

Evolving aggregation windows for disparity estimation

Paul Adriani, 5698707
padriani@science.uva.nl

David de Bos, 0382930
dbos@science.uva.nl

Mark Kroon, 0306185
mkroon@science.uva.nl

Spiros Koulouzis, 0641707
skoulouz@science.uva.nl

January 31, 2008

Abstract

A genetic algorithm is introduced as an instrument to evolve a population of aggregation windows to create optimized Disparity Space Images. This is a new method of selecting window size and shape for stereo matching that is competitive with existing methods such as adaptive window creation and multiple support window selection with fixed size. Our method, which selects windows of different size and shape, was compared to the multiple window selection method. We found that the genetic algorithm created an aggregation window set of which the performance on disparity estimation is comparable to and in some cases stronger than the multiple support window method. Our experiment proves that genetic algorithms are in principle a valuable technique for the selection of aggregation window sets. Future research could have great benefit of using this type of algorithm.

1 Introduction

Stereo image techniques have the potential to be used in many applications like autonomous vehicles and video conferencing. For vehicles to drive autonomously computer vision is required to analyze the terrain. Currently laser range-finder devices are most successful to do this task in real-time [13]. The disadvantage of these laser devices is their limited range (approximately 22 meter) and that they are very expensive. Stereo vision cameras can offer a nice and cheap alternative. One of the major challenges in stereo vision is the matching of stereo images, which is commonly referred to as the correspondence problem. Given a left and right image the goal is to match pixels from one image with the corresponding pixels in the other image. The amount that each pixel is shifted is called the disparity (see Figure 1). Those disparity values are closely related to the depth of an image and can be used for example for obstacle detection.

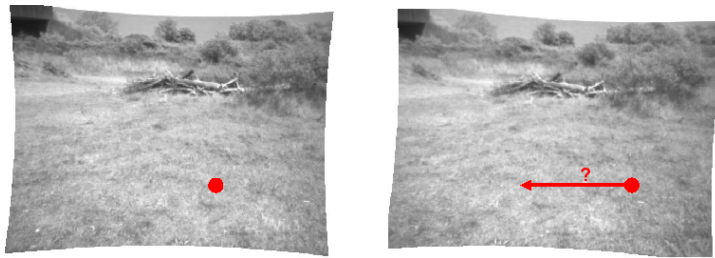


Figure 1: Disparity estimation: the correspondence problem

The structure of the article is as follows. We start with a section about the standard techniques for disparity estimation, followed by a detailed discussion of the genetic algorithm. Section 3 describes the experiments done and discusses the results. We end the paper with a conclusion and proposals for further research.

1.1 Related work

1.1.1 Taxonomy of stereo matching algorithms

In [12] a taxonomy of stereo matching algorithms is described. They distinguish four main steps in stereo algorithms. For this project we extend the taxonomy with a preprocessing step.

- 1 filter input images
- 2 Computation of the matching costs
- 3 Cost (support) aggregation
- 4 Disparity computation / optimization
- 5 Disparity refinement

The first step is a preprocessing step. In this step a filter is applied to the image pairs. Three filters are most commonly used. The Laplacian of Gaussians, [10], the rank and the Census filter, [11]. These filters reduce the influence of differing light intensities caused by the difference in the position of the left and the right stereo camera with respect to the light source.

In the second step so-called matching costs are computed for each pixel individually for each disparity. A Disparity Space Image (DSI) is created, which is a $X \times Y \times D$ matrix containing the disparity costs of all pixels of the reference image. Figure 3. Common methods to calculate those costs are absolute intensity differences (AD), squared intensity differences (SD) and normalized cross-correlation (NCC). Absolute intensity differences per pixel at position (x,y) per disparity d is computed by subtracting the pixel



Figure 2: left: original image, right: rank filter applied to image

value $pl_{x,y}$ of the left image from the pixel value $pr_{x,y}$ of the right image while moving the right image 1 pixel to the left for each disparity.

$$AD_{x,y,d} = |pl_{x,y} - pr_{x,y}|$$

$$SD_{x,y,d} = (pl_{x,y} - pr_{x,y})^2$$

In the cost aggregation step the matching costs for each pixel are aggregated by the costs of the pixels in the surrounding region to increase reliability. The size and shape of this region is determined by aggregation windows. Some methods use one fixed window, for example a 3x3 window, Figure 4, others combine multiple windows with fixed size. The aggregated disparity estimation $ac_{x,y,d}$ is computed by taking the sum of the costs of each pixel in the window. This sum is normalized by the number of pixels in the window n .

$$ac_{x,y,d} = \frac{1}{N} \sum_{x-1}^{x+1} \sum_{y-1}^{y+1} c_{x,y,d}$$

In the fourth step the optimal disparity is chosen for each pixel. Many different approaches can be taken here, ranging from a simple Winner Takes All (WTA) approach that just selects the disparity with minimal cost, to

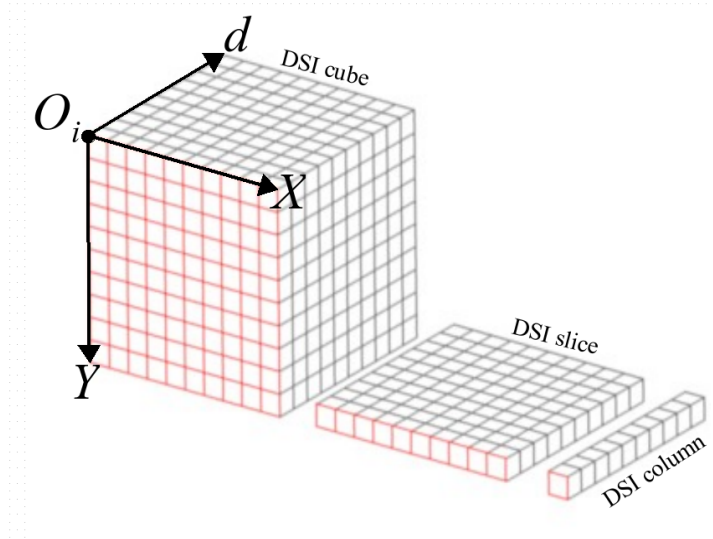


Figure 3: Disparity Space Image

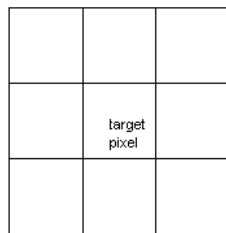


Figure 4: 3 by 3 aggregation window

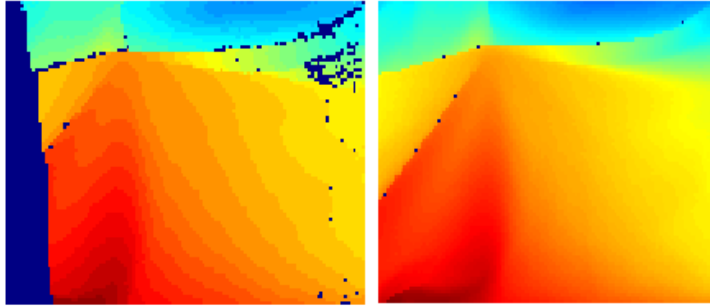


Figure 5: left: visualization of the estimated disparity, right: ground truth

more advanced methods such as graph-cut and dynamic programming [12]. The optimal disparity using the WTA approach is the minimum estimated cost value $mc_{x,y}$ of each disparity column, Figure 3.

$$mc_{x,y} = \min(ac_{x,y})$$

In the last step the resulting disparity map is refined. A common used refinement technique is sub-pixel disparity estimation; this gives a smoother disparity map. A local technique, which is commonly used, is a median-filter which can reduce noise, possibly caused by occlusions. In this experiment we used a left/right consistency check to search for errors [2]. This technique is an effective means for identifying occlusion. After refinement the disparity estimation can be visualized and compared to a ground truth, Figure 5.

In this paper we present a new method for determining the size and shape of aggregation windows. For the other steps we use one of the existing techniques as will be explained in section 2.

1.1.2 The size and shape of aggregation windows

One of the challenges in disparity estimation is handling depth discontinuities, which are found at the border of a foreground and background object. On the one hand, using a window which is too large at these depth discontinuities will introduce errors in the aggregation step as can be seen in Figure 6. On the other hand, the usage of a window which is too small will reduce the reliability. Therefore, it is proposed in [5] to use multiple supporting windows to adapt the size of the window for each pixel. Their general approach is to take a central window and a number of surrounding (non-overlapping or overlapping) windows in an x by x grid, and combine the costs of the central window with the costs of the four best surrounding windows. Although the nine windows are all rectangular and all have the same size, which is a parameter to be set beforehand, the resulting combined window can be of many different shapes.

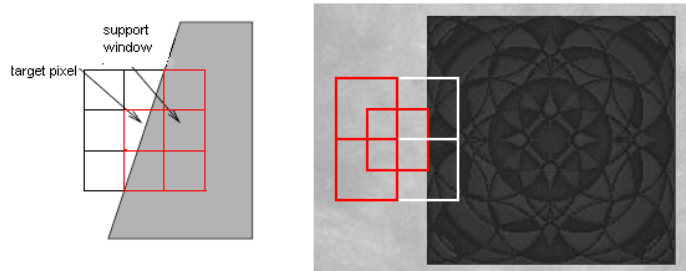


Figure 6: multiple support windows: overlapping(left), non-overlapping(right)

A different approach is adaptive window selection [8]. This method creates aggregation windows for each pixel separately. The idea of the method is that optimal stereo matching depends on the tradeoff between small and large windows. On the one hand, windows need distinctive strength of the cost value (SSD) relative to noise to identify the minimum disparity correctly. On the other hand, windows need small variation in disparity in the window to be able to detect the disparity of different positions in the image. The search of the environment of each pixel individually makes it possible to adapt a window into optimal shape according to a threshold function which is based on the tradeoff. However, creating the windows separately for each pixel is computationally challenging and takes a great amount of processing time. Therefore, it cannot be used for real-time stereo vision.

1.2 Evolving aggregation windows

As the results in [5] show, it is useful to select a window shape for each pixel that introduces a minimum amount of error. In this paper we present a method which can be positioned between the existing methods. Where Hirschmueller et al. only select a combination of 5 out of 9 windows of the same size during the disparity estimation process and Okutomi et al. create a totally new windows for each pixel separately, we use a combination of 9 windows of different shape and size. The shape and size of these 9 windows have been evolved by a genetic algorithm. We expect to find a window set which can be used for disparity estimation in all environments.

There are several reasons which argue towards using a genetic algorithm. First of all, we have no a priori knowledge about the form of the windows that will work best for disparity estimation. In some cases a circular window might work good, but in other cases a horizontally or vertically oriented window might perform better. This makes the total amount of window-forms, i.e. the hypothesis space of the optimal window set, enormous. The computational strength of a genetic algorithm applied to a cluster, see appendix,

makes it possible evaluate part of this hypothesis space. Furthermore, mutation operators which evolve window-shapes can be defined in detail. All these aspects make a genetic algorithm a very suitable instrument for the problem to be solved. This experiment aims at proving that the principles of genetic algorithms form a good framework which is suitable for these kind of tasks. Our experiments show that the genetic algorithm is a strong instrument for optimizing the process of disparity estimation.

2 Methods

2.1 Disparity estimation

We perform disparity estimation by following the steps in the taxonomy of [12] plus a preprocessing step. At each step we selected a technique for implementation. This section explains these implementation choices. The design of the algorithm is based on two aims. Firstly, the algorithm has to estimate disparity as good as or better than existing methods and the algorithm must be fast enough to run-real time, for instance in autonomous vehicles.

In a preprocessing step we use a standard rank filter to focus on relative pixel intensities to take care of different illuminations among the stereo images. The rank filter was selected, because it is commonly known as the best filter.

The second step of the taxonomy is matching cost computation. In this step a DSI is created. We take the cost to be the absolute difference(AD) of pixel (x, y) with the corresponding pixel in the other image when considering disparity d . We take a maximum disparity D of 75. AD is a fast and precise measure for the computation the cost values.

In the third step the costs surrounding an entry in the DSI are aggregated by summing all costs at the same disparity that are specified by the aggregation window. In particular we use a set of 9 windows and sum the costs of the 3 windows with the minimal aggregation costs.

In the fourth step we use a WTA approach to find the optimal disparity value for each pixel in the reference image. The WTA approach is selected, because it is faster than the graph-cut approach or dynamic programming. Afterwards, in the refinement step, the left-/right check is performed on the DSI to detect occlusion.

2.2 Genetic algorithm

We try to maximize the disparity estimation accuracy by finding the optimal set of aggregation windows. Finding this set is an optimization problem which we solve with a basic Genetic Algorithm.

operator	p	description
crossover	1/7	select random windows from two members
add	1/21	add the windows of two members
subtract	1/21	subtract the windows of two members
dilate	1/7	perform a dilation on every window of a member
erode	1/7	perform a erosion on every window of a member
add noise	1/21	add random noise to the windows
subtract noise	1/21	subtract random noise from the windows
regenerate	1/21	create a new member by applying semi-random operations on a member
expansion	1/7	expand (or contract) the size of the windows
rotate	1/21	rotate the windows
quadrant swap	1/21	swap random quadrants of all windows
recreate	1/21	create a new semi-random window set

Table 1: Set of crossover and mutation methods applied in the genetic algorithm together with p , the probability of them to occur while reproducing.

Genetic Algorithms usually start with some initial population of candidate solutions and use operators from evolutionary biology to create new generations. The operators combine and randomly change the members of the population and thereby randomly search the solution space. The algorithm converges to a (local) optimum by allowing only the best performing (fittest) members to reproduce. The fitness of a member is defined by a fitness function that grows with the quality of the solution represented by the member.

In our case a member of a population is a set of aggregation windows. The fitness function gives a higher score to those members for which the disparity estimator performs well. We approximate the performance of the disparity estimator by applying it to a training set.

To reduce the size of the search space we initialize the population with semi-random window sets. An initial member has windows that are created by performing dilation and rotation operators on an arbitrary shape. With each generation the members with the largest fitness are selected from the population. This selection is allowed to reproduce with the mutation and crossover methods from Table 1. The selection is then copied together with these reproductions to the successive generation. The algorithm terminates when the growth of the largest fitness of the population stagnates over a series of generations.

2.3 Fitness function and error measure

The error of a density estimation is defined by its difference from the ground truth. The ground truth is part of the dataset presented in Section 3.1. The error of the disparity estimation of each pixel is divided by the true

disparity of that pixel, and so mistakes made on features further away are more influential. This is natural, because a small error in the estimation of a distant feature is a large error in terms of the distance of the feature.

We use a left/right consistency check [2] to detect occlusions. Only the pixels found consistent are considered when the average difference is computed. The error measure of a density estimator is then the average error of the estimations of the images in the dataset. To this end E_{rel} is determined for every image.

$$E_{rel} = \frac{1}{Q} \sum_m^M \sum_n^N (\delta_{m,n} \wedge d_{m,n}^* > 0) \frac{|d_{m,n} - d_{m,n}^*|}{d_{m,n}^*}$$

$$\text{where } Q = \sum_m^M \sum_n^N (\delta_{m,n} \wedge d_{m,n}^* > 0)$$

Here, $d_{m,n}$ is the disparity estimate of pixel (m, n) and the corresponding ground truth is $d_{m,n}^*$. The binary variable $\delta_{m,n} \in \{0, 1\}$ indicates the consistency of the pixel at location (m, n) . The negative of the error measure is used as the fitness function in the genetic algorithm. However, if the error is the same for two members up to four decimals, the members are compared by their coverage. A fitter member corresponds with a disparity estimator that can make an estimate for more pixels.

3 Experiments and results

3.1 Experiment setup

We did two runs of the genetic algorithm. Experiment one (DOAS-50) used a population of 50 individuals and the fitness was evaluated on two random images from the Middlebury stereo dataset [6]. We let the algorithm run for 41 generations after which it terminated because there was no progression anymore in the last 10 generations. The second experiment (DOAS-100) used a population of 100 individuals and the fitness was evaluated on 4 random images. We ended this run after 11 generations prematurely because of time constraints and therefore didn't include it in the final evaluation.

We compared our best sets of windows with the 5 out of 9 method of Hirschmueller et al. [5]. Their method can be represented as an individual (see Figure 10) as well, with a small adjustment that at least the first (the center) window is used and then the 4 best surrounding windows are selected.

Evaluation of both methods (DOAS-50 and HIRSCH) was done on a testset of 19 randomly selected images from the Middlebury dataset, not including the images used for training. The error-measure used for evaluation is the one described in section 2.3 We also included the percentage of inconsistent estimates (found by the left/right consistency check) in the

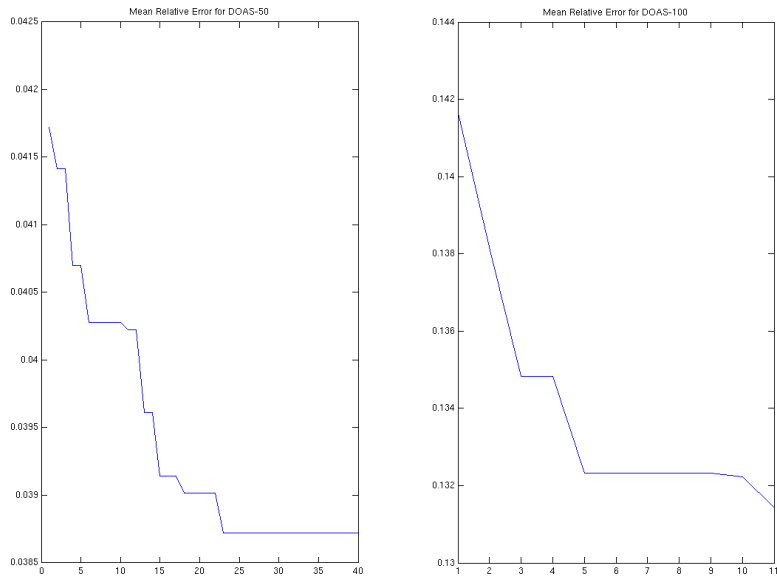


Figure 7: Mean Relative Error of fittest individual in each generation.

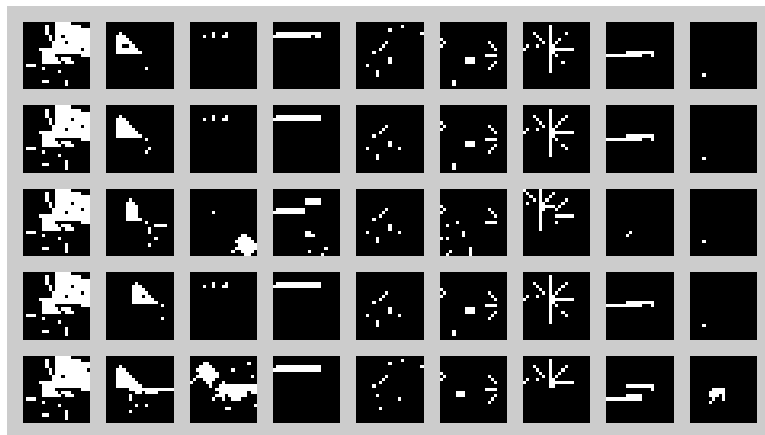


Figure 8: DOAS-50 - the best 5 individuals after 42 generations.

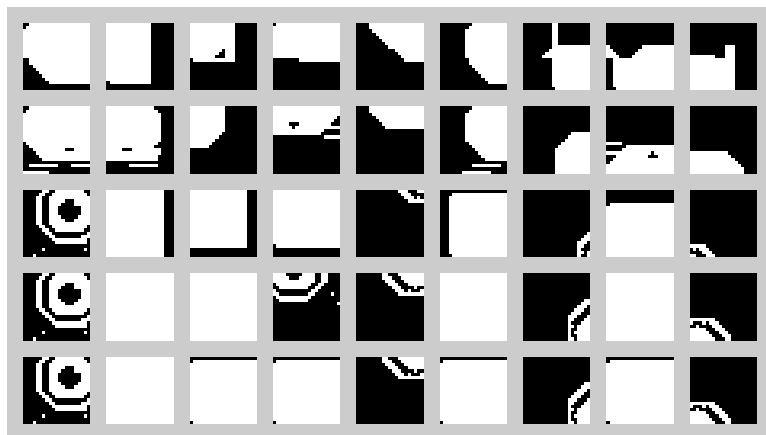


Figure 9: DOAS-100 - the best 5 individuals after 12 generations.



Figure 10: HIRSCH - selects 5 out of 9 windows with equal size.

evaluation because these are left out in the error-measure. Of course this percentage should be as low as possible.

3.2 Results

As mentioned above we performed two training runs for the genetic algorithm, the results of which are shown in Figure 7. Those two graphs indicate that the error from one generation to the next, is either stable, or decreases. Another observation of the two graphs indicates that using a population of 50 individuals (DOAS-50), we were able to obtain better results than when using a base population of 100 (DOAS-100), although this comparison would not be totally objective, since the DOAS-100 experiment was prematurely terminated, and only reached 11 generations, in contrast to the DOAS-50 reaching 40. Performing a comparison of the two methods (our method, and the Hirschmueller) over 19 images, we see that in the majority of the images, our method performs the same, or better than the Hirschmueller method. More specifically Figure 11 shows that the mean relative error, for the first 8 images is better than the Hirschmueller. In the next 6 images, the two methods perform exactly the same, while in the remaining, the Hirschmueller method was able to obtain a smaller error. Finally Figures 12 and 13 present, the estimated disparity for the best and worst performing window set, respectively. The results shown in Figures 7 and 11, indicate that probably with a larger training data set, and with sufficient training

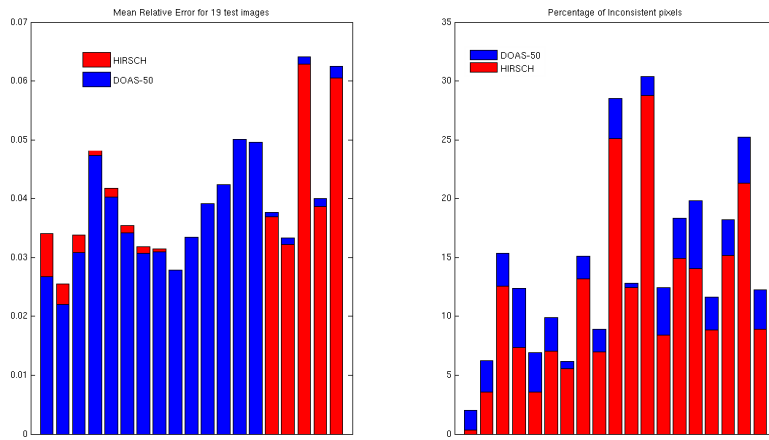


Figure 11: DOAS-50 compared to HIRSCH

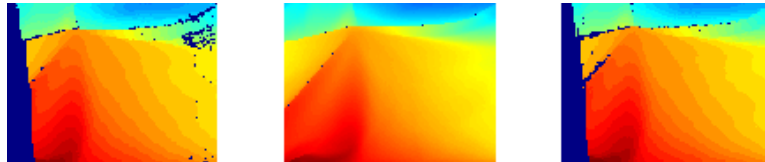


Figure 12: Estimated Disparity (from left to right DOAS-50, groundtruth, HIRSCH)

time our method, would be able to produce better results.

4 conclusion and discussion

It has been shown that disparity estimation is possible with a multiple aggregation window approach. A genetic algorithm has proven to be a good approach for finding a good window set.

The genetic algorithm searches nearly random among the possible windowsets to find the best set of 9 windows. The quality of the window set is measured with a disparity estimator that uses the 3 best windows to aggregate the cost from a DSI. The estimator will function well on the task of

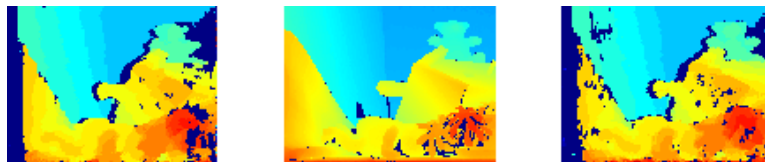


Figure 13: Estimated Disparity (from left to right DOAS-50, groundtruth, HIRSCH)

real-time processing of stereo images.

With the learned window set, the performance of the disparity estimator was equal to the performance of the method described in [5] which uses multiple support windows. Hence, we conclude that a prefixed window set evolved by a genetic algorithm is a competitive method for disparity estimation.

The configuration of the genetic algorithm was not optimal. The results might have been better if speciation was applied. Speciation allows inferior members to preserve functional structure in a separate population. It is also possible to find better results with multiple initial populations since genetic algorithms tend to converge to local optima. Hence, further research into the use of genetic algorithms in this paradigm could very well improve disparity estimators.

Using the current configuration allowed to find a window set that sometimes outperforms Hirschmüller et al. Furthermore, the approach allows a custom training set. This allows one to maximize the performance of disparity estimation in a particular domain. It would be interesting to use the window set produced by the method of Hirschmüller as a base population for the genetic algorithm. Future research could show that combining these methods would further increase the quality of Disparity Space Images.

References

- [1] J. Fernández, M. Anguita, E. Ros, and J.L. Bernier. See toolboxes for the development of high-level parallel applications. *Lecture Notes in Computer Science*, 3992:518–521, 2006.
- [2] P. Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6:35–49, 1993.
- [3] V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183, San Mateo, CA, 1993. Morgan Kaufman.
- [4] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, Cambridge, MA, USA, 1994.
- [5] H. Hirschmüller, P.R. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *Int. J. Comput. Vision*, 47(1-3):229–246, 2002.
- [6] H. Hirschmüller and D. Scharstein. Evaluation of cost functions for stereo matching. *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.
- [7] A. Muhammad, A. Bargiela, and G. King. Fine-grained parallel genetic algorithm: A stochastic optimisation method.
- [8] M. Okutomi and T. Kanade. A locally adaptive window for signal matching signal matching. *International Journal of Computer Vision*.
- [9] Cantú E. Paz. A survey of parallel genetic algorithms, 1997.
- [10] B. Kröse R. Bunschoten. Range estimation from a pair of omnidirectional images. *Robotics and Automation, Seoul, Korea, IEEE international Conference on*, 2000.

- [11] J. Woodfill R. Zabih. Non-parametric local transforms for computing visual correspondance. *Proceedings of the European Conference of Computer Vision*, 94:151–158, 1994.
- [12] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.
- [13] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. accepted for publication.
- [14] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence.

Appendix: Technical Report

Parallelizing Genetic Algorithms

A drawback that come with genetic algorithms is that they can be very slow. While an increased population size should give much better solutions, execution time increases dramatically. However genetic algorithms are well suited for parallelization which, while not removing significant computation, can improve much of the wait time for a solution [9].

Parallel implementations of genetic algorithms may be separated into two categories: Coarse grained and Fine grained. In Coarse grained parallel genetic algorithms, or the Island Model, each node executes the same GA, but with different populations. Thus these populations evolve spectrally, resulting in a more narrow view of the search space for each node. To overcome the segmentation of the search space, a migration step is induced in which periodically a subset of the local population belonging to each node migrates to its neighboring node. It is argued that this implementation of genetic algorithms could provide not just better execution times compared to sequential versions, but also better results [14].

Fine-grained or massively parallel, genetic algorithms assume the placement of an individual on each processing node. This individual can only mate with individuals located on the neighboring processing nodes. Depending on the topology of the network, there could be several versions of massively parallel genetic algorithms [3], [7].

There is a variety of tools and libraries, available for developing parallel programs. One of them is MPI (Message Passing Interface), which is a message-passing API (Application Programmer Interface). MPI has become the de facto standard for communication among processes that model a parallel program running on distributed memory systems, and because of that, implementations have been developed for Fortran, C/C++ and any language capable of interfacing with such routine libraries [4].

Parallel Implementation

In this research, the parallelization of the genetic algorithm was done using a kind of coarse grained parallelization and was implemented with the MPI Toolbox for Matlab (MPITB) [1]. However, the specifications of the project didn't allow for a full implementation of the coarse grained parallelization. Instead the focus was shifted in distributing the fitness evaluation.

The fitness evaluation is the most computationally expensive step in each iteration of the genetic algorithm. This is especially true as the population size gets bigger. This is shown in Figure 14, where clearly most of the time is spent on the fitness evaluation. A simple parallel genetic algorithm using a master-slave architecture is shown in Figure 15. The master maintains the population and sends individuals, or batches of individuals, to the slave

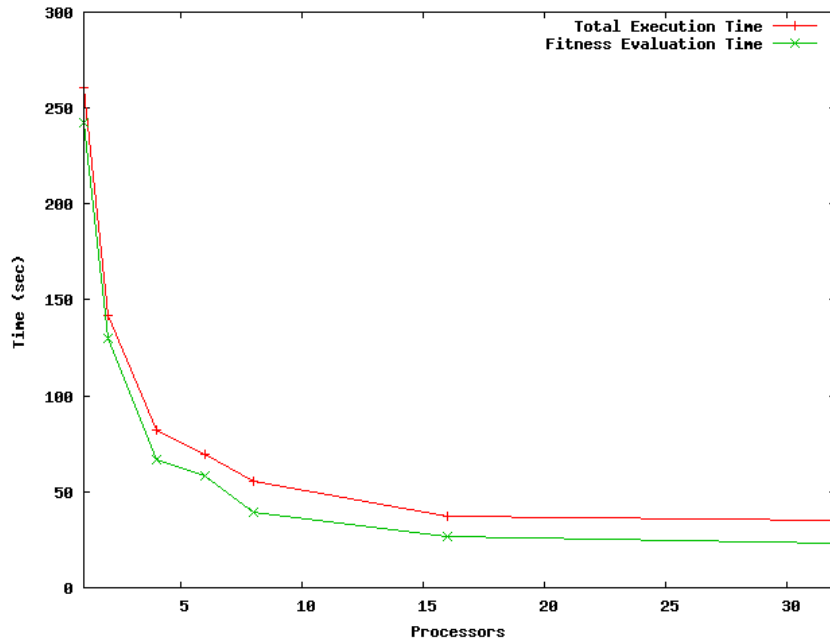


Figure 14: Total execution and fitness evaluation times. It is clear from this graph that the fitness evaluation is the most computational demanding step

processors for evaluation. After the evaluation is done, the slave nodes return the results to the master node, where it creates the next generation and again spreads them to the slave nodes for evaluation.

The advantage of this implementation is that the code is easily transferred into a single processor when needed. If for example it needs to be implemented directly into assembly, the parallelization section can be easily removed. On the down side, the speedup ¹, shown in Figure 16, is relatively low, as there are a lot of levels of parallelism not exploited in this GA implementation.

Further Parallelization

Further speedup is possible as noted in the previous section. One of the features that could be exploited, is the fact that each individual is made up by multiple windows. Spreading the windows of each individual into more processors, would dramatically decrease the computational time. Additionally communication can be minimized in the current implementation by sending

¹speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm, and is defined by $S_p = \frac{T_1}{T_p}$, where p is the number of processors, T_1 is the execution time of the sequential algorithm, and T_p is the execution time of the parallel algorithm with p processors. Linear speedup or ideal speedup is obtained when $S_p = p$

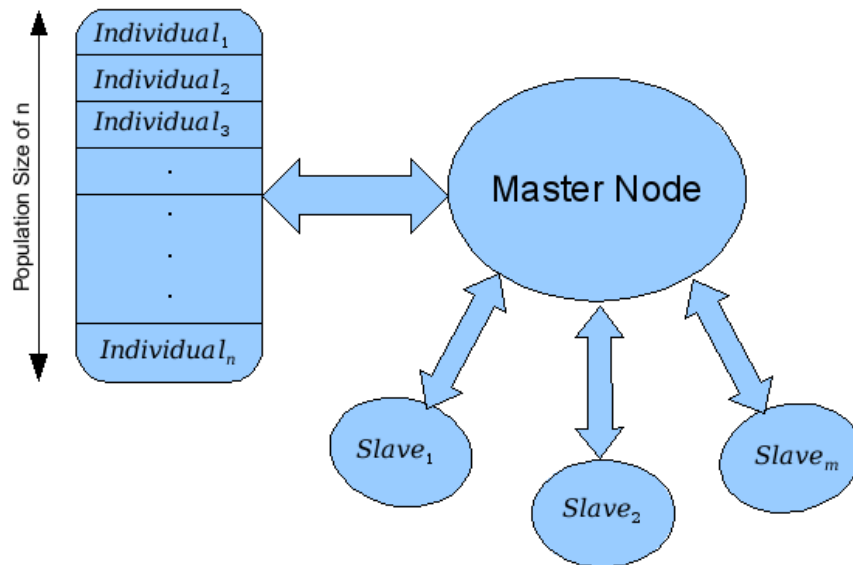


Figure 15: The Parallelization architecture. The master node holds a set of n individuals, where it spreads them over the available slaves. After the fitness evaluation is done results are returned to the master node .

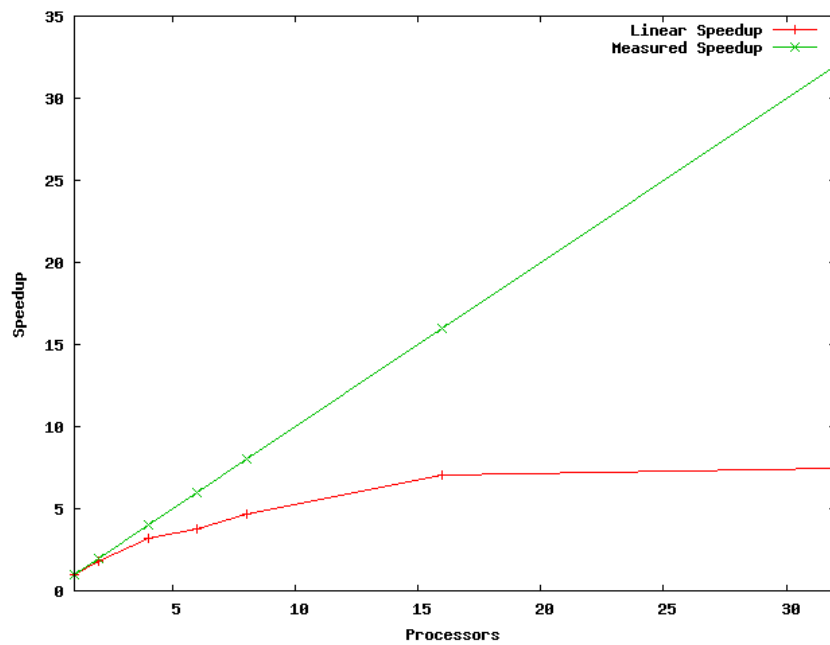


Figure 16: Speedup results. It is clear that with 16 processors or more any significant speed up is lost.

subset of individuals, and only returning their fitness to the master node. Another level of parallelism, can be obtained by spreading the disparity estimation, since each individual is tested on multiple images. Most of these levels of parallelism, are possible because of Matlab's nature. Since Matlab is designed to work with matrices, this makes it ideal environment for employing parallel tools.