Journal of Real-Time Image Processing (2021) 18:539–559
https://doi.org/10.1007/s11554-020-00993-w

ORIGINAL RESEARCH PAPER

The genetic algorithm census transform: evaluation of census windows of different size and level of sparseness through hardware in-the-loop training

Carl Ahlberg[1] · Miguel León[1] · Fredrik Ekstrand[1] · Mikael Ekström[1]

Received: 25 December 2019 / Accepted: 18 June 2020 / Published online: 6 July 2020

Abstract
Stereo correspondence is a well-established research topic and has spawned categories of algorithms combining several processing steps and strategies. One core part to stereo correspondence is to determine matching cost between the two images, or patches from the two images. Over the years several different cost metrics have been proposed, one being the Census Transform (CT). The CT is well proven for its robust matching, especially along object boundaries, with respect to outliers and radiometric differences. The CT also comes at a low computational cost and is suitable for hardware implementation. Two key developments to the CT are non-centric and sparse comparison schemas, to increase matching performance and/or save computational resources. Recent CT algorithms share both traits but are handcrafted, bounded with respect to symmetry, edge lengths and defined for a specific window size. To overcome this, a Genetic Algorithm (GA) was applied to the CT, proposing the Genetic Algorithm Census Transform (GACT), to automatically derive comparison schemas from example data. In this paper, FPGA-based hardware acceleration of GACT, has enabled evaluation of census windows of different size and shape, by significantly reducing processing time associated with training. The experiments show that lateral GACT windows produce better matching accuracy and require less resources when compared to square windows.

Keywords Census transform · Stereo correspondence · Matching cost metric · Genetic algorithm · Real time · FPGA · SoC · VHDL

1 Introduction

With an ever-growing interest in intelligent and autonomous systems follows demands on perception, i.e. how to gather and extract meaningful information, from different sensor modalities, within a specified time frame. For us humans, vision is the most central sense, and for autonomous agents, acting in the real world, image sensors are key, as they are low cost and versatile. Through computer vision, application-relevant information can be extracted from images, such as color, features, objects, size, and depth/distance. To extract 3D information from images is either dependent on a priori information or requires multiple images, from different viewpoints. This can be completed by moving a single camera, but in an unknown and constantly changing environment, a stereo camera, with two horizontally displaced and synchronized cameras, are much preferable as, image displacement is known (static) and there is no interference between ego-motion and object motion. This is referred to binocular, or two-frame, stereo. However, to find depth information the correspondence problem must be solved, that is to establish correspondence between pixels, or regions, from one image to the other. Assuming rectified stereo images, displacement between corresponding pixels, or disparity, will be limited to the horizontal axis. The larger the disparity, the closer the object. Knowing the disparity, d, and the stereo camera parameters, depth, z, is given by the following equation:

✉ Carl Ahlberg
carl.ahlberg@mdh.se

Miguel León
miguel.leonortiz@mdh.se

Fredrik Ekstrand
fredrik.ekstrand@mdh.se

Mikael Ekström
mikael.ekstrom@mdh.se

[1] School of Innovation, Design and Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden

$$z = \frac{fB}{d} \tag{1}$$

where f is the focal length and B is the length of the baseline, i.e., horizontal displacement between the cameras. Hence, solving the correspondence problems, also known as stereo matching, is the core of stereo vision. The complexity of the problem and the numerous applications in different areas, such as autonomous driving, robotics, industrial automation, augmented reality, 3D mapping, entertainment, etc., have firmly established the stereo vision field of research.

Four steps, identified in the taxonomy of (two-frame) stereo algorithms [39], are frequently adopted: (1) matching cost computation; (2) cost aggregation; (3) disparity computation; and (4) disparity refinement. Furthermore, methods are classified as local or global (or hybrid SGM). Global algorithms aim to minimize a total cost function over the full image while, for local methods, disparity, for a given pixel, is decided from a surrounding (and hence local) neighborhood. In general, global methods can produce higher matching accuracy at a higher level of computing complexity and are hence suitable for applications such as creation of 3D-maps or 3D-models of static scenes and objects. Applications that require high frame rates and/or low latency adopt local algorithms.

There are many approaches to matching cost calculation, i.e. determining how similar pixels are between frames, and the closer the correspondence the lower the cost. For local algorithms the cost computation and cost aggregation are closely related where, instead of relying on costs for single pixels, cost support is considered over image patches, reducing the level of ambiguity, analogous to the aperture problem. Cost are divided into parametric; Absolute Difference (AD), Sum of AD (SAD), Zero mean SAD (ZSAD), Normalized Cross-Correlation (NCC), Zero mean NCC (ZNCC), etc., non-parametric; the Census Transform (CT) and rank transform [45] and, Mutual Information [16]. Or a combination, such as AD-Census [29] or truncated absolute difference of colors and gradients [18].

With the introduction of larger datasets, methods adopting deep learning, such as Convolutional Neural Networks (CNNs), as a cost metric [46, 47] and later end-to-end networks [5, 28], now dominate the research field and top the KITTI[1] [30] and Middlebury[2] [38] rankings [41, 48]. Although providing high matching accuracy, CNNs come with the drawback of high complexity, and massive processing cost, even for inference, let alone training, and depend on power-hungry GPUs to achieve near real-time performance [22, 25, 28]. In the context of real-time stereo-vision for resource limited systems, even though the state-of-the-art matching performance of CNN based solutions are promising, they do not provide a viable option, as of yet. Following these restrictions, it still makes sense to improve upon pre-deep learning algorithms.

The CT is based on relative intensities between a center pixel and surrounding pixels. Within the local neighborhood a pixel is represented by 1 if it is of lower intensity, otherwise 0. The bits are concatenated in some canonical ordering, forming a bit-string, referred to as the census value for the center pixel. Similarity is given by the Hamming distance. Contrary to statistical cost metrics, where it is assumed local pixels belong to the same distribution, the CT tolerates factionalism, providing good matching performance along object boundaries and robustness towards outliers. The CT is invariant under changes in gain and bias [45]. The CT is low-cost and suitable for hardware implementation [3, 4, 6, 7, 10, 19, 29, 31, 34–36, 42], often in combination with Semi-Global Matching (SGM) [15].

Over the years the CT has been researched leading to two key developments: (1) sparse CTs, where only selected neighborhood pixels are used for comparison, requires fewer comparisons, which can be distributed over larger areas, and results in shorter bit-strings, saving resources for the subsequent matching [4, 6, 19, 31] and (2) non-centric comparison schemas, where arbitrary neighborhood pixels are connected for comparison by edges, according to predefined patterns, depending less (or not at all) on the center pixel, resulting in better matching accuracy and robustness towards noise [10, 23, 40]. However, the handcrafted CT methods are bound with respect to symmetry, edge length or neighborhood.

In the previous work [2], it was established that optimizing the census comparison schema using a Genetic Algorithm (GA), referred to as GACT, lead to higher matching accuracy and/or lower resource requirements than established CT methods. It was also concluded that outcome was highly dependent on training data (KITTI vs Middlebury) and that the CT should benefit from a larger neighborhood (KITTI).

In this paper, thanks to a new hardware-in-the-loop implementation, the GA training process is accelerated by a factor of 30, enabling evaluation of GACT windows of different size and shape. The experiments show that: (a) the GACT window size has a big effect on matching performance; (b) as GACT is defined for a number of edges (sparse) a change of the window size does not come at a great cost (assuming a trained mask), as the subsequent stereo-matching only depend on the produced bit-string; (c) lateral windows make for improved matching and save resources; and (d) that GACT is suitable for FPGA implementation.

The remainder of this paper is arranged as follows. First related works with respect to the CT are presented followed by an introduction to GA. Then the experimental setup is

[1] http://www.cvlibs.net/datasets/kitti/.

[2] http://vision.middlebury.edu/stereo/eval3/.

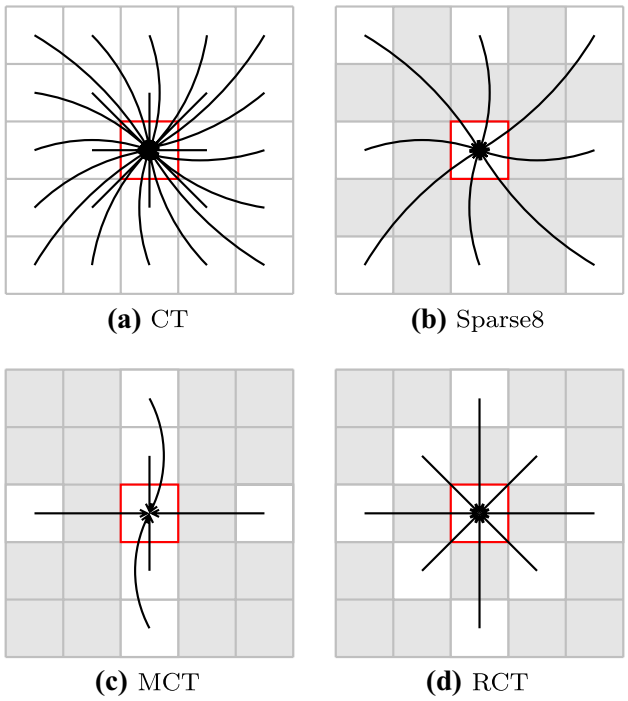


Fig. 1 **a** Shows the original CT, for a 5×5 neighborhood, and **b–d** show different sparsity schemas, discarding greyed out pixels, to save computational resources

described in terms of parameters, dataset and implementation. The experimental results are presented and discussed before the paper is concluded by some final remarks.

2 Related work

The Census Transform (CT) [45] is a non-parametric local transform, meaning that a pixel value is replaced by a value based on intensity ordering within a local neighborhood. In this case a bit string where a bit is set to 1 if the corresponding neighborhood pixel is of lower intensity, otherwise 0. Two census strings are compared using the Hamming distance. The comparison schema of CT is shown in Fig. 1a. CT can be formalized by the following equations:

$$\xi(p, p') = \begin{cases} 0 & p \leq p' \\ 1 & p > p' \end{cases} \tag{2}$$

where p, p' represent pixel intensities.

$$C(p) = \bigotimes_{p' \in N(p)} \xi(p, p') \tag{3}$$

where $\otimes$ represents the concatenation operation and $N(p)$ defines the neighborhood around a pixel p.

The similarity of two pixels is given by the Hamming distance between the bit-strings:

$$d(p_1, p_2) = \text{Hamming}\big(C(p_1), C(p_2)\big) \tag{4}$$

which is also referred to as the matching cost.

The CT relies heavily on the neighborhood center pixel, which makes it sensitive to noise. The modified CT [11] is instead based around the neighborhood mean intensity, $\bar{p}$, leading to an update of Eq. 3 to

$$C(p) = \bigotimes_{p' \in N(p)} \xi(\bar{p}, p') \tag{5}$$

The authors conclude that this increases robustness, with respect to noise, and the ability to capture the image structure, which is important for classification and matching. However, at the cost of a higher computational complexity. This method will be referred to as MeanCT.

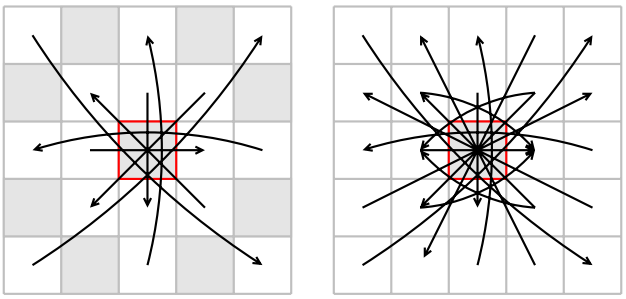
For the Sparse CT [4, 19], a subset of pixels within the neighborhood are selected for CT. The sparse factor, $S = n^2$, defines the sampling rate over the CT window, e.g. for sparse factor 16, 1 in 16 (4×4) pixels is selected for evaluation. The authors show that Sparse CT improves on the CT, given the same number of sample points, i.e., a larger receptive field is beneficial compared to higher resolution. However, an increased sparse factor does infer larger neighborhoods and consequently higher buffering costs, not to be underestimated for resource limited systems, for which the method was originally intended. Within this context, a stronger argument for the sparse CT is that, until a sparse factor of 16, it shows a marginal drop in accuracy compared to the CT, of the same neighborhood size. Figure 1b shows sparse CT with sparse factor of 2, hereafter referred to as Sparse8, as there are 8 sample points. This is extended to a full checkerboard pattern, by adding 4 diagonal sample points, referred to as Sparse12. Note that Sparse8 and Sparse12, are not consistent with the original definition of sparse CT, but are adapted to fit within a 5×5 neighborhood.

Mini-census (MCT) [6] is also a sparse CT targeting resource limited systems. MCT is defined by just 6 sample points, in a 5×5 window, Fig. 1c, to reduce calculation cost and memory resources. A similar example is the Retina CT (RCT) [31], Fig. 1d, with an 8-point circular pattern, inspired by the human retina.

An evolution of the MeanCT, hereafter referred to as Quarternion CT (QCT) [26], makes use of both the center pixel and the neighborhood mean intensity, thereby extending Eq. 2 to

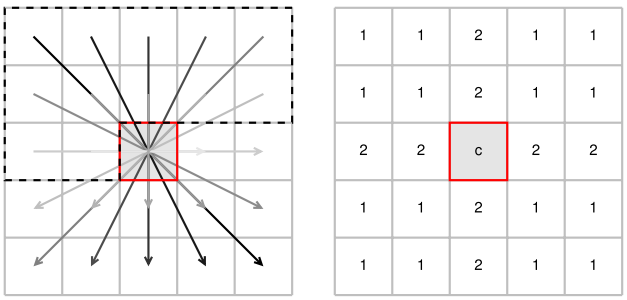
$$\xi(p, p') = \begin{cases} 00 & p' \leq min(p, \bar{p}) \\ 01 & p' < p \\ 10 & p < p' \\ 11 & p' \geq max(p, \bar{p}) \end{cases} \tag{6}$$

resulting in a quaternion, and consequently bit-strings of twice the length. This shows an accuracy improvement (for



(a) GCT8 **(b)** GCT16

Fig. 2 GCT schemas for 8 and 16 edges



(a) CSCT **(b)** Weights

Fig. 3 Center-symmetric schema of CSCT and the optional weights along the central rows and columns

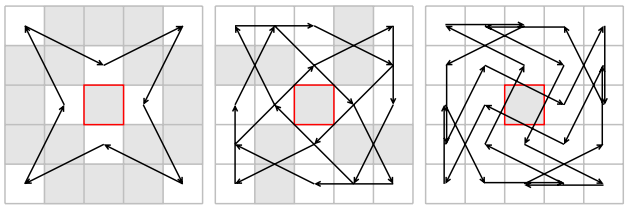
stereo matching) in the Middlebury evaluation, but according to the authors the benefit should be greater for real world images.

The Generalized Census Transform (GCT) [10] is a family of defined masks, in a 5×5 neighborhood, with different levels of sparsity. The comparison schema is defined by a set of coordinate pairs, $\{(c_1, c_1'), (c_2, c_2'), \ldots, (c_i, c_i'), \ldots, (c_n, c_n')\}$, where $n \in \{1, 2, 4, 8, 12, 16\}$. A coordinate pair, (c_i, c_i'), is connected through what is referred to as an edge. The key difference to previous work is that neither c_i nor c_i' refer to the center pixel. Figure 2 shows the 8 and 16 edge GCT. To define GCT, Eq. 3 is updated as follows:

$$C(p) = \bigotimes_{\substack{1 \le i \le n \\ p + c_i, p + c_i' \in N(p)}} \xi(p + c_i, p + c_i') \tag{7}$$

One advantage of GCT is that a 5×5 GCT is comparable to a 7×7 sparse CT and hence the number of line buffers can be reduced.

Presented at the same time as GCT, the Center-Symmetric CT (CSCT) [40] similarly compares pairs of pixels within the census window, albeit center-symmetric. Following the earlier notation, this can be expressed as $\{(c_1, -c_1), (c_2, -c_2), \ldots, (c_n, -c_n)\}$, where $n = (N_W * N_H - 1)/2$. The 5×5 CSCT is shown in Fig. 3a. Contrary to GCT the definition of CSCT extends to different window sizes, but not to different levels of sparsity. However, it will always produce bit-strings of half the size as compared to the original CT, while considering all (but the center) pixels, although at the cost of slight increase in error rate. A second contribution is the introduction of edge weights along the central rows and columns, as shown in Fig. 3b. This will be considered as adding edges but can, according to the authors, be implemented using lookup tables. With weights along the central rows (hwCSCT) or both central rows and columns (wCSCT) boosted results passed CT, while saving resources.



(a) SCT8-2-2 **(b)** SCT16-2-2 **(c)** SCT24-2-1

Fig. 4 Selected SCT sequences with 8, 16 and 24 edges

The Star Census Transform (SCT) [23] extends GCT by defining masks of symmetrical sequences of connected edges, of equal length, forming star-shaped scan-patterns around the center. Let $(c_1', c_2', ..., c_n') = (c_2, ..., c_n, c_1)$ and Eq. 7 holds true for SCT. The authors states that by evaluating masks with different numbers of sample points and edge lengths, candidates are found that improve on CT, MCT and GCT, particularly on CT and MCT with respect to noise, whether Gaussian or impulse. Figure 4 shows the best SCT for 8, 16 and 24 edges.

The Adaptive Census Transform (ACT) [34] applies the principle of ADaptive Support Weights (ADSW) [21] in a census context. For ADSW pixels within the aggregation window are weighted, based on center pixel similarity and proximity. For ACT the binary number is replaced by a weight, an integer defined by a hardware-friendly approximation function, exploiting intensity similarity (the proximity term is omitted as its contribution is limited for small aggregation windows). The census matching cost is then defined by SAD instead of the Hamming distance. This makes ACT expensive to implement, due to the weight function and the added complexity of SAD compared to Hamming. To describe ACT Eq. 2 is updated as follows:

$$\xi(p, p') = \begin{cases} -w & p \le p' \\ w & p > p' \end{cases} \tag{8}$$

and w defined as:

$$w = \begin{cases} p_n & \lfloor \Delta c/\gamma_c \rfloor = n, 0 \le n \le 7 \\ 8 & 8 \le \lfloor \Delta c/\gamma_c \rfloor < 12 \\ 4 & 12 \le \lfloor \Delta c/\gamma_c \rfloor < 16 \\ 2 & 16 \le \lfloor \Delta c/\gamma_c \rfloor < 20 \\ 1 & 20 \le \lfloor \Delta c/\gamma_c \rfloor < 24 \\ 0 & \lfloor \Delta c/\gamma_c \rfloor \ge 24 \end{cases} \tag{9}$$

where Δc represents the absolute intensity difference between p and p', and parameters set as: $\gamma_c = 16$, and p_o to p_7 are 64, 48, 32, 32, 16, 16, 16, 16. The weight function was also adopted for cost aggregation (ADSW). The implementation was later extended [7] incorporating Support Local Binary Pattern (SLBP) [33] and sparse ACT windows. With SLBP a census vector is calculated with respect to each pixel in the census window, not only the center pixel, resulting in as many census strings as pixels, i.e., 9 vectors for a 3×3 window. To allow for larger census windows and to compensate for the additional complexity of SLBP, a sparse approach was adopted [7], where only pixels along the horizontal, vertical and diagonal lines intersecting the center pixel were included. Later a simplified implementation [35], eliminating the SLBP component, for embedded heterogeneous system based on Xilinx Zynq SoC, was presented.

Adaptive window patterns for the CT [24] are based on the idea that uniform image regions require less complex census patterns than non-uniform. Hence, for a CPU implementation, the computational complexity can be reduced by applying different census transforms. A guidance mask, based on Canny edges, and the region intensity statistics (mean and variance) dictates the choice of an 8, 12 or 20 pixel pre-defined census mask in a 9×9 neighborhood. Similarly, a method based on adaptive census window size/shape has been proposed [20], where Sobel gradient images are used to select between square (3×3), portrait (11×3) or landscape (3×11) shaped census windows. Considering all pixels the method performs slightly worse than the corresponding fixed window size, 11×11, but better along depth discontinuities. Another contribution [36] lets the Sobel image dictate as to whether adopt a 5×5, 7×7 or 9×9 CT in an FPGA implementation. Here, to increase accuracy, as for FPGAs all CT-alternatives have to be processed in parallel, thus invalidating the resource reduction argument.

Alternatively, to the Hamming distance, the Tanimoto [37] and the Dixon–Koehler [8] distances, achieve higher matching accuracy at the cost of increased complexity. Tanimoto distance focuses on the ones of the matching bit-strings (1-intersection/union). The Dixon–Koehler is the product between the normalized Hamming and Tanimoto distances. In a comparison study for FPGA implementation [42], a number of different CT window sizes, 5×5 to 23×23, and Hamming, Tanimoto and Dixon–Koehler similarities were evaluated, with respect to matching accuracy and resource requirements. It was concluded that a 13×13 Dixon–Koehler CT produced a better matching result than a 23×23 Hamming, at similar cost in terms of resources. However, the real benefit of Dixon–Koehler is that it can reach a higher level of accuracy, but resources scale badly with window size.

Finally, composite costs, where part of the cost constitutes of CT, are also adopted. These methods are out of scope here, as the focus is on stand-alone CT. AD-Census [29], combines a weighted sum over CT and color-based SAD, where the AD component provides good matching support for textured or slanted areas, while CT preserves edge information. The neighborhood for CT was set to 9×7, the largest possible of odd rows and columns, to produce strings that fit within 64-bit registers. The stand-alone contribution of this larger footprint CT, referred to as CT-7x9 because of the row-column notation, is interesting for comparison, especially since AD-Census has since been revised with GCT [23]. Another example of a composite cost is combining MeanCT over the images, MeanCT over gradient images and SAD [3].

3 Genetic algorithm

The Genetic Algorithm (GA) [17] is a population-based optimization method, proposed in the 60's, extended and popularized in 1989 [12], that belongs to the family of Evolutionary Algorithms [44]. In 1989 extended and made popular, in the context of optimization [12]. GA is based on the evolution of the species, in which the new solutions are created from previous ones and only the stronger solutions survive. GA has been successfully applied in different research areas, as stereo matching [13], real time systems [27] or neuroengineering [32].

GA will have a population, X, in which each individual of the population will be a solution to the problem. The representation of an individual will depend on the specifications of the problem. For the specific problem of GACT, each individual is represented by a CT mask, consisting of tuples of window coordinates, defining edges for pixel comparison. An individual i of population X can be described as

$$X_i = \{(x_1, x_1'), (x_2, x_2'), \ldots, (x_e, x_e'), \ldots, (x_n, x_n')\} \tag{10}$$

where x_e is the starting point of the eth edge, x_e' is the end point of the same edge and n is the number of edges. Additionally, each point is formed with 2 coordinates (r, c), where $r \in \{1, \ldots, R\}$ and $c \in \{1, \ldots, C\}$, for a census window of size $R \times C$.

The first step in GA is to initialize the population, which is formed by ps (population size) individuals, generated at random. Then follows the evolutionary process, generation

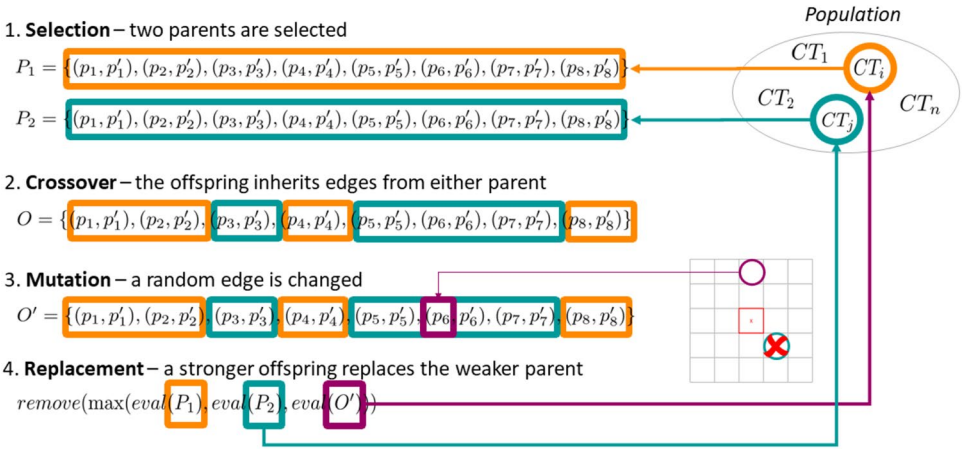


Fig. 5 Genetic algorithm for CT flowchart

by generation, where each iteration, or generation, is composed by four consecutive steps: parent selection, crossover, mutation and replacement. An example of one generation of GA, in a problem with 8 edges, is shown in Fig. 5. After one generation is terminated, a new one will start until the maximum number of generations is reached.

Selection. The first step within one generation is to select parents to create the offspring. In this paper, two parents, P_1 and P_2, are randomly selected, from the entire population, to create one offspring (O). This is repeated for a defined number of offspring each generation. Hence, a high diversity within the population can be maintained, minimizing the chance to suffer local optima stagnation.

Crossover. After selecting the two parents, the new offspring, O, is created. Many different options to perform crossover can be found in the literature [9, 14]. In this paper, a Uniform crossover method has been selected. In this method, the information of O is randomly selected from the two parents with equal probability, as described below:

$$(o_e, o'_e) = \begin{cases} (p_{1_e}, p'_{1_e}) & \text{if } rand < 0.5 \\ (p_{2_e}, p'_{2_e}) & \text{otherwise} \end{cases} \tag{11}$$

where e is the eth edge of either the offspring (O), the first parent (P_1) or the second parent (P_2). Additionally, *rand* is a random number $\in [0, 1)$.

Mutation. Once the offspring is created, this is perturbed, to explore neighbor solutions. In order to apply this perturbation, a random position within an edge, either the start or the end point, is selected. Then, it will be replaced by a random position within the CT constraints.

Replacement. After mutation, the performance, also called fitness, of O is calculated ($f(O)$) and compared with the fitness of the worst parent ($f(P_{worst})$). If, $f(O)$ is better than $f(P_{worst})$, then P_{worst} will be replaced by O. On the contrary, if $f(O)$ is worse, then O is discarded. The description of the fitness calculations is described in Sect. 4.1.

4 Experimental setup

The experiment is setup as a two-phase process, separating training and an evaluation between two processing platforms, as shown in Fig. 6. During training GA is applied to find new GACT comparison schemas (or masks). This involves, for each candidate individual, transforming the input images, in accordance to the mask, and perform stereo matching, followed by an evaluation of the resulting disparity map, with respect to the ground truth. The training phase is implemented on a Xilinx ZCU104, a processing platform combining a CPU and an FPGA, where the most arduous part of the process, the stereo matching, is hardware accelerated, significantly reducing time for training. During the evaluation phase the GA derived mask is evaluated, on a larger dataset, using a MATLAB implementation, for consistency with previous work [2]. First, the parameters for the experiment will be presented followed by a description of the implementation for ZCU104.

4.1 Parameters, data

The GACT experiments are defined by the number of edges and window sizes:

- Number of edges: 8, 16 and 24.
- Square windows: 3×3, 5×5, 9×9, 15×15, 21×21.
- Lateral windows, following *rows* $\times$ *columns*-notation, where *columns* $= 2 * rows - 1$: 3×5, 5×9, 7×13, 9×17, 11×21, 15×29.

In addition, lateral windows 3×7, 3×9, 3×29 and 5×29 are used to compare GACT to established CT methods.

For GA the following parameters are used:

- population size = 30
- offspring size = 8
- max evaluation = 6000

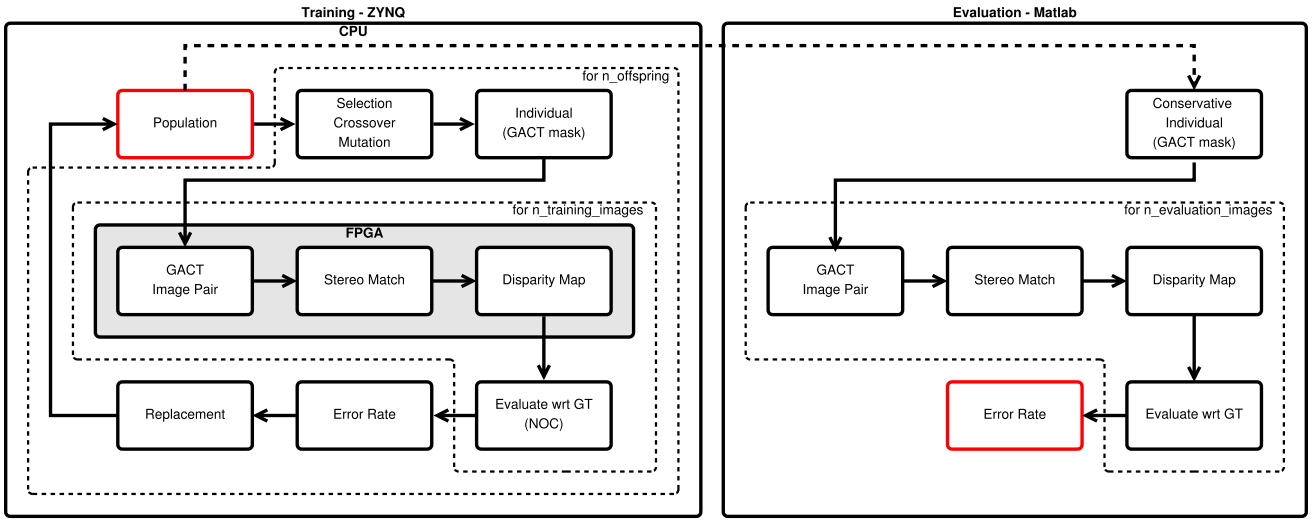


Fig. 6 Flowchart of the two-phase experimental setup

- mutation rate = 20%

GA is repeated 10 times for each CT setting and the median candidates evaluated. The hyper parameters are carried over from preceding experiments [2]. For the current experiment, the search space has expanded, possibly warranting a larger number of evaluations. However, a consistent result, when repeating the experiment, suggests a satisfactory balance between exploration and exploitation with sufficient number of evaluation.

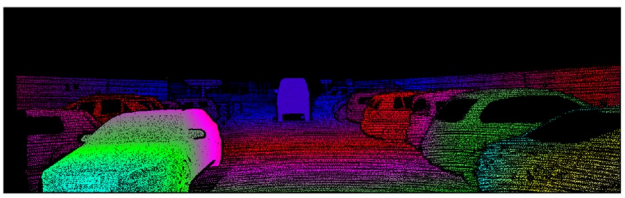
Through stereo matching disparity maps for GACT masks are obtained. Here, the Hamming distance between census bit-strings for 9×9 aggregation windows gives the matching cost for a disparity hypothesis. The disparity range is set to [0, 255] for training (FPGA) and [0, 230] for evaluation (MATLAB). From the disparity hypotheses the best candidate is selected according to a winner takes all (WTA) strategy. No left-right consistency check (LRC), no subpixel interpolation, propagation, refinement filters, etc., are applied.

Disparity maps, whether during the training or evaluation phase, are evaluated using KITTI[3] stereo evaluation. The KITTI 2015 benchmark [30] consists of training and evaluation datasets, with an associated ranking list. The KITTI dataset targets autonomous driving and the scenes represent real-world natural images with noise, reflections, challenging contrast, etc. Fig. 7 shows an example with ground truth. For the training set the stereo pairs are completed by the ground truth images, enabling supervised learning. As KITTI evaluation is to be performed once, and associated with a single publication, until a more complete stereo framework around GACT is finalized, the experiments are carried out on the training set. This dataset is split into local training and evaluation subsets. To limit training time, and as GACT works well with a relatively small amount of training data, 5 training scenes were selected at random (seq. no. 39, 101, 3, 166, 40), leaving 195 for evaluation. The end-point error is defined as $< 3px$ or $< 5\%$. During training, only non-occluded (NOC) pixels are considered. For evaluation occluded (OCC) results are also presented.

4.2 Implementation/Processing platform

To be able to extend the experiments, performed in previous work [2], for multiple window sizes and different shapes, the time required for training had to be considerably reduced, even though parallel for-loops, utilizing 8 cores, one per

(a) Left image



(b) Ground truth, NOC

Fig. 7 KITTI 2015 sample

[3] http://www.cvlibs.net/datasets/kitti/.

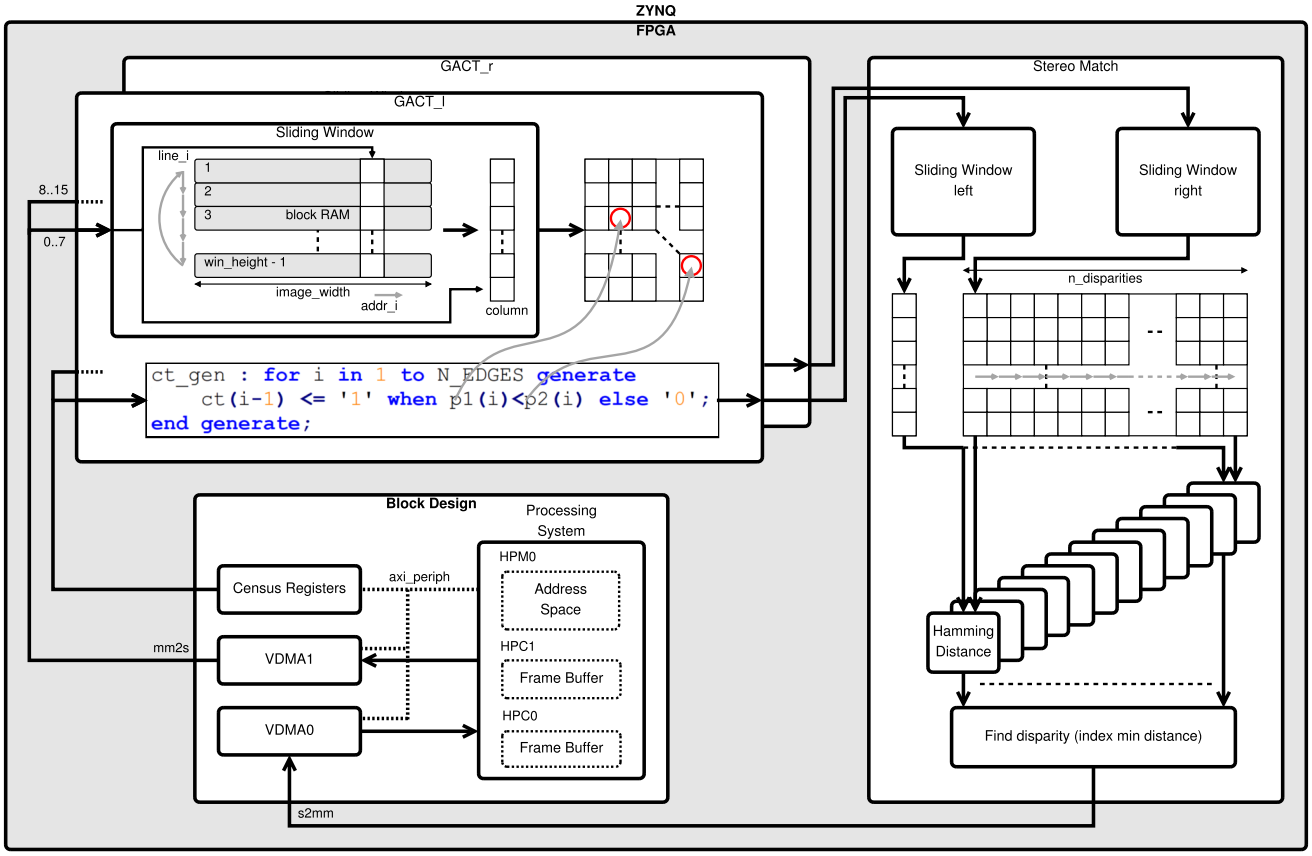


Fig. 8 Experimental setup FPGA

offspring, were adopted. As the intended future processing platform for GACT is FPGAs, more specifically GIMME2 [1], this platform can also serve as an accelerator during training. However, the non-deterministic data pattern of GACT individuals, proposed by GA, in correlation with large windows and disparity range, deemed GIMME2 short on resources. Instead the more powerful Xilinx ZCU104 was used for the experiments. Both share the Zynq SoC platform and the source/IP cores can be reused between them.

The most computationally expensive, and hence time consuming, part during training is the stereo matching, followed by the census transform. These are implemented on the FPGA in a pipelined design, benefiting from the parallel processing capabilities of the FPGA. GA itself is low-cost. The GA-indices are randomly selected inferring a non-deterministic data-access pattern, which is not suitable for FPGA implementation. Evaluation involves division which is costly on the FPGA. Since neither GA nor evaluation is part of the stereo algorithm, there is no motivation for it to be implemented on GIMME2.

In accordance with the Xilinx Vivado design flow, the implementation will be described starting from the FPGA side, also referred to as the Programmable Logic (PL), followed by the CPU side, or Processing System (PS). This is contrary to the processing flow of training, which is controlled by the PS application. Returning to Fig. 6, the reminder of the section will focus on the left box, the training phase, and first the FPGA part, the gray block.

4.2.1 FPGA/PL

At a high abstraction level the FPGA-side of the design can be divided into three components; block design, GACT and stereo matching, as shown in Fig. 8. Block design is part of the Zynq design flow and specifies system properties and the PL/PS interface, here setup for exchange of images (left, right and disparity) and GACT masks (as dictated by GA). Two GACT components (left/right) converts intensity images to images of census bit-strings, according to the GACT coordinates. Stereo matching calculates the disparity, using the Hamming distance. GACT and stereo matching are the core components, both relying on a forth component, the sliding window. A sliding window approach is necessary as on the FPGA, there are only enough resources to process a small part of the image, at a time. Following this approach an image representation is changed to a data stream, where a new pixel is presented every clock cycle. Below follows a more detailed description of the components.

Block Design The block design comprises of three IP cores: processing system, VDMA and census register. In the processing system the I/O, clocks, memory and PS-PL interface settings for the Zynq system are configured. The ZCU104 evaluation board can be selected as target device with a preset processing-system configuration with respect to the hardware. More interesting are design dependent configurations, such as the PS-PL interface, here with two high performance slaves, for image data transferal, and one master, for IP register control.

The VDMA IPs from Xilinx provide high performance memory mapped channels to enable data streaming between the FPGA fabric and the system memory. In this design the read and write channels are separated into two separate VDMA components. The read channel, VDMA1, handles a 24-bit stream (non-optimal), divided into 8-bits of left and right pixel intensities, respectively, and 8-bits of (natural) ground truth (not currently used). The write channel, VDMA0, is setup as an 8-bit stream to encode 256 levels of disparity. The control/status of the VDMA cores is performed by writing/reading register values. The register space is accessed (from the PS) over the AXI4-lite master interface.

The census register IP is an AXI4 peripheral, with the straight forward objective to setup a shared memory area between the PS-PL for GACT control. The census register memory area holds 25 32-bit register, one control register and 24 edge register. An edge is represented by two points, start and end point, each having two coordinates. For this application directionality is of minor/marginal importance and is hence neglected. Each coordinate is encoded by 8-bits to fit an edge into a 32-bit register. Census registers are mapped as signals to the GACT component.

Sliding Window The function of the sliding window is to buffer data, to provide a small image patch, from an image data stream. Both the GACT and stereo match blocks are window based and incorporates this component. The sliding window has generic parameters to be able to cope with different window sizes, data widths, and image sizes. For this experiment different window sizes and data widths are used by the GACT and stereo matching components.

The sliding window requires $win\_height - 1$ row buffers. As there is a large amount of data to be stored, the buffers are placed in the FPGA block RAM, which is on chip, but memory 'circuits' and not logic resources. The buffers are controlled by two indices, *addr_i* for horizontal position and $line_i$ for vertical. For every valid input data, the row buffers are read for the current address, *addr_i*. The resulting data column is synchronized with the input and put in a shift register, with the width of the window. The input data is written to *addr_i* to the oldest row buffer, *line_i*, and the address index is incremented. When reaching an end-of-line or start-of-frame signal, the address index is reset, and the row index cycled. Hence, every row buffer address is written to once but read *win_height* times until the window overlaps. The sliding window block is shown to the upper left, embedded the GACT block, in Fig. 8.

GACT Two parallel GACT-components are instantiated, to handle the left and right images, respectively. Provided the sliding window component and the GACT coordinates the implementation is straight forward, as can be seen in Fig. 8. Each position in the census bit string is set by comparing two window coordinates, connected by an edge. The current implementation supports 24 edges. Any lower amount of edges can be used as edges point to the same coordinate by default and is hence self-cancelling. Similarly, the same circuit can be used for larger and smaller windows by restricting the coordinate indices. This is however controlled by the GA on the PS side. Input data width is 8-bits, and output is 24-bits.

Block Match The stereo correspondence is calculated using block matching. For each pixel in the reference image, extract a small image patch around the pixel, and compare for similarity against patches from the target image, over a range of horizontal offsets, the disparity range. The image patches are referred to as aggregation windows, here implemented by two 9×9 sliding window components, one for each 24-bit census stream. The output of the right window is put into a shift register, with the width of the disparity range, in this design 256 disparity hypothesis are evaluated. Similarly, for census transformed images is defined by the Hamming distance and is realized by a separate component. 256 parallel Hamming components calculates the similarity for the current position in the reference image and an offset (delay) of 0 to 255 pixels in the target image. As the Hamming distance is calculated over the aggregation window a two clock-cycle approach is adopted, starting with the column sums (vertical), followed by horizontal aggregation. From 256 hypothesis the best match is to be found, along with its offset. This is implemented as a tree-like tournament, of different branching factor, over 3 clock cycles. The winner, the patch with the smallest distance, has a disparity of the corresponding offset, an 8-bit value, which is mapped to the block design and VDMA0.

To carry out the experiments, two different FPGA implementations of the same design were derived, one for square census windows and one for lateral. The implemented circuits handle worst case scenarios, i.e. maximal window size, 21×21 and 29×15, respectively, with 24 census edges.

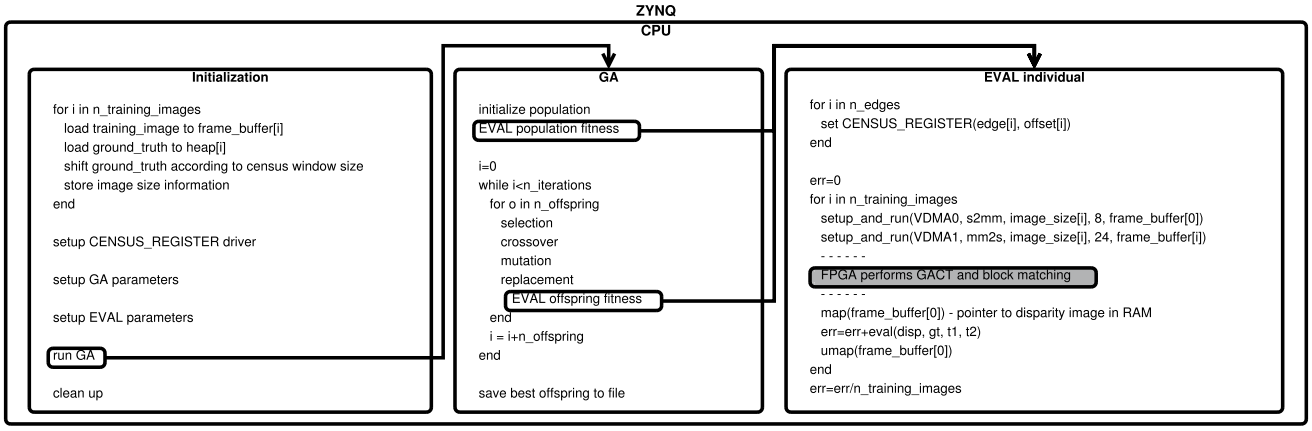


Fig. 9 Experimental setup CPU

Experiments for smaller windows and fewer edges could be run using the same implementations. (However, with shifted output). The design is fully pipelined, clocked at 50 MHz and handles 256 levels of disparity.

4.2.2 CPU/PS

From the hardware design, a configuration file is created containing the information for generating low-level system startup files and the device tree, describing the hardware (with addresses) for the operating system, in this case Petalinux, a Xilinx specific Linux distribution for the Zynq systems. One specific configuration was to, in the device tree, reserve part of the PS RAM memory for image frame buffers, i.e., set an upper RAM limit for Linux, so that the operating system would not interfere with the frame buffers. Greatly simplified, the address space can be divided into three parts: the normal RAM memory, the reserved RAM-area for frame buffers, and the hardware address space. Thanks to Linux GA is a straight forward application, on the CPU side, which can be hierarchically divided into three parts: initialization, GA and evaluation, as shown in Fig. 9.

To minimize processing and data transfer KITTI training samples were combined into one file per sample, containing both left and right intensity images, together with an integer ground truth value (for future use). This completes a 24-bit 3 channel image.

Initialization The application begins with an initialization phase. First, the training images are loaded. As these are to be forwarded to the FPGA, and not to be processed by the PS, they are loaded to static addresses in the frame buffer memory area, outside of the memory range of the operating system. Next the ground truth images (of float precision) are loaded into allocated heap memory (RAM), as these are only to be accessed by the PS application. The ground truth needs to be shifted, as the output from the FPGA is not padded, and depends on census (varying) and aggregation (fixed) window sizes. The smaller the census window (compared to the supported size) the greater the shift required to align the images. KITTI images are of slightly different size, adding another requirement on the application (on both PS and PL sides).

With the image data loaded the next part of the initialization phase is to setup the census register driver and the parameters for GA and evaluation. The census register driver provides an interface for manipulating census registers, in the hardware address space, from the PS application. Later, candidate GACT masks will be shared with the PL through these registers. The driver requires the hardware address and the census size and clear the associated memory area at initialization. GA parameters are setup; population size, number of offspring, number of evaluations and mutation rate, as mentioned earlier, along with more experiment specific parameters such as census window size and the number of edges. Finally, the evaluation parameters are setup. These are the thresholds associated with KITTI evaluation, pointers to the input images and the output (disparity) image (frame buffer addresses), pointers to the ground truth images (heap), addresses to the VDMA IP cores (hardware address space) and information about image size. There is a distinction between the GA and the evaluation when it comes to data. GA derives candidate masks and requires the fitness, independent of how the evaluation is performed and on what data. The evaluation, on the other hand, is independent of GA data.

GA The implementation of GA is straight forward following the algorithm described in Sect. 3 and shown in Fig. 9. The algorithm is neither particularly space or time consuming (and hence not implementation critical).

An individual in this application is defined by a GACT comparison schema of a specified number of edges. This set of edges can be compared to a genome and each edge a gene. As described, an edge can be fitted into a 32-bit register, and an equivalent 4-byte edge datatype is defined. A GACT mask is simply defined as an array of edges, and a population as an array of individual masks.

First the population is randomly generated in accordance with the GA parameters. Over generations offspring is generated from the population through selection (two random individuals from the population are selected as parents), crossover (combination of edges from the parents) and mutation (change a random edge). Finally, stronger offspring replaces the weaker of its parents and the population is set for the next generation.

Evaluation needs to be performed, first for the initial population, and continuously throughout for every offspring. Before running evaluation, the current individual needs to be presented to the FPGA over the census_register.

EVAL The EVAL part handles the image data, i.e., stereo image(s), disparity map and ground truth. The transferal of image data between the PS and the PL is done using VDMA IP cores, implemented on the FPGA side. First VDMA0, for receiving the disparity map, is setup. This includes specifying image size, data width, frame buffer address and resetting and starting the core. For VDMA0 the width of the stream is 8-bit to support the disparity range. The frame buffer address for the disparity map is always the same. Next the VDMA1 core is setup similarly, but this time the 24-bit image stream of the stereo images is sent to the FPGA. During the initialization the training images were loaded from files directly into different frame buffers. Hence, it suffices to change the frame buffer address instead of reloading images.

The FPGA performs the census transform, according to the mask, and stereo matching before the resulting disparity map can be read from the VDMA0 frame buffer. The frame buffer is mapped into user space and the disparity image is compared pixel by pixel to the ground truth (also loaded during the initialization) returning the error rate, for non-occluded pixels, given the evaluation parameters. The frame buffer is then released.

This process has to be repeated for each training image. The fitness of the individual is the average error rate, over the set of training images.

5 Experimental result

In this section, the experimental results are presented. Firstly, the results from training are presented. Conservative GACT candidates have been evaluated to investigate how parameters affect the matching result, followed by and analysis of derived GACT patterns. This is performed for square followed by lateral census windows. Then results regarding the implementation are presented and discussed. Finally, GACT masks have been compared to established CT methods.

5.1 Training

In previous work it was established that for the KITTI dataset the correlation between training and evaluation result was strong, i.e. a training candidate with low training error rate will, with a high probability, have a low evaluation error rate [2]. It was also concluded that GACT did not converge to a single solution, but to many similarly good solutions sharing common traits, representing the information from the training data. To investigate the discrepancy between different solutions, training was repeated 10 times for each parameter set. The training results, for square windows of different size and number of edges, are shown in Table 1. Comparing the max and min for the different entries, the divergence is small, compared to the total error. For the worst case, 3×3 window and 8 edges, the difference is 0.36% (21.75–21.39%). The GA produces solutions of acceptable consistent values, and hence a single training run could suffice. The training results for rectangular census windows are shown in Table 2. The conclusions for square windows hold true for rectangular windows, however the training error rates are lower.

5.2 Evaluation—square GACT

Even though training result are acceptably accordant, the median training candidates were selected for evaluation, to achieve the highest level of consistency, at a potential loss of highest accuracy possible. However, this is to reflect the result if running GA once.

First square GACT windows will be considered. The evaluation results for GACT of 8, 16 and 24 edges of different window sizes are presented in Fig. 10 and Table 3, where the error rates are plotted against the number of pixels in the census window. From the evaluation results it can be concluded that larger census windows reduce the error, at a exponentially decaying rate. The final step almost doubles

Table 1 Square windows–training error rates–% bad pixels, NOC, end-point error $< 3px$ or $< 5\%$

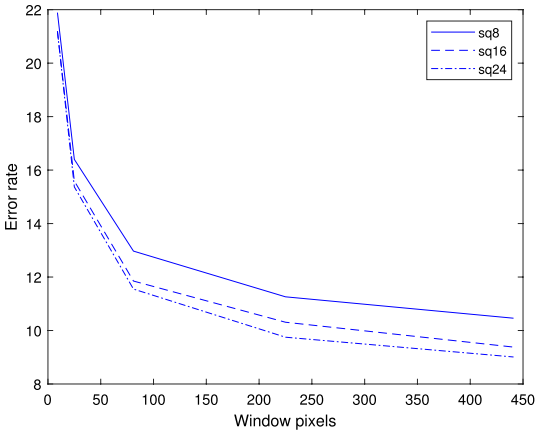
Window	Pixels	Edges	Min	Mean	Median	Max
3×3	9	8	21.39	21.48	21.50	21.75
3×3	9	16	20.88	20.93	20.94	20.98
3×3	9	24	20.76	20.82	20.82	20.87
5×5	25	8	14.83	15.07	15.09	15.19
5×5	25	16	14.01	14.06	14.04	14.17
5×5	25	24	13.66	13.79	13.81	13.89
9×9	81	8	10.64	10.79	10.80	10.89
9×9	81	16	9.37	9.46	9.46	9.59
9×9	81	24	9.03	9.13	9.16	9.22
15×15	225	8	8.57	8.77	8.75	9.02
15×15	225	16	7.24	7.37	7.36	7.52
15×15	225	24	6.79	6.87	6.86	6.97
21×21	441	8	7.57	7.82	7.83	8.01
21×21	441	16	6.28	6.47	6.45	6.71
21×21	441	24	5.87	6.00	5.99	6.09

Table 2 Lateral windows–training error rates–% bad pixels, NOC, end-point error $< 3px$ or $< 5\%$

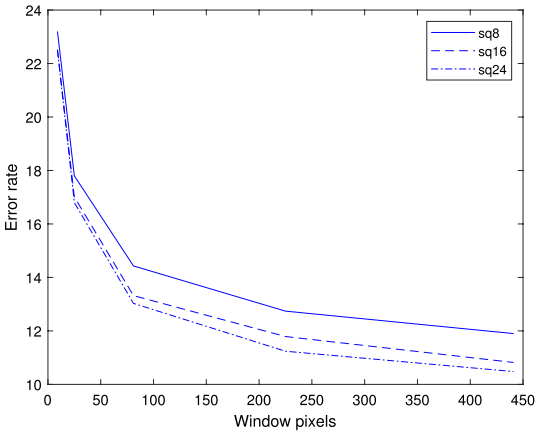
Window	Pixels	Edges	Min	Mean	Median	Max
3×5	15	8	16.70	16.88	16.89	17.03
3×5	15	16	16.09	16.17	16.17	16.24
3×5	15	24	15.98	16.02	16.00	16.10
5×9	45	8	11.37	11.54	11.53	11.74
5×9	45	16	10.34	10.45	10.45	10.57
5×9	45	24	10.10	10.22	10.20	10.35
7×13	91	8	9.32	9.44	9.40	9.66
7×13	91	16	8.09	8.24	8.25	8.37
7×13	91	24	7.89	7.95	7.95	8.05
9×17	153	8	8.34	8.45	8.45	8.63
9×17	153	16	7.10	7.19	7.17	7.31
9×17	153	24	6.69	6.80	6.81	6.89
11×21	231	8	7.76	7.88	7.86	8.18
11×21	231	16	6.38	6.55	6.53	6.77
11×21	231	24	6.00	6.10	6.10	6.20
15×29	435	8	6.98	7.17	7.18	7.37
15×29	435	16	5.67	5.78	5.79	5.88
15×29	435	24	5.32	5.39	5.39	5.48

the census window at a quite small accuracy improvement. It has previously been established that larger CT windows broaden object boundaries [3, 19], just as aggregation windows, where large induce foreground fattening [19], however, to a lower extent regarding CT [3], and that large CTs are unfeasible [42] or even detrimental [3, 19]. Applying GACT to too large census windows, edge location is optimized to minimize the error rate, omitting edges in unfeasible areas. It can hence be argued that the effect on GACT, in terms of accuracy, will not be detrimental if increasing the window size. Instead a steady state can be expected, where optimal accuracy is achieved, and extending window size beyond this point is a waste of resources. Another point regarding census window size is that the negative effect of too large census windows is suppressed by noisy data [19]. Here, experiments are based on KITTI, which comprises of natural, noisy images. Hence, in combination with GACT optimisation, too large windows should not be an issue.

Increasing the number of edges increases matching accuracy, as seen in Fig. 10 and Table 3. The improvement is larger when going from 8 to 16 edges than from 16 to 24 edges, and is relatively consistent across window sizes. Regarding the number of edges as compared to window size, only for relatively small windows, an increase in window

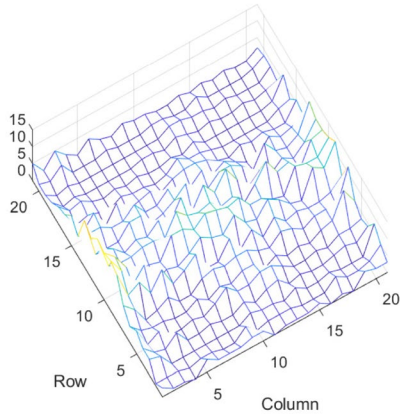


(a) NOC

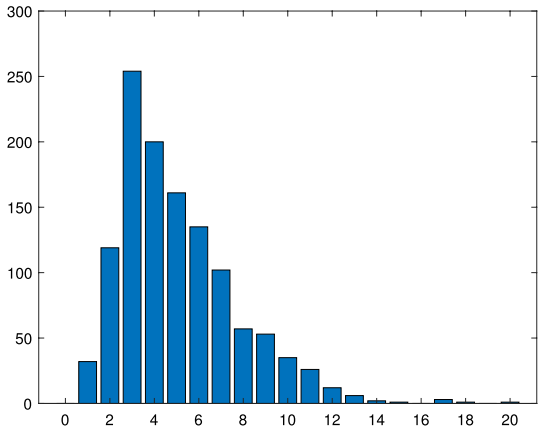


(b) OCC

Fig. 10 Square GACT error rates for different GACT windows sizes and number of edges



(a) Coordinate activation



(b) Edge lengths

Fig. 11 Square windows: histograms of coordinate activation and edge length, 30 runs, 21×21 neighborhood, 24 edges

Table 3 Square windows–evaluation error rates–% bad pixels, end-point error $< 3px$ or $< 5\%$

Window	Pixels	Edges	NOC	OCC
3×3	9	8	21.89	23.20
3×3	9	16	21.20	22.52
3×3	9	24	21.18	22.50
5×5	25	8	16.40	17.80
5×5	25	16	15.59	17.01
5×5	25	24	15.37	16.80
9×9	81	8	12.97	14.43
9×9	81	16	11.85	13.32
9×9	81	24	11.55	13.03
15×15	225	8	11.26	12.74
15×15	225	16	10.31	11.79
15×15	225	24	9.75	11.24
21×21	441	8	10.46	11.90
21×21	441	16	9.38	10.82
21×21	441	24	9.01	10.48

size can compensate for a larger number of edges. However, increasing edges comes at a much higher cost as it affects the later stereo matching, see Sect. 5.6. It can also be expected that there is a break-point, where introducing more edges will not lead to any accuracy improvement.

5.3 Distribution—square GACT

To conclude the results on square census windows a total of 30 training runs of the largest example (21×21, 24 edges) were performed. The activated coordinates for all masks were put in a histogram to show the GACT distribution for the training data. The histogram is shown in Fig. 11. Note that this shows that a coordinate within the census window is activated, but not to which other coordinate it is connected. It can be observed that GACT activates data, forming a horizontal ridge along the middle row, within the window. On the other hand, the top and bottom regions are more or less flat (non-activated), and hence a waste of resources. These

should instead be dedicated extending the window laterally, to better cover the activation distribution. Before going to the results of lateral windows a note on edge lengths. Similarly to the activation histogram, a histogram was created over the length of all edges. This histogram is shown in Fig. 11. As can be seen, the most common edge length is 3, and there are few edges longer than 10. Assuming a large enough window, edge lengths does not really increase with larger windows. Edges do not span the entire window, but they do however populate the entire width of the window.

5.4 Evaluation—lateral GACT

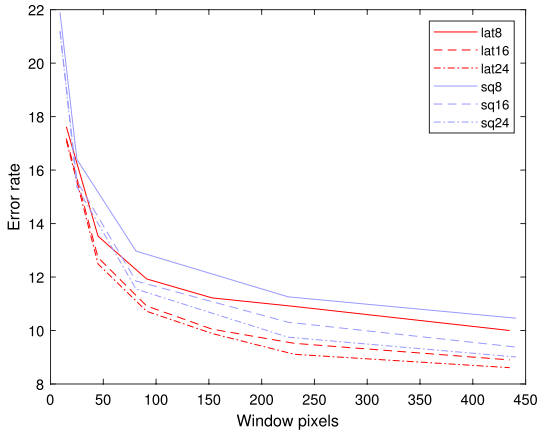
For the second part of GACT evaluation, lateral census windows were considered, to better correlate with the distribution of selected coordinates found for square windows. The number of rows and columns were set as a fixed ratio of $columns = 2 * rows - 1$. Similar to square windows, the training was repeated 10 times for each window size and parameter set and the median candidates were evaluated. The results are shown in Fig. 12 and Table 4.

The conclusion from the experiment is that, for a census window of a certain number of pixels, the GACT of a lateral shape performs better than a square. This is best visualized by Fig. 12 where the results for square windows have been included for reference. The lateral series (red) are below the corresponding square series (blue).

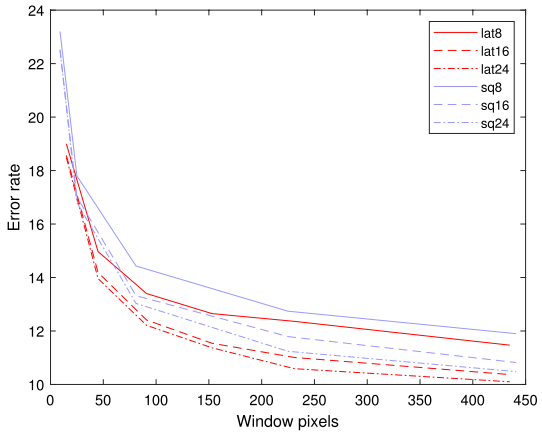
5.5 Distribution—lateral GACT

Similar to square GACT, lateral GACT was trained 30 times for the largest training parameters, i.e., 15×29 window with 24 edges, to investigate the distribution of coordinate activation and edge lengths. The results are shown in Fig. 13. Looking at the edge length distribution, Fig. 13b, it is resemblant of square GACT, with 3 being the most common length. Once again, edges spanning the entire window are deemed unfavorable. The coordinate selectivity histogram, Fig. 13a, on the other hand, shows a more interesting result. First, it should be noted that the distribution declines vertically from the center row. This indicates that not much information is lost by vertically limiting the window. Secondly, the horizontal stretch shows that coordinates are not activated along a ridge, but there are rather two separate parts: (1) a central distribution and (2) the most lateral regions of the window. Knowing this, a similar pattern can be distinguished from the square coordinate distribution, Fig. 11a.

To further investigate the nature of the coordinate distribution, experiments were run for different window sizes. The resulting distributions are shown in Fig. 14a–d. For the 15×29 window a normal probability distribution was estimated from the central part of the data, Fig. 14i. This was subtracted from the other histograms and the



(a) NOC



(b) OCC

Fig. 12 Lateral GACT error rates for different GACT windows sizes and number of edges. Blue represents square windows

remaining distributions are shown in Fig. 14e–h. It can be observed that the examples share the same central distribution with the supporting lateral regions following the expansion of the window. From the example, approximately half of the edges adhere to each of these parts, respectively. A hypothesis is that the central distribution represent matching on similarity while the peripheral edges help to eliminate uncertainty.

To conclude the evaluation results for lateral GACT, similarly as to square GACT, lower error rates will be achieved by increasing the window size and/or the number of edges. The better the result the higher the cost for an improvement. It can be noted that there is little difference between 16 and 24 edges until the two final window sizes. It can also be noted that lateral GACT16 is better than square GACT24. This is of great importance when considering implementation trade-offs for resource limited systems.

Table 4 Rectangular windows–evaluation error rates–% bad pixels, end-point error < 3*px* or < 5%

Window	Pixels	Edges	NOC	OCC
3 × 5	15	8	17.62	19.00
3 × 5	15	16	17.18	18.57
3 × 5	15	24	17.10	18.49
5 × 9	45	8	13.53	14.97
5 × 9	45	16	12.73	14.19
5 × 9	45	24	12.49	13.96
7 × 13	91	8	11.93	13.40
7 × 13	91	16	10.92	12.41
7 × 13	91	24	10.72	12.21
9 × 17	153	8	11.22	12.65
9 × 17	153	16	10.05	11.54
9 × 17	153	24	9.89	11.37
11 × 21	231	8	10.90	12.36
11 × 21	231	16	9.52	11.01
11 × 21	231	24	9.11	10.59
15 × 29	435	8	10.00	11.47
15 × 29	435	16	8.90	10.37
15 × 29	435	24	8.61	10.10

5.6 Implementation

Utilization for the two FPGA implementations, i.e., for square and lateral GACT widnows, are shown in Tables 5 and 6. GACT windows are set as large as possible, 21 × 21 and 15 × 29, for the Xilinx ZCU104 target board, considering a data width of 24 edges, a 9 × 9 aggregation window, and 256 levels of disparity. As can be seen from the utilization tables, the LUTs are the limiting resource. Clearly, a stereo matching considering a large number of disparity hypotheses will require a considerable amount of resources. Not as apparent, is the cost associated with the GACT, for the specific GA implementation. The GA works under the presumption that any census window coordinate can be selected, as an edge point, at any time. Implementation of this is straight forward for a CPU, where elements from an array-like structure, representing the image patch, can be accessed at constant time (very high level). However, on the circuit level of an FPGA, array indexing is a different proposition, as each index requires a signal tap, a physical connection for each bit of the data. An edge is defined by a start and an end point. The design supports up to 24 edges. Hence, there are 48 elements to be accessed each clock cycle. Adding to the problem is that the operation must be performed for both input images. It is unavoidable that the circuitry required for routing/multiplexing rapidly grows out of proportion as census window sizes increase.

However, when implementing a circuit for a trained, and hence deterministic, GACT mask the resource utilization can be considerably reduced, as routing can be limited to specific indices. The resource utilization for an arbitrarily defined GACT mask is represented by the first column of Table 7. This can be compared to the training setup, Table 6.

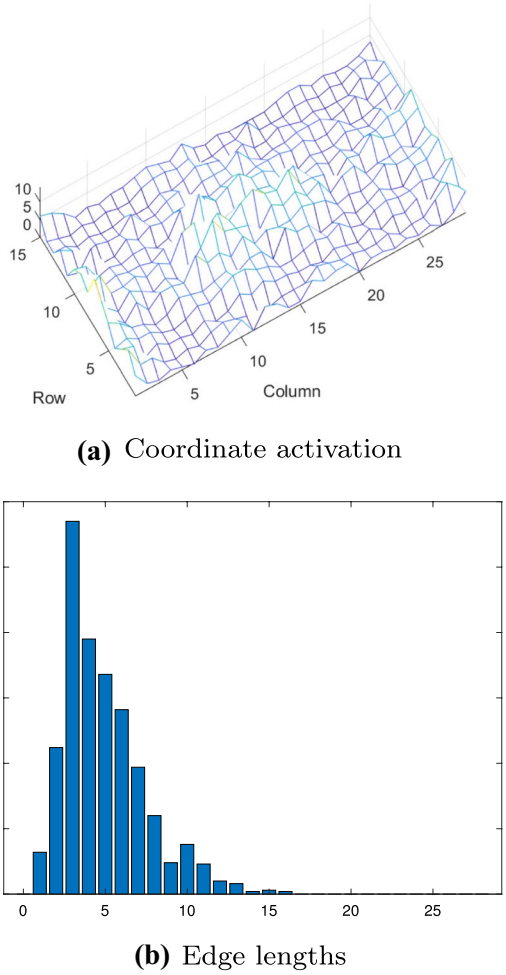
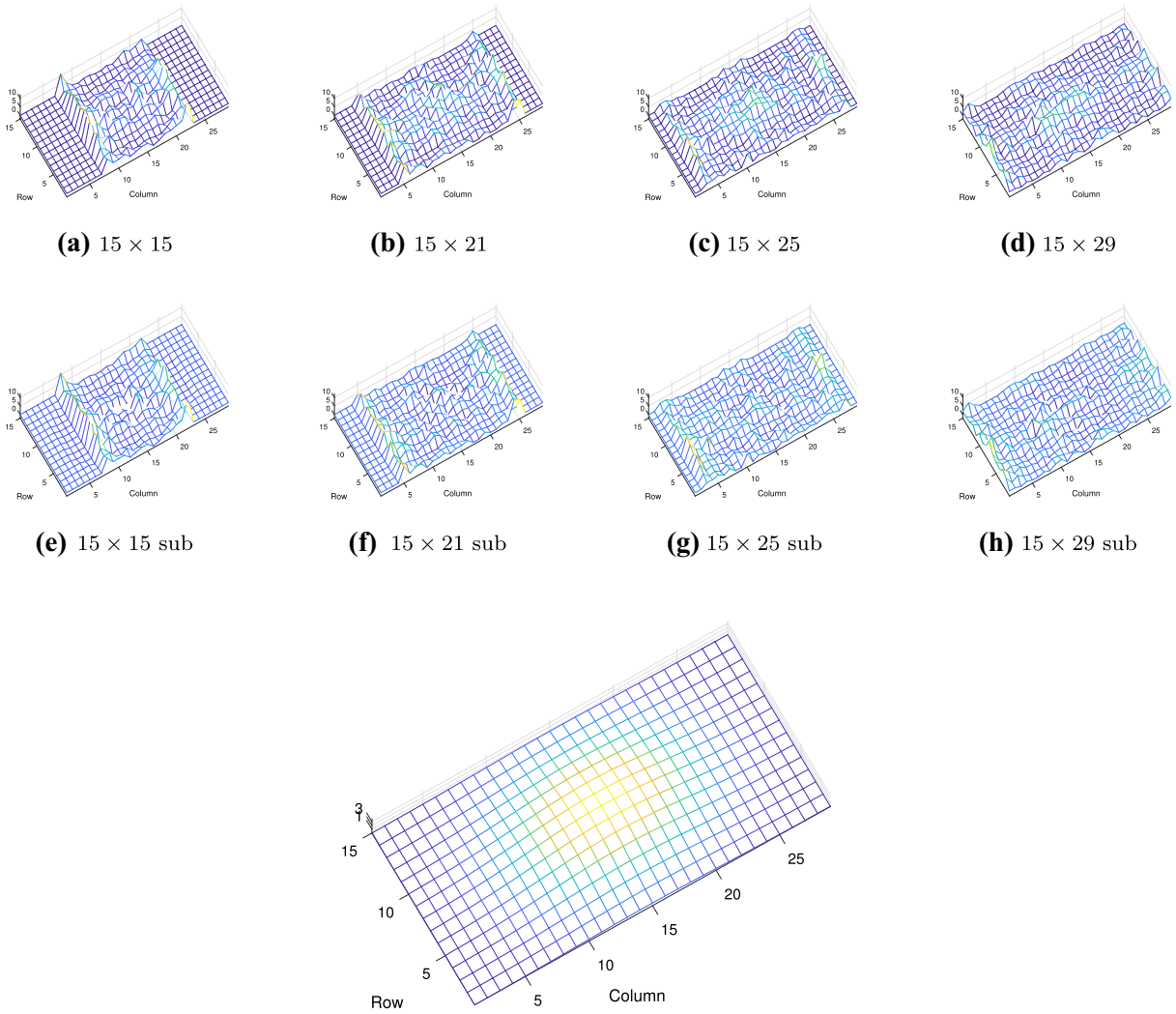


Fig. 13 Lateral windows: Histograms of coordinate activation and edge length, 30 runs, 15 × 29 neighborhood, 24 edges

Opposed to the original CT, for GACT, the window size can be altered without affecting the output data width, as the number of edges is defined. Hence, increasing the window size is a valid option to achieve higher matching accuracy. However, a larger window requires more buffering resources in the GACT component. On the FPGA, where the image is represented as a stream, instead of a two-dimensional grid, the concept of neighboring pixels/pixel connectivity, is redefined. For a stream, the distance to horizontal neighbors are one pixel, just as in the 'normal' case. Vertical neighbors, on the other hand, are one full width of the image away, and require buffering of a full row. This is handled by the sliding window component using block ram. Elongating the GACT window in the horizontal direction comes at a very low additional cost, while extending the window vertically



(a) 15×15 **(b)** 15×21 **(c)** 15×25 **(d)** 15×29

(e) 15×15 sub **(f)** 15×21 sub **(g)** 15×25 sub **(h)** 15×29 sub

(i) Central distribution

Fig. 14 First row: Coordinate activation histograms for different GACT widths. Second row: Histograms with central distribution subtracted. Third row: Estimated central normal probability distribution

Table 5 FPGA utilization GACT 21×21

Resource	Utilization	Available	Utilization %
LUT	212,634	230,400	92.29
LUTRAM	545	101,760	0.54
FF	97,500	460,800	21.15
BRAM	52.5	312	16.83
BUFG	7	544	1.29

Table 6 FPGA utilization GACT 15×29

Resource	Utilization	Available	Utilization %
LUT	214,889	230,400	93.27
LUTRAM	545	101,760	0.54
FF	97,807	460,800	21.23
BRAM	46.5	312	14.90
BUFG	7	544	1.29

consumes block ram resources. This can be seen in Table 7 for the default, 5×29, 29×5 and 5×5 setups, where 29×5 is the most expensive. 5×29 and 5×5 on the other hand show similar utilization figures. Also, the evaluation results for 5×29 and 29×5 are 8.90% vs 12.40% for non-occluded pixels—a substantial difference. Similarly, window height has a great impact on latency, as it requires readout of a full row, while lateral change can be counted in terms of

Table 7 FPGA utilization (%) for different settings

Resource	default[1]	5 × 29	29 × 5	5 × 5	8 edges	128 disparites
LUT	44.30	44.10	44.45	44.23	20.58	23.65
LUTRAM	0.54	0.54	0.54	0.54	0.54	0.54
FF	19.78	19.68	19.74	19.69	11.01	10.94
BRAM	14.90	11.68	19.39	11.70	9.76	14.90
BUFG	0.92	0.92	0.92	0.92	0.92	0.92

[1] default parameters: pre-defined GACT mask, 15 × 29 census window, 24 edges, 256 disparities

individual clock cycles. It can be concluded that lateral windows are resource efficient and perform better.

Limiting the number of edges for GACT will of course save resources, not only in the GACT component itself, where fewer pixels are accessed, but more importantly in the subsequent stereo matching component, which no longer has to support the full data width. This is apparent when comparing the FPGA utilization for 8 to 24 edges, as can be seen in Table 7. Finally, the supported disparity range of the circuit is a major contributor to high implementation cost. The disparity range is dictated by the application/problem and not a variable parameter as such. However, for the KITTI training dataset 0.0022% of the pixels are of disparity larger than 127. Assuming that the evaluation dataset has the same disparity distribution, limiting the disparity range to 127 can be considered a fair trade off (to save resources for more elaborate stereo matching). From the subset of images randomly selected for GACT training there are no disparities greater than 127 so for the current training setup there would be no penalty associated with a disparity range reduction. The FPGA utilization for 128 disparities is listed in Table 7.

The FPGA pipeline is clocked at 50 MHz. As 256 disparity hypotheses are evaluated in parallel this equates to 12,800 MDE/s. KITTI images are of 0.5 Mpixel, hence the frame rate is 100 fps. At this rate a training cycle, for the current set of parameters, would complete in 5 min. However, the current SoC setup requires approximately 40 min (2325 s), with a single core CPU load of 18.5%. Consequently the bottleneck of the system is believed to adhere to memory mapping of image data on a driver level. Regardless, the hardware acceleration is considerate, compared to previous work [2], where a high-level CPU implementation required approximately 20 h to complete a training cycle, using an Intel Xeon X5650 2.67 GHz, even though the 8 offsprings were calculated in parallel through multi-core processing. Hence, the SoC setup accelerates training by a factor of 30.

5.7 Evaluation with respect to related work

GACT has been compared to related works of a 5 × 5 census window size, except for CT 7 × 9. Several of the related works are sparse and should be compared to methods of similar number of edges. To include the aspect of rectangular windows, additional GACT masks were included for 3 × 9 (27 pixels) and 3 × 7 neighborhoods (21 pixels). GACT24 5 × 29 has also been appended as a reference, as it has been established that the implementation cost is similar to GACT24 5 × 5. The results are shown in Table 8.

Table 8 KITTI error rates–% bad pixels, end-point error < 3*px* or < 5%

Method	NOC	OCC	Edges
GACT24 3 × 9	13.58	15.03	24[1]
GACT16 3 × 9	13.85	15.30	16[1]
GACT8 3 × 9	14.60	16.03	8[1]
GACT24 3 × 7	14.99	16.42	24[2]
GACT16 3 × 7	15.17	16.59	16[2]
GACT24 5 × 5	15.37	16.80	24
GACT16 5 × 5	15.59	17.01	16
CT 7 × 9	15.80	17.22	62[3]
GACT8 3 × 7	15.84	17.25	8[2]
GACT8 5 × 5	16.40	17.80	8
GCT16	17.66	19.04	16
GCT8	17.81	19.20	8
hwCSCT	18.08	19.46	14
CSCT	18.16	19.54	12
wCSCT	18.25	19.63	16
QCT	18.50	19.87	48[4]
SCT16-2-2	18.83	20.19	16
MeanCT	18.96	20.33	24[5]
ACT	19.43	20.77	24[6]
SCT24-2-1	19.78	21.12	24
SCT8-2-2	20.09	21.43	8
Sparse8	20.22	21.55	8
CT	20.60	21.92	24
Sparse12	20.64	21.96	12
RCT	23.20	24.48	8
MCT	24.11	25.38	6
GACT24 5 × 29	9.09	10.54	24*
GACT24 3 × 29	9.44	10.88	24**

[1] 27 pixel census window
[2] 21 pixel census window
[3] 63 pixel census window
[4] bit strings of length 48 and requires mean value calculation
[5] requires mean value calculation
[6] requires weight and SAD calculation
* 145 pixel census window, but similar FPGA utilization as GACT24 5 × 5
** 87 pixel census window, but similar FPGA utilization as GACT24 3 × 7 and 3 × 9

The results show that the 5×5 GACT performs better than other CT methods of the same size and number of edges. In fact, only CT 7×9, which comes associated with considerably higher resource costs, achieves a better score than the sparsest GACT. GACT 5×5 is comparable with previous results [2], which corroborates the convergence of GACT from a small training set for homogeneous datasets.

In line with the experimental results, the accuracy is improved by adopting rectangular census windows. Both GACT 3×7 and 3×9 perform better than their quadratic counterpart. GACT 3×7 is of a slightly smaller neighborhood, while GACT 3×9 is slightly larger. It is evident that GACT makes good use of the extra lateral columns, and for this setup it requires the same amount of resources, which in terms of row buffering, is half to GACT 5×5. If resources are the focus, and 5×5 is considered the default experiment, GACT 5×29 comes with no, or low, additional cost, for the current setup. However, the result supersedes the smaller windows, by a margin.

To perform well it is apparent that a 5×5 census window is not enough. Looking at the related works, only MCT, RCT, GCT and SCT are defined to size. CT is often adapted in a 7×9 configuration, for bit-strings to fit within 64-bit registers. However, bit-string length quickly increases with window size, and hence also the processing cost for matching. The sparse CT can produce bit-strings of a specific length, for different window sizes, by adapting different levels of sparseness. The main argument of sparse CT was that given an equal number of comparisons, a larger sparse CT performs better than a larger dense, which is preferable if/when the processing resources are limited.

Center based CTs are sensitive to noise and methods using different comparison schema, GCT, SCT and CSCT, have been proven successful, and are the ones to improve upon. Both GCT and SCT are defined within a 5×5 neighborhood and of a certain number of edges. For SCT there are also different variations depending on edge length. To perform a proper extension for windows of different sizes and shape would require quite an effort as the methods are handcrafted. The simple solution would be extended into larger windows by introducing empty rows and columns, analogous to sparse CT. However, this raises concerns regarding edge distribution and length. SCT could capture local lateral comparisons but is limited to one single edge length. GCT has edges spanning the entire neighborhood and these edges are not favorable according to GACT.

GACT on the other hand can produce bit-strings of a specified length, independent of window size. It will also find a good distribution between the edges, both regarding positioning and length. However, this comes at the cost of training. By adopting the proposed hardware accelerated approach, training is quick and does not require much training data.

Table 9 Block matching vs SGM

	Block matching		SGM	
Method	NOC	OCC	NOC	OCC
CT 5×5	20.60	21.92	15.82	17.23
GACT24 5×5	15.37	16.80	12.73	14.20
GACT24 3×9	13.58	15.03	11.48	12.97
GACT24 5×29	9.09	10.54	8.77	10.22

5.8 Outlook

The experiments and evaluations have been performed using a basic block-matching framework. This gives a base for comparison between different CT methods, as cost metrics, but is not a full and final stereo algorithm where concepts such as matching confidence, left-right consistency, different strategies for cost aggregation, refinement filters, sub-pixel interpolation, etc., are considered. Both algorithm optimization steps and adaptations for FPGA implementation has effect on matching accuracy [43]. The first question, of course, is how well GACT can perform in such an algorithm, and secondly, if and how extending the algorithm affects GACT training result. Is the edge distribution depending more on image information or algorithm. These questions are left for future works. However, as a small experiment, SGM [15] was adopted, for some of the GACT mask from the experiments, i.e. trained using basic block matching. The results are shown in Table 9.

It is clear that SGM optimization improves the result. For larger GACT windows, though, the improvement is small. Two reasons for this are that 1) the lower the error rate, the more challenging and costly to make improvements, (similarly to larger window sizes and more edges), and more to the point 2) larger windows results in a larger perceptive field, including 'semi-local' information otherwise provided by SGM. However, the question whether GACT and SGM share a symbiotic advantage, if employed during training, remains.

6 Conclusion

The CT is a well-established cost metric for stereo matching suitable for implementation on resource limited systems. Over the years several different CT methods have been proposed, from which two key developments can be identified: (1) sparse CTs save resources by not evaluating all pixels within the census window, and a larger sparse CT performs better than a smaller dense, making for a similar implementation cost come the actual matching and (2) non-centric comparison schemas make CT produce a better result and be less sensitive to noise. The GACT takes advantage of

both these developments, but instead of using a handcrafted comparison schema relies on GA to position the edges, optimized for the image data. Previous work [2] shows that GACT performs better than other CT methods with the same number of edges. In this paper the training time for GACT has been significantly reduced through hardware acceleration, adopting FPGA-based GACT and stereo matching. This has enabled evaluation of GACT for multiple parameter sets, altering window size and shape, and the number of edges. The experiments suggest that GACT has a preference for selecting two different types of edges, central and lateral, of limited length compared to the larger neighborhood max length. Hence, GACT benefits from adopting lateral windows, further improving the previously established GACT result, while at the same time, from an implementation perspective, requiring less buffer resources.

Acknowledgements Open access funding provided by Mälardalen University. The research leading to the presented results has been undertaken within the research profile DPAC (http://www.es.mdh.se/dpac/)—Dependable Platform for Autonomous Systems and Control project, funded by the Swedish Knowledge Foundation, and the ECSEL-JU project SWARMs (www.swarms.eu).

References

1. Ahlberg, C., Ekstrand, F., Ekstrom, M., Spampinato, G., Asplund, L.: Gimme2 - an embedded system for stereo vision and processing of megapixel images with fpga-acceleration. In: 2015 International conference on ReConFigurable computing and FPGAs (ReConFig), pp. 1–8 (2015). https://doi.org/10.1109/ReConFig.2015.7393318
2. Ahlberg, C., Ortiz, M.L., Ekstrand, F., Ekstrom, M.: Unbounded sparse census transform using genetic algorithm. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1616–1625 (2019). https://doi.org/10.1109/WACV.2019.00177
3. Ambrosch, K., Kubinger, W.: Accurate hardware-based stereo vision. Comput. Vis. Image Und. **114**(11), 1303–1316 (2010). https://doi.org/10.1016/j.cviu.2010.07.008
4. Ambrosch, K., Zinner, C., Leopold, H.: A miniature embedded stereo vision system for automotive applications. In: 2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel, pp. 000786–000789 (2010). https://doi.org/10.1109/EEEI.2010.5662105
5. Chang, J., Chen, Y.: Pyramid stereo matching network. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5410–5418 (2018). https://doi.org/10.1109/CVPR.2018.00567
6. Chang, N.Y., Tsai, T., Hsu, B., Chen, Y., Chang, T.: Algorithm and architecture of disparity estimation with mini-census adaptive support weight. IEEE Trans. Circ. Syst. Video Technol. **20**(6), 792–805 (2010). https://doi.org/10.1109/TCSVT.2010.2045814
7. Cocorullo, G., Corsonello, P., Frustaci, F., Perri, S.: An effcient hardware-oriented stereo matching algorithm. Microprocess. Microsyst. **46**, 21–33 (2016). https://doi.org/10.1016/j.micpro.2016.09.010
8. Dixon, S.L., Koehler, R.T.: The hidden component of size in two-dimensional fragment descriptors: Side effects on sampling in bioactive libraries. J. Medi. Chem. **42**(15), 2887–2900 (1999). https://doi.org/10.1021/jm980708c
9. Eiben, A.E., Raué, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: International Conference on Parallel Problem Solving from Nature, pp. 78–87 (1994)
10. Fife, W.S., Archibald, J.K.: Improved census transforms for resource-optimized stereo vision. IEEE Trans. Circ. Syst. Video Technol. **23**(1), 60–73 (2013). https://doi.org/10.1109/TCSVT.2012.2203197
11. Froba, B., Ernst, A.: Face detection with the modifed census transform. In: Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings., pp. 91–96 (2004). https://doi.org/10.1109/AFGR.2004.1301514
12. Goldberg, D.: Genetic algorithm in search, optimization and machine learning. Addison-Wesley, New York (1989)
13. Gong, M., Yang, Y.H.: Multi-resolution stereo matching using genetic algorithm. In: Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001), pp. 21–29 (2001). https://doi.org/10.1109/SMBV.2001.988759
14. Herrera, F., Lozano, M., Perez, E., Sanchez, A., Villar, P.: Multiple crossover per couple with selection of the two best offspring: an experimental study with the blx-alpha crossover operator for real-coded genetic algorithms. In: Advances in Artifcial Intelligence-IBERAMIA, pp. 392–401 (2002)
15. Hirschmuller, H.: Stereo processing by semiglobal matching and mutual information. IEEE Trans. Pattern Anal. Mach. Intell. **30**(2), 328–341 (2008). https://doi.org/10.1109/TPAMI.2007.1166
16. Hirschmuller, H., Scharstein, D.: Evaluation of stereo matching costs on images with radiometric differences. IEEE Trans. Pattern Anal. Mach. Intell. **31**(9), 1582–1599 (2009). https://doi.org/10.1109/TPAMI.2008.221
17. Holland, J.H.: Adaptation in Natural and Artifcial Systems. The University of Michigan Press (1975)
18. Hosni, A., Bleyer, M., Rhemann, C., Gelautz, M., Rother, C.: Real-time local stereo matching using guided image filtering. In: 2011 IEEE International Conference on Multimedia and Expo, pp. 1–6 (2011). https://doi.org/10.1109/ICME.2011.6012131
19. Humenberger, M., Zinner, C., Weber, M., Kubinger, W., Vincze, M.: A fast stereo matching algorithm suitable for embedded real-time systems. Computer Vision and Image Understanding **114**, 1180–1202 (2010). https://doi.org/10.1016/j.cviu.2010.03.012
20. Ko, J., Ho, Y.: Stereo matching using census transform of adaptive window sizes with gradient images. In: 2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), pp. 1–4 (2016). https://doi.org/10.1109/APSIPA.2016.7820827
21. Yoon, Kuk-Jin: In So Kweon: Adaptive support-weight approach for correspondence search. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(4), 650–656 (2006). https://doi.org/10.1109/TPAMI.2006.70

22. Kuzmin, A., Mikushin, D., Lempitsky, V.: End-to-end learning of cost-volume aggregation for real-time dense stereo. In: 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1–6 (2017). https://doi.org/10.1109/MLSP.2017.8168183
23. Lee, J., Jun, D., Eem, C., Hong, H.: Improved census transform for noise robust stereo matching. Optical Engineering **55**(6), 1–10 (2016). https://doi.org/10.1117/1.OE.55.6.063107
24. Loghman, M., Kim, J.: Sgm-based dense disparity estimation using adaptive census transform. In: 2013 International Conference on Connected Vehicles and Expo (ICCVE), pp. 592–597 (2013). https://doi.org/10.1109/ICCVE.2013.6799860
25. Luo, W., Schwing, A.G., Urtasun, R.: Efficient deep learning for stereo matching. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5695–5703 (2016). https://doi.org/10.1109/CVPR.2016.614
26. Ma, L., Li, J., Ma, J., Zhang, H.: A modified census transform based on the neighborhood information for stereo matching algorithm. In: 2013 Seventh International Conference on Image and Graphics, pp. 533–538 (2013). https://doi.org/10.1109/ICIG.2013.113
27. Madureira, A., Ramos, C., do Carmo Silva, S.: A coordination mechanism for real world scheduling problems using genetic algorithms. In: IEEE Congress on Evolutionary Computation, pp. 175–180 (2002)
28. Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., Brox, T.: A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4040–4048 (2016). https://doi.org/10.1109/CVPR.2016.438
29. Mei, X., Sun, X., Zhou, M., Jiao, S., Wang, H., Xiaopeng Zhang: On building an accurate stereo matching system on graphics hardware. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp. 467–474 (2011). https://doi.org/10.1109/ICCVW.2011.6130280
30. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3061–3070 (2015). https://doi.org/10.1109/CVPR.2015.7298925
31. Michalik, S., Michalik, S., Naghmouchi, J., Berekovic, M.: Real-time smart stereo camera based on fpga-soc. In: 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), pp. 311–317 (2017). https://doi.org/10.1109/HUMANOIDS.2017.8246891
32. Miguel, L., Ballesteros, J., Tidare, J., Xiong, N., Astrand, E.: Feature Selection of EEG Oscillatory Activity Related to Motor Imagery Using a Hierarchical Genetic Algorithm. In: IEEE Congress on Evolutionary Computation, p. 8. Wellington, New Zealand (2019)
33. Nguyen, V.D., Nguyen, D.D., Nguyen, T.T., Dinh, V.Q., Jeon, J.W.: Support local pattern and its application to disparity improvement and texture classification. IEEE Trans. Circ. Syst. Video Technol. **24**(2), 263–276 (2014). https://doi.org/10.1109/TCSVT.2013.2254898
34. Perri, S., Corsonello, P., Cocorullo, G.: Adaptive census transform: a novel hardware-oriented stereovision algorithm. Comput Vis Image Understand **117**(1), 29–41 (2013). https://doi.org/10.1016/j.cviu.2012.10.003
35. Perri, S., Frustaci, F., Spagnolo, F., Corsonello, P.: Design of real-time fpga-based embedded system for stereo vision. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2018). https://doi.org/10.1109/ISCAS.2018.8351886
36. Qiao, S., Yang, J., Meng, L., Yan, S.: Hardware implementation of census stereo matching algorithm. In: 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), pp. 1–3 (2019). https://doi.org/10.1109/EDSSC.2019.8753990
37. Rogers, D.J., Tanimoto, T.T.: A computer program for classifying plants. Science **132**(3434), 1115–1118 (1960). https://doi.org/10.1126/science.132.3434.1115
38. Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X., Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth. In: Pattern Recognition - 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings, pp. 31–42 (2014). https://doi.org/10.1007/978-3-319-11752-2_3
39. Scharstein, D., Szeliski, R., Zabih, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001), pp. 131–140 (2001). https://doi.org/10.1109/SMBV.2001.988771
40. Spangenberg, R., Langner, T., Rojas, R.: Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In: Wilson, R., Hancock, E., Bors, A., Smith, W. (eds.) Computer Analysis of Images and Patterns, pp. 34–41. Springer, Berlin (2013)
41. Taniai, T., Matsushita, Y., Sato, Y., Naemura, T.: Continuous 3d label stereo matching using local expansion moves. IEEE Trans. Pattern Anal. Mach. Intell. **40**(11), 2725–2739 (2018). https://doi.org/10.1109/TPAMI.2017.2766072
42. Tavera-Vaca, C.A., Almanza-Ojeda, D.L., Ibarra-Manzano, M.A.: Analysis of the efficiency of the census transform algorithm implemented on fpga. Microprocess. Microsyst. **39**(7), 494–503 (2015). https://doi.org/10.1016/j.micpro.2015.08.002
43. Wang, W., Yan, J., Xu, N., Wang, Y., Hsu, F.: Real-time high-quality stereo vision system in fpga. IEEE Transactions on Circuits and Systems for Video Technology **25**(10), 1696–1708 (2015). https://doi.org/10.1109/TCSVT.2015.2397196
44. Xiong, N., Molina, D., Leon, M., Herrera, F.: A walk into metaheuristics for engineering optimization: principles, methods, and recent trends. Int. J. Comput. Intell. Syst. **8**(4), 606–636 (2015)
45. Zabih, R., Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In: Proceedings of the Third European Conference on Computer Vision (Vol. II), ECCV '94, pp. 151–158. Springer-Verlag, Berlin, Heidelberg (1994)
46. Zbontar, J., LeCun, Y.: Computing the stereo matching cost with a convolutional neural network. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1592–1599 (2015). https://doi.org/10.1109/CVPR.2015.7298767
47. Žbontar, J., LeCun, Y.: Stereo matching by training a convolutional neural network to compare image patches. J. Mach. Learn. Res. **17**(65), 1–32 (2016). URL http://jmlr.org/papers/v17/15-535.html
48. Zhang, F., Prisacariu, V., Yang, R., Torr, P.H.: Ga-net: Guided aggregation net for end-to-end stereo matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 185–194 (2019)

Carl Ahlberg received his M.Sc. in Information Technologies from Uppsala University, Sweden, in 2003. In 2009 he returned to academia and joined Mälardalen University, Sweden, from where he received his Ph.D. in 2020. His research interests include robotics, machine vision and stereo vision.

Miguel Leon received the M.S. degree in Computer Science from Granada University, Spain in 2013, and the Ph.D. degree in the School of Innovation, Design, and Engineering, Mälardalen University, Sweden in 2019. His research interests include evolutionary algorithms, differential evolution, and applications of evolutionary algorithms.

Fredrik Ekstrand received his Tech.Lic. in Electronics from Mälardalen University, Västerås, Sweden, in 2013 where he currently heads the division of Intelligent Future Technologies. His research interests include machine vision, FPGAs and robotics.

Mikael Ekström is a professor in robotics at Mälardalen University, Västerås, Sweden, where he is the head of the robotics research group. He received his M.Sc. and Ph.D. in physics from Uppsala University, Sweden in 1993 and 1999 respectively. His research interests include robotics, autonomous vehicles, sensors, and communication.