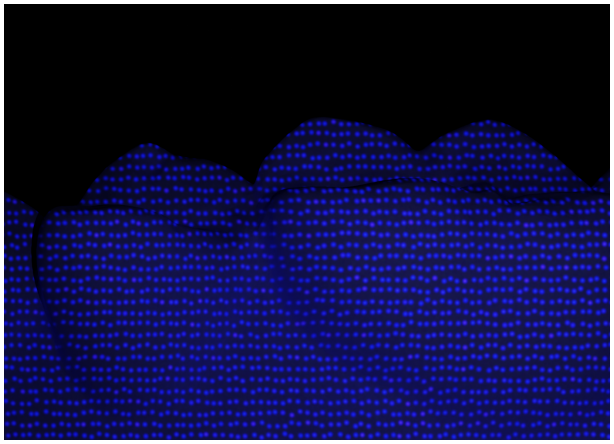
$$\text{disparity} = \frac{f_{\text{px}} \times b}{\text{depth}} \tag{3.1}$$

Where the focal length f_{px} is expressed in pixels, not millimeters. In order to perform the conversion:

$$f_{\text{px}} = \frac{f_{\text{mm}} \times W}{\text{sensor\_width}_{\text{mm}}} \tag{3.2}$$

In the node tree, the input node *Render Layers* is what allow us to retrieve depth information (through its *Depth* output). Using *Render Layers*'s output, to retrieve disparity information, it is only needed to replicate the aforementioned formulas using the *Compositor*'s nodes as depicted in Figure 3.6. Since the disparity is obtained from the depth, it might happen, in case the rig is too close to the mesh, that the disparity's formula gives values that are superior to the image's width. This particular case is handled by masking all those pixels whose value is greater than the image's width (by setting them to 0). This is done in the node tree by using the *Less Than* node, which receives the disparity map as input. *Less Than* node outputs a binary mask that has set to 1 all of those pixels whose disparity values verify the condition: $\boldsymbol{d}(x, y) < W$, 0 otherwise. The produced mask is then multiplied to the raw disparity to mask the disparity values higher than the width. This particular case, though, should not happen if the distances in the scene are well managed.

Saving the output is handled by *File Output* node. While saving the stereo images as .png files is sufficient, this does not apply to disparities as well.



(a) Synthetic image

(b) Disparity

Figure 3.7: In 3.7(b) is represented an example of synthetic disparity used for supervision. In order to obtain this image, the disparity values have been normalized with the minimum and maximum disparity values. 3.7(a) is the color image related to 3.7(b).

Since disparities are composed by floating point values (because we are dealing with sub-pixel disparity values) whose range goes potentially up to the width of the image (depending on the size of the input image it might be way more than what a simple 8 bit image could handle), it is necessary to find a file format that preserves these characteristics. Among the ones offered by Blender the only one capable of working with floating point values is *OpenEXR* (Open EXtended dynamic Range), but it is not a conventional format to work with. The file format most affirmed in literature to deal with

floating point images is *PFM* (Portable Float Map). As such, in order to obtain disparities in .pfm format, we save the disparities in Blender in .exr and then, using Python, we convert them in .pfm. This can be easily done through OpenCV:

```
import os
os.environ["OPENCV_IO_ENABLE_OPENEXR"] = "1"
import cv2

def exr2pfm(path_to_exr_file, out_dir_path):
    disparity = cv2.imread(path_to_exr_file,  cv2.
        IMREAD_ANYCOLOR | cv2.IMREAD_ANYDEPTH)[:,:,0]
    cv2.imwrite(os.path.join(out_dir_path, path_to_file.
        split(".")[0]+".pfm"), disparity)
```

The .exr file outputted by Blender is automatically set to be a color image. Since the disparity computed in Blender with the aforementioned procedure is represented a single channel image, the three channels of the .exr file are all equivalent.

In Figure 3.7 can be observed an example taken from the synthetic dataset collected with the aforementioned procedure.

3.2 Proxy-Labeling

The supervision given by synthetically generated images, even though precise, still suffers from the domain gap issue. The results given by the synthetic approach shows how much photorealism is necessary in this domain.

Among the reviewed literature, it has been found in the biomedical domain, which greatly suffers from the lack of ground truth data for the task of disparity/depth estimation, that a technique commonly known as proxy-labeling can be deployed to make up for the lack of sensors apt to gather ground truth data in difficult contexts such as laparoscopy, etc.

Proxy-labeling is a technique used to gather ground truth disparity data by relying on a traditional disparity computation method. The disparity gathered from the traditional method (that for this work will be the SGM baseline) will be filtered by some confidence measures [21], in this way we will be able to create a sparse supervision in which only those disparity values that are considered more reliable by the chosen confidence measures are kept, while all of the others are discarded.

There are many confidence measures that can be applied to a disparity, and they can be categorized in different types depending on which segment of the disparity estimation pipeline they use in order to decide whether a disparity value is reliable or not.

The macro-categories in which the confidence measures can be subdivided are:

- Minimum cost and local properties:

  Confidences that exploit only local information of the cost curve. They often rely on the global minimum and the second local minimum but, since the second local minimum is not always available (as in the case of the ideal cost curve), in its place the second global minimum is used (in this case the confidence measure used takes the prefix *naive*). The second global minimum is always defined.

- The entire cost curve:

  Confidences that relies on the entire cost curve. Most of them make use of the minima introduced in *minimum cost and local properties*.

- Left-right consistency:

  While the target of the disparity estimation's task is more often than not the computation of the left-to-right disparity (mainly because of an affirmed convention), computing the right-to-left disparity from the DSI is nevertheless a straightforward operation. Having the right-to-left disparity as well, allows to check for correspondences between the two

views by relying on some cues.

- Disparity map analysis:

  Filtering is performed by relying exclusively on disparity cues and/or features. No information extracted from the DSI is used.

- Reference image analysis:

  Class of confidence measure computed by relying on features computed over the input image and/or other priors (such as the position of the pixel with respect to the border, etc.).

- Self-matching:

  Self-matching analyzes the distinctiveness of a pixel with respect to its neighbours lying on the horizontal scanline. Self-matching methods are implemented by running a stereo matching algorithm over two instances of the left image and adopting a disparity range of $D = [-D_{\max}, D_{\max}]$ centered over a given point $\boldsymbol{p}$. This class' measures make use of the produced DSI to output a confidence.

- Semi-Global Matching measures:

  This class of confidences can be used only with SGM and compute a score based on cues available in its pipeline.

Among the categories mentioned above only *minimum cost and local properties*, *the entire cost curve* and *left-right consistency* have been used as deemed more fit for this particular context.
Minimum cost and local properties and *the entire cost curve* bases the selection of reliable disparity values on the analysis of the cost volume (disparity space image). In particular, for each spacial location some form of evaluation of the curve of the costs is performed. In Figure 3.8 can be observed an example of ideal and ambiguous cost curve. Ideally, the cost curve should present a single distinct global minimum, while all of the other costs have high values. In practice, a real cost curve resemble more the ambiguous one.

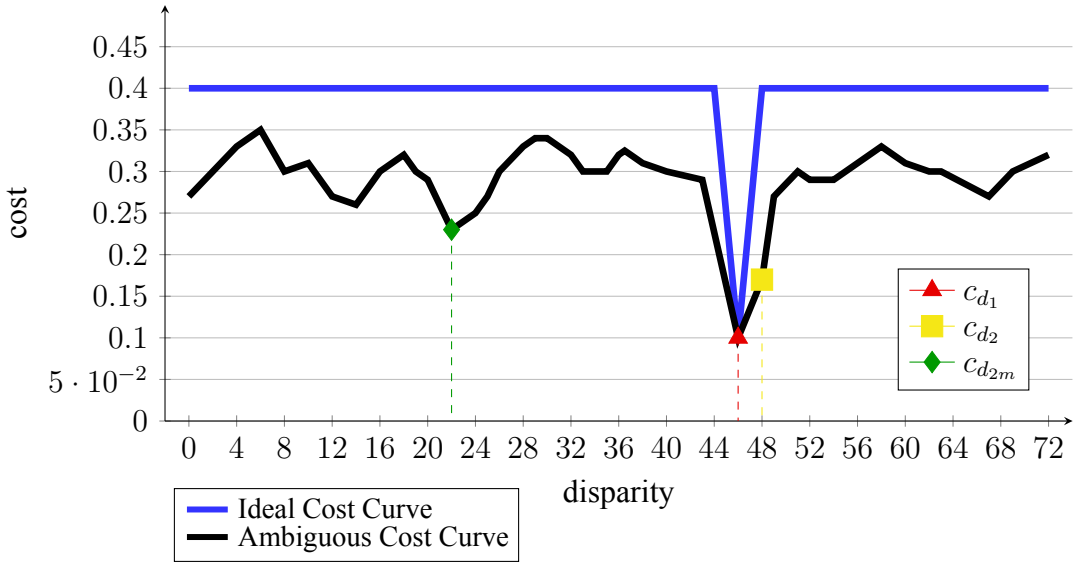


Figure 3.8: In the plot are depicted an ideal cost curve and an example of a realistic cost curve. c_{d_1}, c_{d_2} and $c_{d_{2m}}$ are respectively the global minimum, second global minimum and the second local minimum.

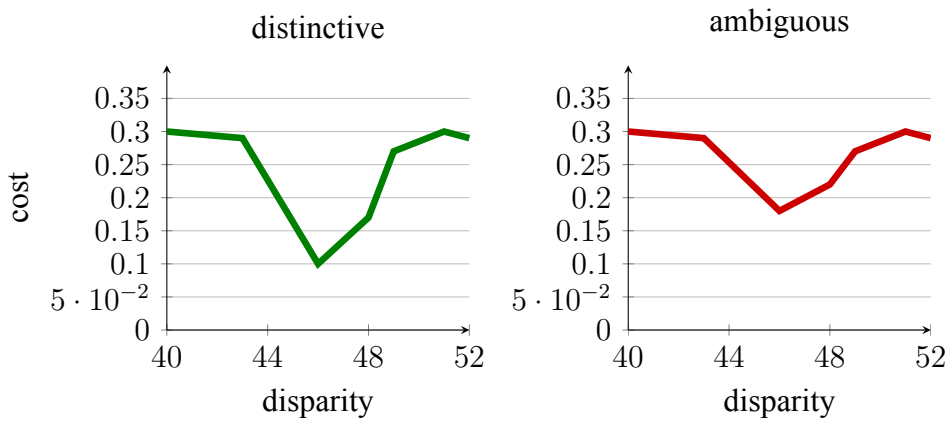
For notational convenience we define $c_{d_1}(\boldsymbol{p})$ as the minimum of a cost curve, $c_{d_2}(\boldsymbol{p})$ as its second minimum and $c_{d_{2m}}(\boldsymbol{p})$ as its second local minimum.

Following, it will be given a throughout review of the confidence measures that have been selected with their relative mathematical formula:

Matching Score Measure (MSM) [5]: the lower the cost is, the more probable the matching is precise.

$$\text{MSM}(\boldsymbol{p}) = -c_{d_1}(\boldsymbol{p}) \tag{3.3}$$

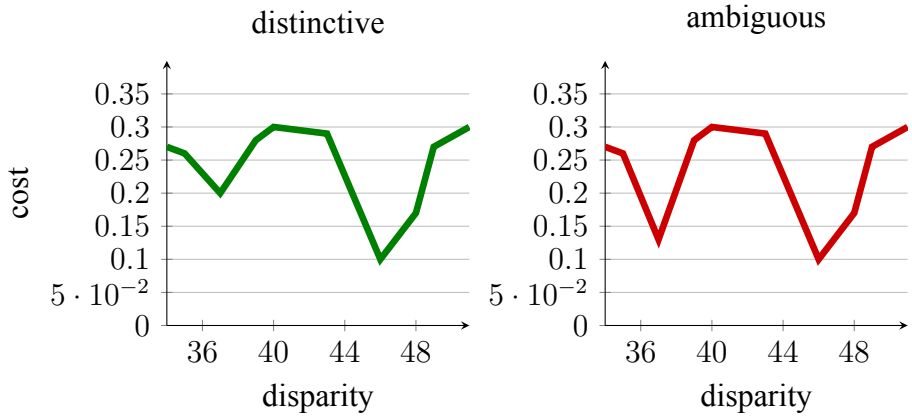
For instance, it is easy to see in the two plot below that even though they both have minimum in 46 the quality of the match of the first plot is much higher than the second's.



Maximum Margin (MM): the greater the difference between the global minimum and the second local minimum is, the less ambiguous the cost curve should be.

$$\mathrm{MM}(\boldsymbol{p}) = c_{d_{2m}}(\boldsymbol{p}) - c_{d_1}(\boldsymbol{p}) \tag{3.4}$$

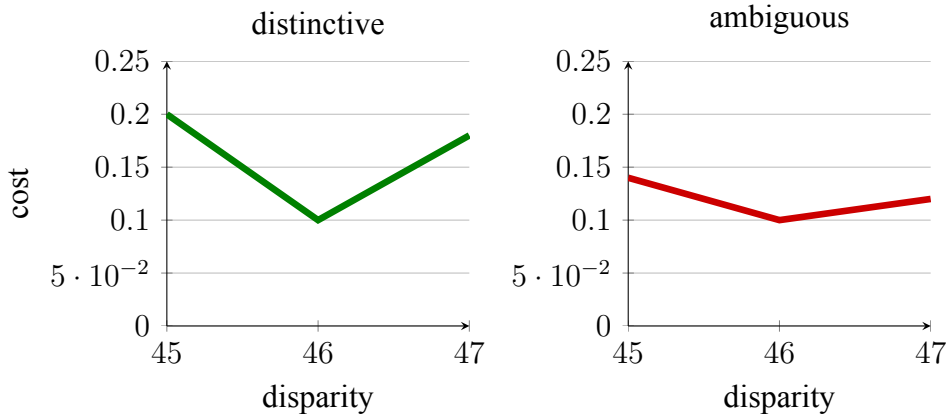
Having a local minimum with a value very similar to the global minimum represents uncertainty since a small difference in the matching results might be the outcome of undesired factors (such as noise). A great difference between the global minimum and the second local minimum is a measure of distinctiveness (as shown in the plots below).



Curvature (CUR) [5]: a pronounced steepness of the curvature surrounding the global minimum implies a great distinctiveness of the match.

$$\mathrm{CUR}(\boldsymbol{p}) = -2c_{d_1}(\boldsymbol{p}) + c_{d_1-1}(\boldsymbol{p}) + c_{d_1+1}(\boldsymbol{p}) \tag{3.5}$$

The plots below clearly shows the contribution of Curvature: having a relative poorer matching results of the direct scanline neighbours of a certain pixel allows the matching to be more distinctive.



Winner Margin (WMN) [24]: similarly to maximum margin the global minimum and the second local minimum are compared, but this time the result is normalized by the summation of the costs of the entire cost curve.

$$\mathrm{WMN}(\boldsymbol{p}) = \frac{c_{d_{2m}}(\boldsymbol{p}) - c_{d_1}(\boldsymbol{p})}{\sum_{i \in D} c_i(\boldsymbol{p})} \tag{3.6}$$

Average Peak Ratio (APKR) [14]: is the sum of the ratios of the costs computed in the neighbourhood of a given pixel $\boldsymbol{p}$ and positioned on the disparity dimension at the index of its second local minimum cost and global minimum cost respectively.

$$\mathrm{APKR}(\boldsymbol{p}) = \sum_{\boldsymbol{q} \in N(\boldsymbol{p})} \frac{c_{d_{2m}(\boldsymbol{p})}(\boldsymbol{q})}{c_{d_1(\boldsymbol{p})}(\boldsymbol{q})} \tag{3.7}$$

This confidence measure calls for coherence about the positioning of the global minimum cost (which aligns with the smoothness assumption) and incoherence about the positioning of the second local minimum cost (in this way it is more probable that its presence is given by spurious and undesired factors such as noise).

Left-Right Consistency (LRC) [4]: for more details, please refer to subsection 2.2.4

Moreover, other than the aforementioned confidence measures, the disparities have been masked with the thresholded input image: since the target domain's input images present uninformative black areas wherever the surface is not hit by the structured light, it is useful to filter out the spurious predictions left in those areas after the confidence filtering. Before the thresholding, the image has to be converted to grayscale. In order to perform the said conversion, it is sufficient to take only the blue channel since, among the three, it is the only one that holds relevant information (as it can be seen in Figure 3.4(b)).

To summarize, the code for this last masking step can be simply structured as follows:

```
import cv2

def compute_mask(image_path):
    image_bgr = cv2.imread(image_path)
    image_gray = image_bgr[:,:,0]
    _, image_threholded = cv2.threshold(image_gray
        ,30,255,cv2.THRESH_BINARY)
    return image_thresholded
```

It is worth noting that the parameters used in the code snippet's thresholding reflect those effectively used during the filtering stage.

3.2.1 Implementation

Given a set of candidate confidence measures, it is necessary to find a way to aggregate them taking into account the fact that some of them are binary (such as LRC), while others return real values whose range varies considerably from measure to measure.

Relevant literature in this field [6] propose an effective way to automatically generate training labels from stereo pairs such that their distribution resemble as much as possible the ground truth data. The generation of such labels happen through a selection process carried out by using hand-crafted confidence measures, which allow to discriminate between correct and wrong disparity assignments. Relying on learned confidence measures is not possible as they would need ground truth labels in order to be trained, which is what we are trying to obtain, hence, a careful selection of the confidence measures is needed.
In literature, a well established method to determine the goodness of a confidence measure is the analysis of the ROC curve. The behaviour of the curve is able to describe certain characteristics of the confidence measures. For instance, a flat part of the curve imply that a subset of the disparity's pixels share the same confidence value. Ideally, all correct matches should be selected before all errors, resulting in the smallest possible AUC for a given disparity map [10]. An extensive analysis of different confidence measures [11] has demonstrated that different metrics, depending on the processed cues and adopted strategy, show differences on the evaluation of the assigned disparity values which are, at times, contradictory among one another. [6] proposes to use a set of confidence measures deemed as reliable by the available literature to automatically generate supervision data resembling as much as possible the actual ground truth labels.

This method assigns to a subset of pixels belonging to the disparity either a wrong label L_0 or a correct label L_1. The produced confidence estimate either comes in the form of binary values (like LRC) or are integer/real valued (like MSM). For each confidence it is possible to define two sets: C_0 and C_1. C_0 contains the wrong predictions while C_1 the correct ones. While assigning values of binary confidences to either one of the two sets is a straightforward operation, it is not the same for integer/real valued confidence measures since

they often work with widely varying ranges of values. The solution proposed to address this issue is to sort the pixels by ascending order of confidence's values and select a fraction of the first ones as belonging to C_0 and a fraction of the last ones as belonging to C_1.
The newly created C_0 and C_1 can be annotated as:

$$\begin{aligned} C_0 &= \{\boldsymbol{p} \in \boldsymbol{d} \mid 0 \leqslant C(\boldsymbol{p}) \leqslant \delta_0\} \\ C_1 &= \{\boldsymbol{p} \in \boldsymbol{d} \mid 1 - \delta_1 \leqslant C(\boldsymbol{p}) \leqslant 1\} \end{aligned} \tag{3.8}$$

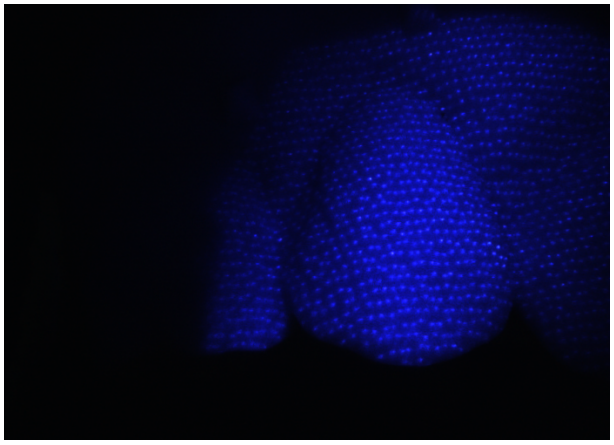
δ_0 and δ_1 represents the fraction of lowest scoring pixels and highest scoring pixels to add to C_0 and C_1 respectively.

Repeating this procedure for all of the selected confidence measures grouped in the pool $P = \{C', C'', ...\}$, would produce a set containing all of the confidence measures' wrong matches $P_0 = \{C'_0, C''_0, ...\}$, and one containing all of their correct matches $P_1 = \{C'_1, C''_1, ...\}$.
To determine the actual wrongness or correctness of a disparity pixel, it is necessary to aggregate both P_0 and P_1. The way it has been proposed to be done is to intersect all of the sets $C_0 \in P_0$ (which outputs the mistakenly estimated disparity values) and intersect all of the sets $C_1 \in P_1$ (in order to determine the correctly estimated disparity values):

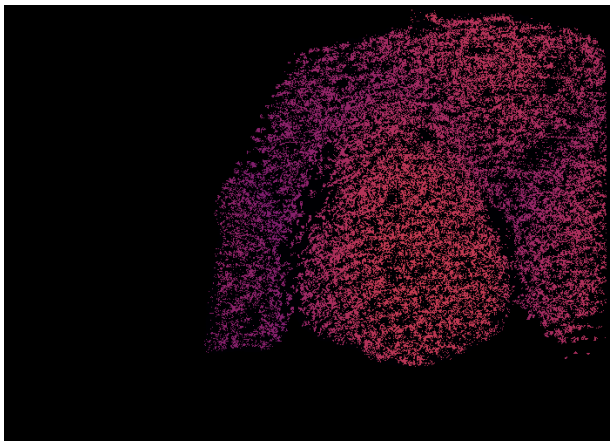
$$G_0 = \bigcap_{C_0^k \in P_0} C_0^k \,, \quad G_1 = \bigcap_{C_1^k \in P_1} C_1^k \tag{3.9}$$

In this way only those pixels classified as wrong/correct by all the confidence measures will be chosen.

A major contribution that this thesis brought to [6] is the re-implementation of its repository in CUDA so that it could be executed on GPU devices. Since the aim of using proxy-labeling is to select confident disparity values, the part regarding the selection of wrong disparity pixels has not been re-implemented. The greatest bottleneck of the application is the sorting of the pixels since an



(a) Input image



(b) Proxy-labels

Figure 3.9: In 3.9(b) is represented an example of proxy-labels used for supervision. In order to obtain this image, the disparity values have been normalized with the minimum and maximum disparity values. 3.9(a) is the input image that generated the labels in 3.9(b).

image of 500×696 would need 348 000 elements to be sorted. The GPU re-implementation of this part has been handled by using *Thrust* library's sorting implementation.

The issue about the usage of the sorting method with the SGM implementation that has been used for this thesis is that its produced cost volume is composed of integer values whose upper bound is $32 \times 8 = 256$ where 32 is given by the width of the features generated by the Center-Symmetric Census Transform and 8 is from the aggregation of the paths. Since the values' range is relatively small, it happens that many pixels have the same confidence

values, with the consequence that, as a result of the sorting, only some of the pixels having a certain value are discarded, while others having the same value as those discarded are retained, which is not ideal since, logically speaking, when considering confidence values beneath a threshold as not reliable all of those values should be discarded. Following these considerations, it has been deemed to be a more appropriate solution converting the real/integer valued confidence measures to binary confidences by introducing a threshold, whose value has been determined by manual tuning with the difficulty that all of the confidence measures present different ranges of values.

It is also worth to mention that the collection of the proxy-labels' dataset has been performed at half the resolution. This has been done because the maximum disparity range that the used SGM implementation is able to handle is 256 while, originally, our disparity range is of 500. Halving the image size also means halving the disparity range, this allow us to bring the effective disparity range to 250. Since the adopted SGM implementation works only with disparity ranges at 64, 128 and 256, it has been selected 256 as the final disparity range.

In Figure 3.9 can be observed an example taken from the proxy-labeling dataset collected with the aforementioned procedure.

Chapter 4

Deep Learning-Based Depth Estimation

Stereo depth estimation is a fundamental problem in computer vision where, given two horizontally aligned cameras (both physically and digitally through rectification), it is computed a disparity, which is the difference in the displacement of pixels between a reference camera view (usually the left one) and the view of the other aligned camera. For each pixel of the left image $I_L(x, y)$, its corresponding pixel in the right image is found at $I_R(x - \boldsymbol{d}_L(x, y), y)$.
Large pixel displacement means that the object is close, while small displacement imply that the observed object is far from the view. The scene's depth can be directly derived as the inverse of the disparity by means of $Z = \frac{fb}{\boldsymbol{d}}$, where f is the camera's focal length and b (baseline) is the distance between the two cameras' optical centers.

As described in section 2.1, stereo's standard pipeline can be divided in four main parts: matching cost computation, cost aggregation, disparity computation/optimization, disparity refinement.
Performing precise disparity estimation remains, though, a tough problem due to occlusions, large textureless areas, thin structures, repetitive textures, distortions given by the parallax effect, non-Lambertian and transparent surfaces.

In order to attempt to improve stereo matching's accuracy and solve the aforementioned issues, it has been proposed to integrate Deep Learning techniques in the stereo pipeline.

The first attempt to integrate Deep Learning in the stereo pipeline was done by replacing the matching cost computation in favor of a learning based module [29] composed of a siamese network (two architecturally identical networks that share the same weights) which extract features from the left and right frames, concatenates the produced feature vectors based on the target disparity range and then, through a classification head, solves a binary classification problem with the aim of estimating the correspondence of the concatenated pixel's features. The scores of the patches being a no-match is then used as the matching cost for the pair.

Successive contributions shifted towards end-to-end learning based approaches. The first of those approaches was *DispNetC* [18]. The extraction of the left and right images' features is handled by a siamese feature extractor. Thereafter, the features of the two images are merged with a correlation layer in order to form the cost volume. The correlation layer is a non-learned layer which perform a non-normalized cross-correlation between the feature's vectors. The output of the cross-correlation layer goes trough the decoder, which upscale it up to the input size and reduce its width to a single channel. The supervision is performed through a L1 loss applied at each upscaling stage of the decoder (to make it possible the ground truth disparity is downscaled accordingly).

GCNet [12], instead, attempted to replicate in a more faithful manner the pipeline proposed by SGM by explicitly building a cost volume through the concatenation of the features given by the siamese network, which has as result the creation of a 4D tensor of dimension $\frac{1}{2}D \times \frac{1}{2}H \times \frac{1}{2}W \times 2F$. The 4D cost volume is then used as input to a multi-scale 3D (vector-valued) convolution module whose aim is to emulate the aggregation step performed in standard stereo pipelines. From the final cost volume, in order to deduce sub-pixel

disparity values, a differentiable soft argmin, which computes the expected values over the cost curve distribution, is introduced:

$$\hat{\boldsymbol{d}}(x,y) = \sum_{d=0}^{D-1} d \cdot \text{softmax}(-C(x,y,d)) \tag{4.1}$$

4.1 RAFT-Stereo

4.1.1 Introduction

A problem closely related to disparity estimation is *optical flow*. The only difference between the two is that the corresponding pixel to seek does not necessarily have the same displacement relative to the y axis, thus, the search for the corresponding point must be carried out over a window (and not over the scanline as done in stereo matching).
Optical flow models have been developed by following a different line of research compared to disparity estimation models where, since *GCNet* [12], 3D convolution established the baseline for the task. Optical flow generally rely on iterative refinement in order to achieve its goals. An optical flow estimation model named RAFT [27] proposes to build a 4D cost volume by computing the correlation between all of the pixels' extracted features and use it as input to a GRU-based update module [1] to guide the iterative updates of the flow field.

From this thread of research a model named RAFT-Stereo [16] has been created. Differently from RAFT, the processed volume is 3D since only similarity between pixels belonging to the same scanline are computed. Additionally, it is introduced a multilevel GRU operator that keeps (and pass to the next iterative update) hidden states at multiple resolutions. The disparity update is, though, always done at full resolution. This integration is thought to help improving the global consistency of the disparity field by propagating information across the image.

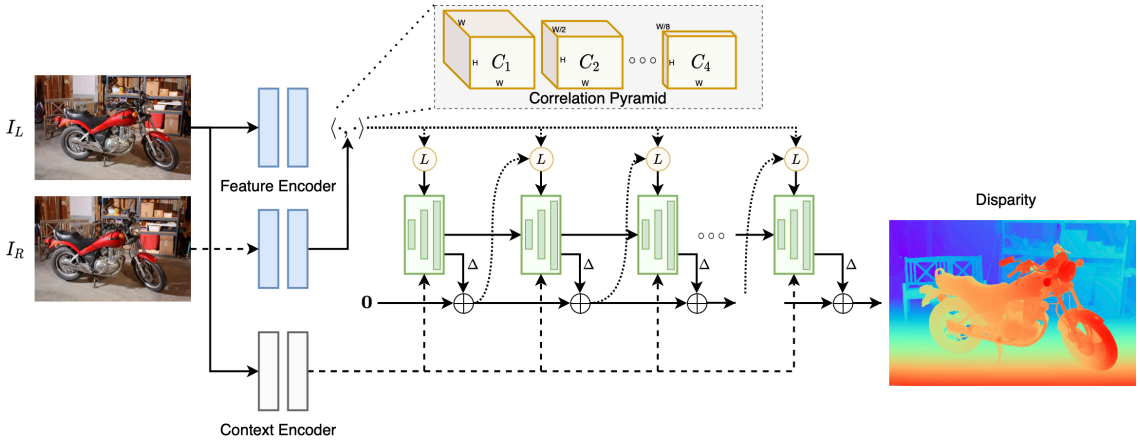


Figure 4.1: Visual representation of RAFT-Stereo.

With respect to the main thread of research in the field of disparity estimation, which made mainly use of 3D convolutions, RAFT-Stereo uses only 2D convolutions and a lightweight cost volume built by mean of a single matrix multiplication. Furthermore, this setup gives the possibility to trade accuracy for efficiency by changing the number of iterative GRU updates.

4.1.2 Approach

Given a pair of rectified images (I_L,I_R), RAFT-Stereo aims to estimate the disparity $\boldsymbol{d}$, which defines the horizontal displacement of the image I_L's pixels to the image I_R.

RAFT-Stereo is composed by a feature extractor, whose role is to extract the features from the left and right images. The computed features are then used to build a correlation pyramid of correlation volumes, which will be used by the GRU-based module in order to update the disparity field.

Feature Extraction

In order to produce the images' features two separate encoders are used: a *feature encoder* and a *context encoder*. The feature encoder is applied to both I_L and I_R and produces, for each image, a dense feature map which will be used to build the correlation volume. The feature encoder makes use of instance

normalization.
The context encoder is identical to the feature encoder, except that the context encoder uses batch normalization instead of instance normalization and that it is only applied to I_L. The output of the feature encoder is used to initialize the hidden state of the GRU-module and it is fed to it at each of its iterations.

Correlation Pyramid

In order to build the **correlation volume**, the visual similarity is computed as the dot product between feature vectors. Differently from RAFT, which builds the 4D correlation volume by correlating pixel's features for all pixels' pairs, the construction of the correlation volume is performed only by pairing the features of pixels laying on the same scanline.
Given $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{R}^{H \times W \times F}$, which are respectively the feature map of the left and right image, the 3D correlation volume can be computed as:

$$\boldsymbol{C}_{ijk} = \sum_{h} \boldsymbol{f}_{ijh} \cdot \boldsymbol{g}_{ikh}, \ \boldsymbol{C} \in \mathbb{R}^{H \times W \times W} \tag{4.2}$$

As done for the 4D correlation volume in RAFT, the 3D volume can be computed by means of a simple matrix multiplication.

The **correlation pyramid** is formed by stacking 4 correlation volumes built by repeated average pooling of the last dimension: the k^{th} correlation volume is constructed from the correlation volume at level k by applying a 1D average pooling that slides over the disparity dimension, with kernel size 2 and stride 2, producing a correlation volume of size $H \times W \times W/2^k$. Through pooling it is possible to increase the receptive field without having to diminish the spacial dimension, which allows to retrieve fine structures as well.

Since the disparity estimates done at each step of the GRU updates are at

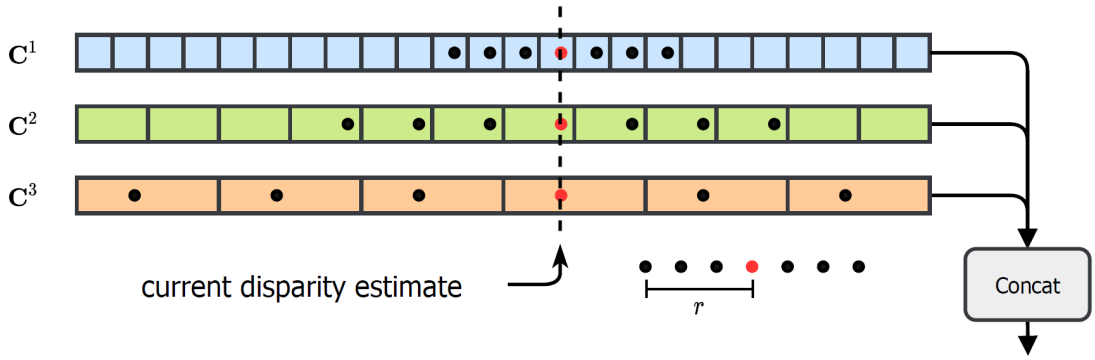


Figure 4.2: Visual representation of the lookup operator.

sub-pixel precision, an operator capable of performing the **lookup** of the correlation pyramid at non-integer indices is required. In order to do so RAFT-Stereo proposes to build a 1D grid with integer offsets around the current disparity estimate. The grid is used to index from each level in the correlation pyramid. In order to address the fact that grid values are non-integers, linear interpolation is used when indexing each volume. The retrieved values are then concatenated into a single feature map.

Multi-Level Update Operator

During the execution of RAFT-Stereo, the iterations performed by the GRU module produce N disparity fields $\{\boldsymbol{d}_1, ..., \boldsymbol{d}_N\}$ with the starting point being $\boldsymbol{d}_0 = \mathbf{0}$.

For each iteration, the disparity produced at the previous iterative step is used to index the correlation pyramid (through the lookup operator) producing a set of correlation features. Both the disparity produced at the previous step and the correlation features are passed through two (different) convolutional layers and then concatenated together with the context features and fed to the GRU module which will produce the new hidden state. The hidden state is then used to predict the disparity update.

Updating only at high resolution has the issue that the size of the receptive

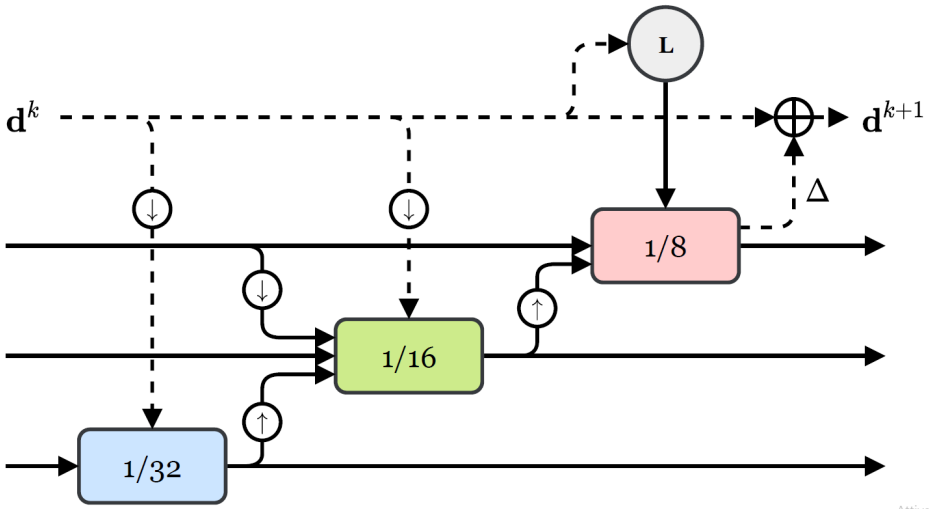


Figure 4.3: Visual representation of the GRU-based module.

field increase very slowly hence, for large untextured areas (which lack local information), many GRU iterations would be needed in order to aggregate enough context to produce a reliable disparity estimate. To address this issue, it has been proposed a **multi-resolution update operator** which operates on feature maps strided of $1/8$, $1/16$, $1/32$ with respect to the input resolution. The output of the computation performed over all of the strided feature maps is propagated to the next GRU's iteration through the outputted hidden state. Correlation lookups and disparity updates are handled by the GRU's module only at the highest resolution. Experiments have been performed also at $1/4$, $1/8$, $1/16$ strides.

In order to upsample the coarse disparity map at full resolution, it is used a convex upsampling method: the disparity values at full resolution are taken to be the convex combination of the 3×3 grid of their coarse resolution neighbours. The convex combination weights are predicted by the highest resolution GRU.

Method	KITTI-15	Middlebury full	Middlebury half	Middlebury quarter	ETH3D
HD3	26.5	50.3	37.9	20.3	54.2
gwcnet	22.7	47.1	34.2	18.1	30.1
PSMNet	16.3	39.5	25.1	14.2	23.8
GANet	11.7	32.2	20.3	11.2	14.1
DSMNet	6.5	21.8	13.8	**8.1**	6.2
RAFT-Stereo	**5.74**	**18.33**	**12.59**	9.36	**3.28**

Table 4.1: (from [16]) synthetic to real generalization experiments. All methods were trained on SceneFlow[23] and tested on the KITTI-2015, Middlebury, and ETH3D validation datasets. Errors are the percent of pixels with end-point-error greater than the specified threshold. For each dataset the standard evaluation thresholds are used: 3px for KITTI, 2px for Middlebury, 1px for ETH3D.

Supervision

RAFT-Stereo is supervised using a l_1 distance between the ground truth disparity $\boldsymbol{d}_{GT}$ and all the disparity fields produced at each iterative step with exponentially increasing weights.

$$\mathcal{L} = \sum_{i=1}^{N} \gamma^{N-i} \|\boldsymbol{d}_{GT} - \boldsymbol{d}_i\|_1, \quad \text{where } \gamma = 0.9. \tag{4.3}$$

4.1.3 Zero-Shot Generalization

In the original paper, the ability of RAFT-Stereo to generalize over unseen real data being trained only on synthetic data, i.e. zero-shot generalization, is also evaluated . This is a critical ability for a model since, currently, there are no large real-world datasets in existence that can be used for training. In this instance, RAFT-Stereo has been trained on Scene Flow [17] and evaluated over KITTI-15 [19], Middlebury [22] and ETH3D [25]. The results and comparisons with other models trained in the same scenario are reported in Table 4.1.

Across all three validation datasets, RAFT-Stereo exhibits state-of-the-art performance in the zero-shot synthetic-to-real setting. RAFT-Stereo is trained for 200k iterations using data augmentation.

Chapter 5

Evaluation

The evaluation of the data gathering techniques used during the scope of this thesis begins with the selection and training of a model fit for the kind of context in which we are operating.
The evaluation of the trained model will be carried out by performing two different kind of comparisons. The first one will be done at disparity level by comparing the results produced by the model and the disparity ground truth. The second one will be a 3D comparison between the mesh generated through a SLAM pipeline using as input a sequence of disparities and a 3D ground truth gathered using an high precision scanner. Since the training with the synthetic dataset did not bring good enough results to enable a 3D reconstruction through the SLAM pipeline, its 3D metric will not be reported.

5.1 Model Selection

The model chosen for this task is RAFT-Stereo [16]. The reason behind this choice is that, other than being one of the top performing models in the disparity estimation task for various benchmarks, it is also able to handle a very large disparity range, which is a necessary property in our settings since the adopted disparity range is 500 (with a minimum of 200 and a maximum of 700). Many of the well established models are not able to handle such an high

range's value. Moreover, its generalization capabilities encourage us to see whether it is able to handle the domain shift between the synthetic domain used to train the model and the target domain we wish to test the model on. However, it should be noted that it is difficult to expect good generalization capabilities in a zero-shot setting on the target domain since it is composed of non-natural images (this is given by the fact that the lighting is not natural since it comes entirely from the structured light projector and the scene is not retro-illuminated by natural light).

5.1.1 Training

Proxy-Labels Dataset

The training has been performed on a NVIDIA RTX 3800 Laptop (16GB VRAM) using RAFT-Stereo pre-trained over Scene Flow by training all the model's layers on the proxy dataset's training set (none of the model's layers have been frozen) for 80000 steps with a batch size of 4.

The optimizer used is AdamW, and it has been chosen to schedule the learning rate through a 1cycle learning rate strategy, according to which the learning rate increases linearly for the first 800 steps (1% of the steps) from an initial learning rate of $0.0002/25 = 0.000008$ to a learning rate of 0.0002 and then decreases linearly in the successive steps.

An aggressive vertical crop has been performed (from an initial height of 500 px to 200 px) while the horizontal dimension has been kept as it is (696 px). This choice is mainly motivated by the fact that there was a need to reduce the load on the GPU's VRAM given by the activations. The asymmetry of the crop is given by the fact that an horizontal crop might have determined the removal of valid matching points since they lay on the same row. The vertical crop does not, instead, suffer from this issue and it might actually bring a regularization effect by limiting the receptive field (and thus the context) that the network can use to predict the match.

The only data augmentation used (other than the crop) is a color saturation-kind of augmentation (which is also the configuration used originally to train RAFT-Stereo).

Synthetic Dataset

The training performed on the synthetic dataset retains most of the considerations done for the proxy-labels' one, with the difference that it has been trained for 50000 steps and that the cropping adopted reduce the input images to a size of 200 px in height and 1000 px in width.

5.1.2 Inference

Proxy-Labels Dataset

In the inference phase the input images needs to be downsampled by a factor of 2, and the produced disparities are to be upsampled by a factor of 2 to bring them back to the original size. This is done because the model has been trained at with half resolution images. Before upsampling the disparity, it is necessary to multiply its values by 2. The upsampling is done through a bicubic interpolation. The upsampled disparities are filtered by a mask computed by thresholding the left input image. This is done because part of the input images contain black areas (see Figure 3.4(b) for an example) which are uninformative and, consequently, produce unreliable disparity estimates. The code used to perform this operation is the same as the one reported in section 3.2.

Synthetic Dataset

Retains most of the considerations done for proxy-labeling with the difference that, in this case, there is no need for upsampling since the model has been trained on full resolution images.

Checkpoint	EPE	1px	3px	5px
5000	0.3706	0.9581	0.9983	0.9991
10000	0.3740	0.9555	0.9985	0.9992
15000	0.3921	0.9468	0.9982	0.9991
20000	0.3979	0.9422	0.9983	0.9992
25000	0.4478	0.9224	0.9969	0.9985

Table 5.1: RAFT-Stereo evaluation on proxy-labels' validation set.

Checkpoint	EPE	1px	3px	5px
10000	9.3291	0.7223	0.7625	0.779
20000	1.2998	0.8008	0.899	0.9383
30000	1.4987	0.7663	0.8597	0.9159
40000	1.662	0.7844	0.8906	0.9324
50000	1.0941	0.7754	0.8936	0.9542

Table 5.2: RAFT-Stereo evaluation on synthetic dataset's validation set.

5.2 Disparity Metrics

The disparity metrics adopted are the ones classically used for this task:

- Average end-point error (**EPE**): this measure was originally introduced for the task of optical flow [20] as the mean of absolute error among all pixels:

$$\begin{aligned} \text{EPE} &= \frac{1}{WH}\|\boldsymbol{d} - \boldsymbol{d}_{GT}\|_1 \\ &= \frac{1}{WH}\sum_{x=0}^{W}\sum_{y=0}^{H}|\boldsymbol{d}(x,y) - \boldsymbol{d}_{GT}(x,y)| \end{aligned} \tag{5.1}$$

  RAFT-Stereo reinterpret EPE in its euclidean form and implements it in the following way:

$$\begin{aligned} \text{EPE} &= \frac{1}{WH}\|\boldsymbol{d} - \boldsymbol{d}_{GT}\|_2 \\ &= \frac{1}{WH}\sqrt{\sum_{x=0}^{W}\sum_{y=0}^{H}\left(\boldsymbol{d}(x,y) - \boldsymbol{d}_{GT}(x,y)\right)^2} \end{aligned} \tag{5.2}$$

- **1px**: percentage of pixels that have EPE < 1.
- **3px**: percentage of pixels that have EPE < 3.
- **5px**: percentage of pixels that have EPE < 5.

The results of the evaluation metrics on the proxy-labels' validation set are reported in Table 5.1. The best model selected is the checkpoint 5000 as it reports the lowest EPE and highest 1px.

The results obtained on the synthetic dataset's validation set are reported in Table 5.2.

Proxy-Labeling Training Results

The metrics chosen to evaluate the training are EPE, 1px, 3px, 5px. Below can be found the plots representing their behaviour during training, together with the loss' (in Figure 5.1, 5.2, 5.3, 5.4 and 5.5).

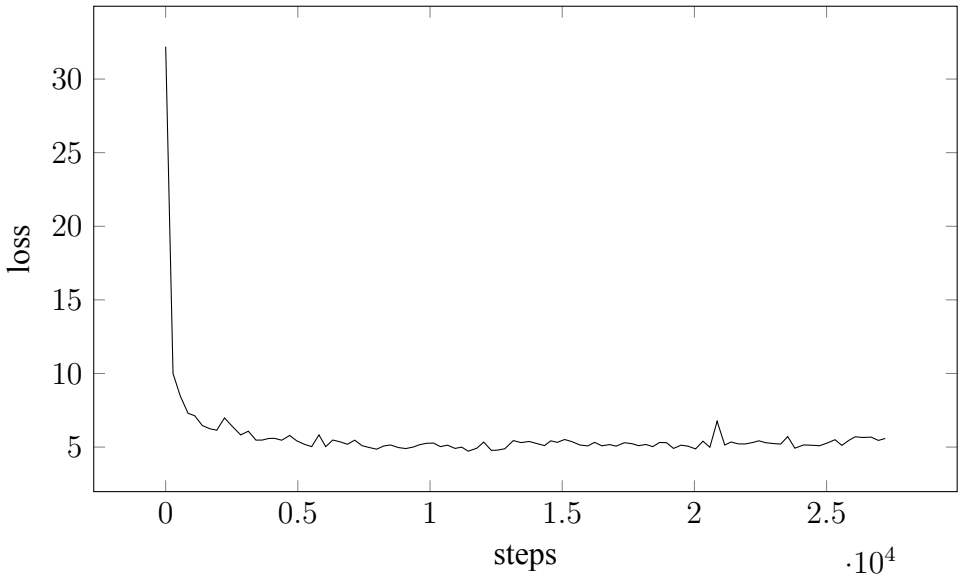


Figure 5.1: RAFT-Stereo training **loss**. The original plot curve has been smoothed with a 1D Gaussian kernel with $\sigma = 3$ and downsampled from 10000 to 100 points.

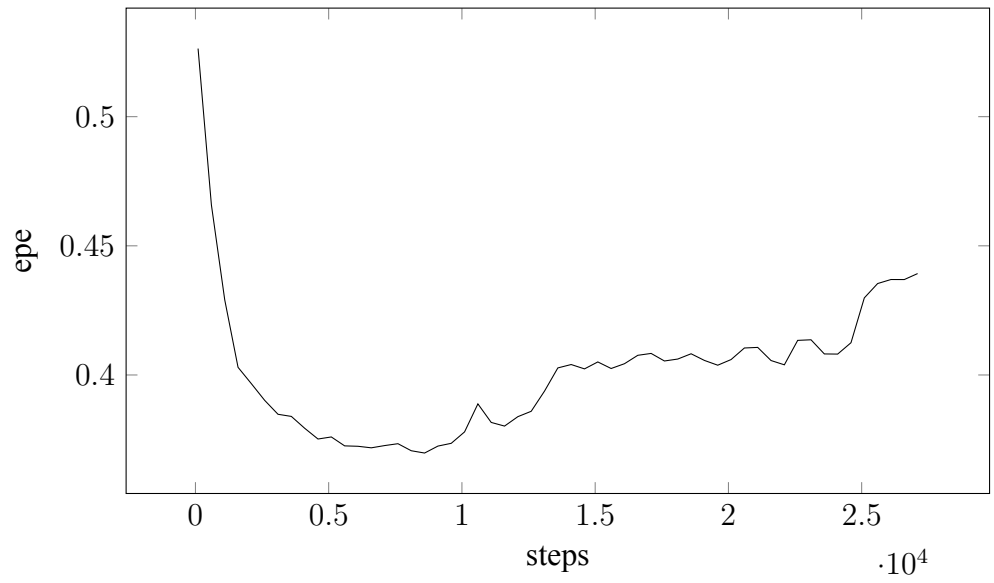


Figure 5.2: RAFT-Stereo training **EPE**. The original plot curve has been smoothed with a 1D Gaussian kernel with $\sigma = 3$ and downsampled from 275 to 55 points.

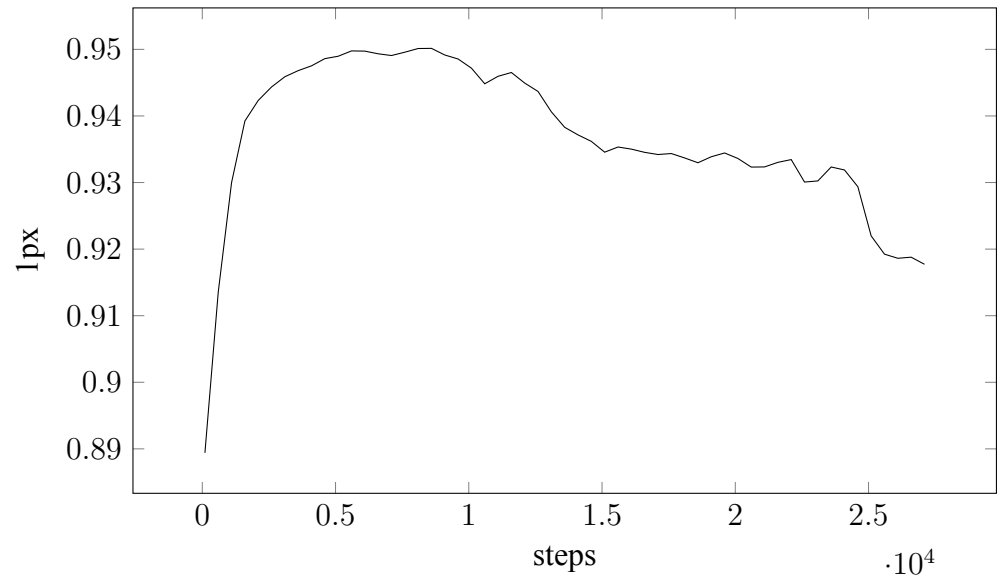


Figure 5.3: RAFT-Stereo training **1px**. The original plot curve has been smoothed with a 1D Gaussian kernel with $\sigma = 3$ and downsampled from 275 to 55 points.

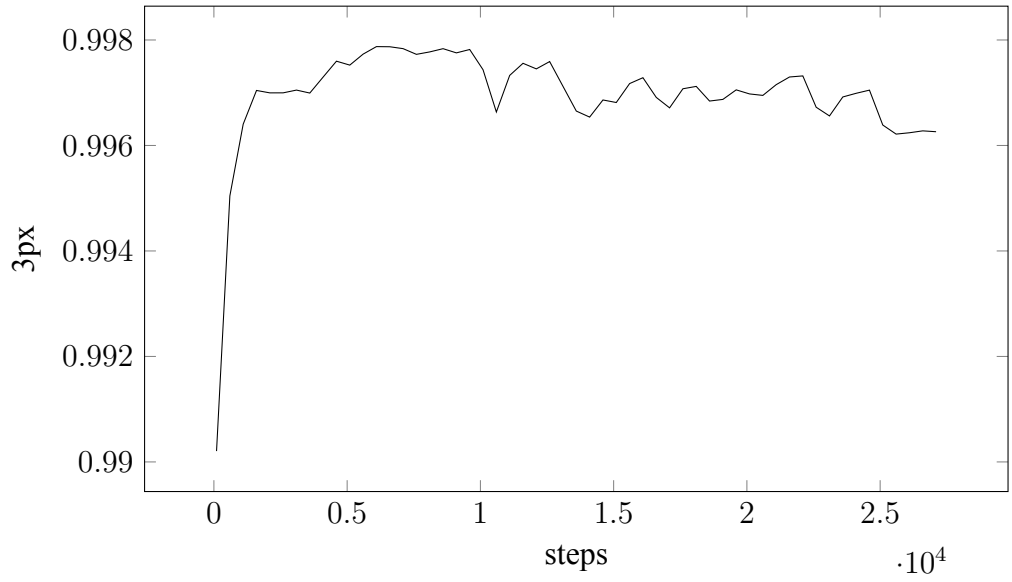


Figure 5.4: RAFT-Stereo training **3px**. The original plot curve has been smoothed with a 1D Gaussian kernel with $\sigma = 3$ and downsampled from 275 to 55 points.

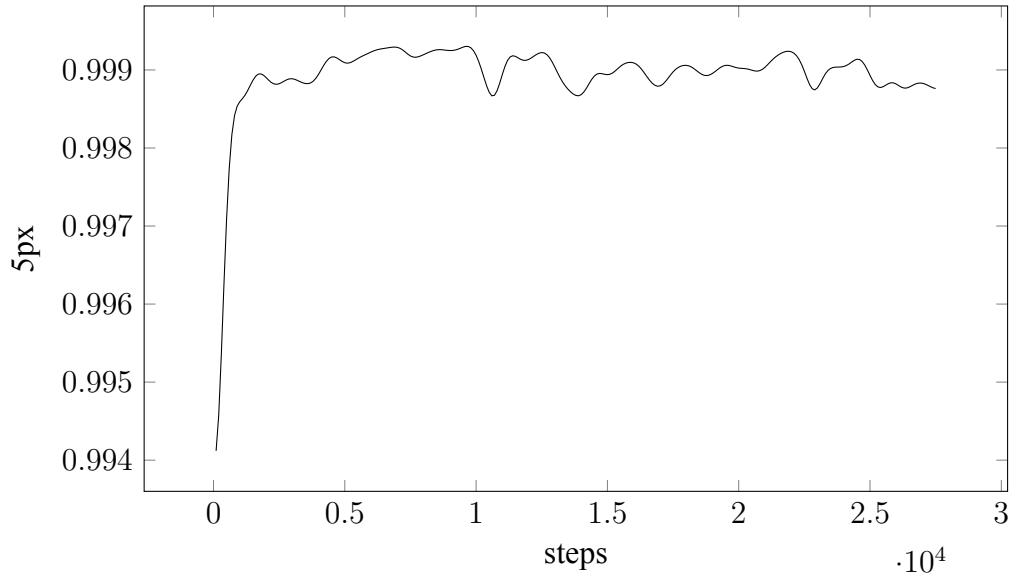


Figure 5.5: RAFT-Stereo training **5px**. The original plot curve has been smoothed with a 1D Gaussian kernel with $\sigma = 3$ and downsampled from 275 to 55 points.

5.3 3D Metrics

As also stated at the beginning of this chapter, the 3D evaluation is the results of the comparison between the mesh produced by a SLAM pipeline, in which RAFT-Stereo is the depth-sensing component, and a ground truth gathered with an high precision scanner.

The scanner adopted to gather the ground truth is a benchtop scanner that relies on a digital light stripe projection scanning technique and is able to scan targets whose materials are not strongly absorbent, reflecting or transparent, with a precision that ranges from $4.9\ \mu m$ to $0.9\ \mu m$.

The software used in order to compute the 3D metrics is called CloudCompare [2]. Through this software all the preliminary operations needed to be able to compare the meshes are performed.
The first step would be to scale the entities that needs to be compared, but it can be skipped since the ground truth mesh and the produced one have both already the same scale. The successive step is to roughly register the meshes: this can be done with the interactive transformation tool or with the tool that allows to align two meshes by picking four (roughly) equivalent points on both of them. The third step is to finely register the roughly aligned entities through iterative closest point (ICP). While these steps would be normally enough to compute the final metric, in practice an additional segmentation is carried out in order to select the specific area of the produced mesh that we wish to compute the metrics on. The last step is to once again finely register the two entities.
The comparisons are normally performed over a single tooth or just a part of it. One of the reasons behind this choice is that, since the captured area is small, the contribution to the total error introduced by the pose estimation algorithm is minimized.

Once the preliminary operations are completed, it is possible to compute the metrics by comparing the two meshes. In CloudCompare, the distance

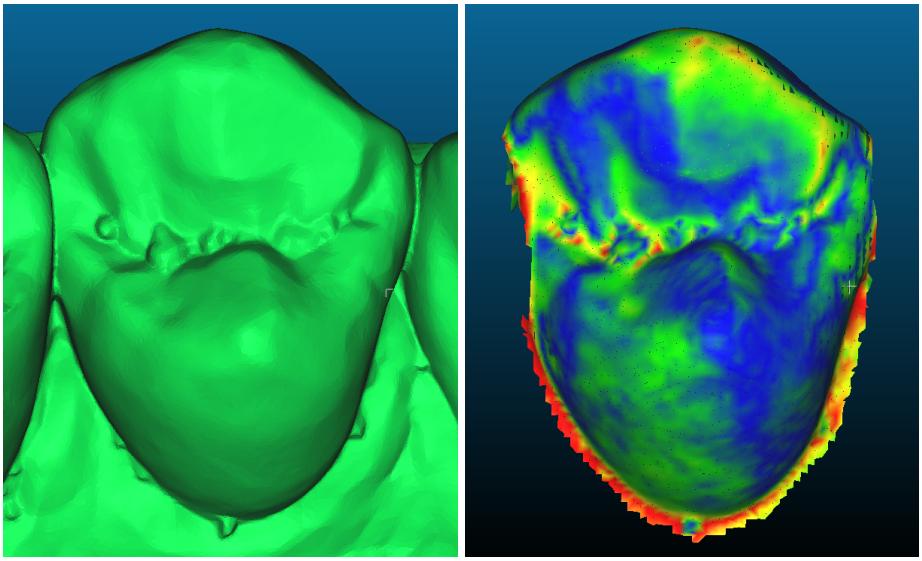
Index	RAFT-Stereo		SGM	
	Mean distance	Std deviation	Mean distance	Std deviation
1	24.9799	22.8143	26.7032	23.8567
2	15.5599	15.0312	16.573	15.4524
3	17.188	13.0985	18.7317	12.628

Table 5.3: Results of the 3D metrics. The reported values are in micrometers (μm). The smaller the mean distance, the better the result.

between a point cloud and a mesh is computed through the *Cloud-to-Mesh Distance* tool, which computes the euclidean distance between a point of the compared cloud with the nearest triangle of the reference mesh. Even though CloudCompare does not offer a mesh-to-mesh comparison tool, is still possible to perform such comparisons though the Cloud-to-Mesh Distance tool, which compute the euclidean distance between the compared mesh' vertices and the closest reference mesh' triangles. The final result returned is the average of the distances computed for each closest vertex-triangle pair (the standard deviation is computed as well).

Through the aforementioned procedure, a total of three measurements have been done over meshes produced from different image sequences. The results of the measurements can be seen in Table 5.3, which reports the mean distance between the compared mesh' vertices and the ground truth mesh' closest triangles and the standard deviation. From the table it can be observed how the results given by RAFT-Stereo bring a general improvement with respect to the SGM baseline over the considered 3D metrics. The RAFT-Stereo's checkpoint used to compute the 3D metric is the 5000, as it was reported to have the best results on the disparity metrics.

An issue with using this kind of measurement to evaluate the quality of disparities is that the part of the SLAM pipeline that comes after the disparity estimation has a significant influence on the result. The update of the scene



(a) Ground truth mesh (b) Predicted mesh with error colormap

Figure 5.6: The two figures refers to the evaluation indicated with index 1 in the Table 5.3. 5.6(a) refers to the 3D ground truth while 5.6(b) is the generated mesh (segmented as described in this Section). The colors of the generated mesh indicate the magnitude of the error (blue for small errors and red for high errors).

plays an important role to produce a good quality mesh: during the recording of a scene it might happen to view a certain part more than once, thus more disparities capturing its geometry will be produced. Therefore, results on the 3D metric are heavily influenced also by the ability of the SLAM algorithm to discern which of the different disparity maps, capturing same parts of the scene, are to be used in the final 3D reconstruction.

In Figure 5.6 are presented the results related to the evaluation indicated in the Table 5.3 with index 1. It is possible to see from the error colormap of the generated mesh how the majority of the errors are gathered around small structures on top of the tooth and at the border between the tooth and the gums. There are multiple reasons for this behaviour: one is that thin structures are inherently very hard areas to correctly estimate disparity for. Another reason concern the structure of the imaging system: being that the baseline is quite large (at least in proportion to the working range), a lot of occlusions are

present which makes difficult to see some areas. Lastly, since the coverage given by the structured light's blue dots is sparse, only some parts of the thin structures are illuminated rendering impossible to confidently estimate their shape in one go. This also means that the update of the scene (mentioned in the precedent paragraph) needs to be very well calibrated in order to integrate predictions from different frames to enable a precise 3D reconstruction of a given thin structure by assuring a complete light coverage of its whole geometry.

Chapter 6

Conclusions

6.1 Final Remarks

In this thesis, it has been explored how deep learning based depth estimation can be integrated in a particular instance of the biomedical domain, which is characterized by scarcity of visual features and limitations about the gathering of ground truth depth data through sensors, which is the most adopted approach to deploy Supervised Deep Learning algorithms for this kind of task.

Two techniques have been attempted in order to make up for the absence of ground truth data:

- **Synthetic data generation**: a synthetic dataset, whose domain slightly differ from the target domain, has been created in order to supervise the training of a Deep Learning model in the attempt of exploiting its generalization capabilities to perform well even on a domain different with respect to the one it has been trained on. The results of the prediction of temporally adjacent images (belonging to the target domain) showed high inconsistency and using the produced disparities in the SLAM pipeline did not allow the 3D reconstruction to be executed.

- **Proxy-labeling**: in order to supervise the model it has been used the output of the SGM baseline filtered by confidence measures with the

aim of discarding unreliable disparity estimates while keeping the most confident ones.

Proxy-labeling obtained promising results both in terms of temporal consistency of the prediction (which suggests a certain robustness of the features produced by the model) and also in terms of results obtained on the adopted 3D metric, which, for this application, is the main reference metric even though it presents the limitations described in section 5.3.

The selection of the model for this project fell upon RAFT-Stereo [16]. The main reason behind the selection of this model is its ability to handle very large disparity ranges, which is a necessary characteristic for this application since the target disparity range is very large (more than what a Deep Learning-based model usually is able to handle). Other characteristics that determined the choice of this model is the fact that RAFT-Stereo is one of the top scoring disparity estimation models on the Middlebury benchmark and that it also offers a real-time implementation (which is necessary in an online SLAM pipeline).

The evaluation results given by the RAFT-Stereo model trained on the proxy-label dataset highlighted the good capacity of the trained model to lead to a more accurate 3D reconstruction with respect to the baseline it has been trained on.

6.2 Future Developments

We can divide the work that can be done to improve the current application in three main threads: one that aims to build a better deep-learning based model, a second one that attempts to find different ways to gather ground truth data and the last one that consists in an alteration of the current imaging system.

(a) Input image (b) RAFT-Stereo's output

Figure 6.1: This example (taken from [15]) shows how texture level information that do not stem from matching can be detrimental for disparity estimation.

In this section, some ideas will be given about how to approach the aforementioned areas of improvement.

6.2.1 Models

Throughout the development of this thesis a new model that beat the state of the art, which until that moment was represented by RAFT-Stereo, was presented at CVPR 2022 under the name CREStereo [15].

CREStereo, other than presenting higher performances on the well known benchmarks with respect to RAFT-Stereo, retains the ability to work with high disparity ranges (characteristic necessary for this application) while presenting interesting features and overcoming some limitations present in RAFT-Stereo. One of the interesting features introduced by CREStereo is named *Adaptive Group Correlation Layer* which, through a sub-module called *2D-1D Alternate Local Search*, extends the 1D correlation computation to a 2D computation in an alternate manner, in this way it can be performed matching outside of the traditional scanline mitigating the influence of imperfect rectification.

Another contribution that CREStereo might bring to the application at hand with respect to RAFT-Stereo is that RAFT-Stereo uses as information

to compute the disparity update in its recurrent GRU module the features extracted from the left image through its *context encoder*. While this might effectively contribute to an higher performance since it might give information about the boundary of the object (or just because it enlarges the capacity of the model), it can be argued that this contribution might be detrimental for the kind of lighting adopted since, while it is effectively able to enhance the matching capability of the system in a textureless context, it does not give much visual information about the shape/boundaries of the object, bringing the risk of introducing artifacts in the output disparities given by the shape of the structured light's dots. Moreover, as also shown by some examples in CREStereo's paper (see Figure 6.1), it appears that injecting image's features other than the matching ones might compromise the ability of the model to generalize as the consequence of an excessive extrapolation of texture information. CREStereo does not make use of this kind of information, thus, it is possible to expect a better generalization capabilities and not to generate artifacts given by the non-natural nature of the input images.

Another addition that might prove to be useful for the task of 3D reconstruction is the introduction inside the model of a confidence estimation module capable of filtering out those disparity values that are deemed as imprecise or that stem from an ambiguous matching. Since, in 3D reconstruction, precision should be preferred over density, this module might provide an effective aid to the update rules of the 3D scene discussed in chapter 5.

6.2.2 Dataset

While proxy-labeling has shown interesting results, other techniques that do not involve the usage of sensor (for the limitations explained during the course of this thesis) and that are capable to deliver more precise ground truth data should be considered as well.

One of such alternative techniques involves the usage of the 3D ground truth in order to gather the disparity ground truth. In particular, by exploiting the SLAM pipeline and the current SGM baseline as its depth sensing component, it is possible to obtain the predicted output mesh (which is the one used for comparison with the 3D ground truth in section 5.3) together with the poses (location and rotation) where the camera passed through in order to capture the images used to generate the mesh.
By registering the predicted mesh with the ground truth mesh (with a process similar to the one described in section 5.3), the camera poses obtained from the SLAM pipeline can be used on the ground truth mesh as well in order to obtain the depth information for each image of the sequence that generated the predicted output mesh, which can then be converted in dense sub-pixel disparities through the camera parameters. The disparities extracted through this method can be used as ground truth to supervised a Deep Learning-based model.

A major limitation that can be encountered with this data gathering technique is the error introduced by the pose estimation algorithm. In order to mitigate this issue it is important to capture short sequences of images in order to not accumulate too much error.

6.2.3 Imaging System

While the presence of the structured light is rendered necessary by the absence of textures of the target, the total absence of natural light gives to the input images a non-natural appearance. From one side this might hinder a correct prediction of the disparity by the used model as there are no extensive studies on the performance of these systems on non-natural imaging scenarios and in case of RAFT-Stereo, which uses the image context information to aid disparity estimation, it might lead to the generation of artifacts. Moreover, as also discussed at the end of chapter 5, the fact that the distribution of the

structured light's dots is sparse, means that the full coverage of thin structures is not guaranteed, leading to sub-optimal disparity estimates where the estimation of structures not covered by the dots might be given by the propagation of neighbouring disparity estimates rather than being the result of a direct matching.

In order to solve the aforementioned issues it might be useful to alter the imaging system in a way that, other than the structured light, also some natural light is cast upon the scene with an intensity that would not completely suppress the contribution given by the structured light. A bit of natural light would contribute to delineate in a more detailed way the shape of thin structures and the object boundaries.

Bibliography

[1] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[2] Cloudcompare, version 2.12.4. URL: https://www.cloudcompare.org/. License: GPL.

[3] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam. URL: http://www.blender.org.

[4] G. Egnal and R. Wildes. Detecting binocular half-occlusions: empirical comparisons of five approaches. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1127–1133, 2002. DOI: 10.1109/TPAMI.2002.1023808.

[5] G. Egnal, M. Mintz, and R. P. Wildes. A stereo confidence metric using single view imagery with comparison to five alternative approaches. *Image and vision computing*, 22(12):943–957, 2004.

[6] F. T. et al. Learning confidence measures in the wild. In G. B. Tae-Kyun Kim Stefanos Zafeiriou and K. Mikolajczyk, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 133.1–133.13. BMVA Press, September 2017. ISBN: 1-901725-60-X. DOI: 10.5244/C.31.133. URL: https://dx.doi.org/10.5244/C.31.133.

[7] H. Hirschmuller. 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.

[8] H. Hirschmuller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1582–1599, 2009. DOI: 10.1109/TPAMI.2008.221.

[9] H. Hirschmüller. Semi-global matching-motivation, developments and applications. *Photogrammetric Week 11*:173–184, 2011.

[10] X. Hu and P. Mordohai. A quantitative evaluation of confidence measures for stereo vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2121–2133, 2012. DOI: 10.1109/TPAMI.2012.46.

[11] X. Hu and P. Mordohai. A quantitative evaluation of confidence measures for stereo vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2121–2133, 2012. DOI: 10.1109/TPAMI.2012.46.

[12] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry. End-to-end learning of geometry and context for deep stereo regression. *CoRR*, abs/1703.04309, 2017. arXiv: 1703.04309. URL: http://arxiv.org/abs/1703.04309.

[13] J. Kim et al. Visual correspondence using energy minimization and mutual information. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1033–1040. IEEE, 2003.

[14] S. Kim, D.-g. Yoo, and Y. H. Kim. Stereo confidence metrics using the costs of surrounding pixels. In *2014 19th International Conference on Digital Signal Processing*, pages 98–103, 2014. DOI: 10.1109/ICDSP.2014.6900808.

[15] J. Li, P. Wang, P. Xiong, T. Cai, Z. Yan, L. Yang, J. Liu, H. Fan, and S. Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation, 2022. DOI: 10.48550/ARXIV.2203.11483. URL: https://arxiv.org/abs/2203.11483.

[16] L. Lipson, Z. Teed, and J. Deng. Raft-stereo: multilevel recurrent field transforms for stereo matching. *CoRR*, abs/2109.07547, 2021. arXiv: 2109.07547. URL: https://arxiv.org/abs/2109.07547.

[17] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL: http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16. arXiv:1512.02134.

[18] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *CoRR*, abs/1512.02134, 2015. arXiv: 1512.02134. URL: http://arxiv.org/abs/1512.02134.

[19] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[20] M. Otte and H.-H. Nagel. Optical flow estimation: advances and comparisons. In *European conference on computer vision*, pages 49–60. Springer, 1994.

[21] M. Poggi, S. Kim, F. Tosi, S. Kim, F. Aleotti, D. Min, K. Sohn, and S. Mattoccia. On the confidence of stereo matching in a deep-learning era: a quantitative evaluation. *CoRR*, abs/2101.00431, 2021. arXiv: 2101.00431. URL: https://arxiv.org/abs/2101.00431.

[22] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *German conference on pattern recognition*, pages 31–42. Springer, 2014.

[23] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1):7–42, 2002.

[24] D. Scharstein and R. Szeliski. Stereo matching with nonlinear diffusion. *International journal of computer vision*, 28(2):155–174, 1998.

[25] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[26] R. Spangenberg, T. Langner, and R. Rojas. Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In *International Conference on Computer Analysis of Images and Patterns*, pages 34–41. Springer, 2013.

[27] Z. Teed and J. Deng. RAFT: recurrent all-pairs field transforms for optical flow. *CoRR*, abs/2003.12039, 2020. arXiv: 2003.12039. URL: https://arxiv.org/abs/2003.12039.

[28] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer, 1994.

[29] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. *CoRR*, abs/1409.4326, 2014. arXiv: 1409.4326. URL: http://arxiv.org/abs/1409.4326.

Acknowledgements

I would like to thank professor Samuele Salti and Dr. Matteo Poggi for assisting me through the many technical difficulties. I am grateful to my mother and girlfriend for all the moral support they gave me.
Last but not least, I would like to thank BlueThink for giving me the opportunity to develop this thesis.