Chapter 10

Wavefront/Systolic Algorithms for Implementation of Stereo Vision and Obstacle Avoidance Computations on a Very Low Power MIMD Many-Core Parallel Architecture: Applications for Mobile Systems and Wearable Visual Guidance

Francesco Diotalevi, Amir Fijany and Giulio Sandini

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/46028

1. Introduction

Mobile robots and humanoids represent an interesting and challenging emerging example of embedded computing applications. On one hand, in order to achieve a large degree of autonomy and to perform unmanned actions, these systems require a significant computational capability to perform a variety of tasks. On the other hand, they are severely limited in terms of size, weight, and particularly power consumption of their embedded computing system since they should carry their own power supply.

The conventional computing architectures are not well suited for these types of embedded applications due to the low computing power and/or high power consumption. However, emerging highly parallel and low-power architectures provide a unique opportunity to overcome the limitations of conventional computing architectures. These parallel architectures provide a much higher computing performance over conventional architectures while consuming significantly less power, resulting in a significantly better overall computing performance in terms of GFLOPs or GOPs per watt. Exploiting these novel parallel architectures, our current objective is to develop a flexible, low-power, lightweight supercomputing architecture for mobile robots and humanoid systems for performing various tasks and, indeed, for enabling new capabilities. In fact, some of our target small mobile robots are extremely limited in terms of power consumption and demand very low-power computing architecture.

Stereo vision (SV) computation is widely used in mobile robots applications to estimate the depth map of a 3D environment. SV is based on the computation of two images captured at the same time with slightly different viewpoints. In fact, the first step in a large set of image processing applications, e.g., for 3D modeling of environment [3], obstacle avoidance [4], navigation [2], object tracking [5], etc., is the computation of the depth information of 3D environment. Although the implementation of SV algorithms represents only the first step and part of the overall computation, conventional computing architectures, even while fully exploiting their features, cannot provide adequate performance for real-time computation [2] and/or in terms of low power requirements [9]. The NASA/JPL Mars Exploration Rovers (MERs) represent salient examples of low power mobile robots that rely on their on-board computing system to achieve local autonomy [22, 23, 24]. Both Spirit and Opportunity rovers rely on SV computation for autonomous navigation [23]. These rovers were designed to be, as much as possible, limited in terms of power consumption since they need to produce energy by using their solar panels. We mention these two examples of Mars missions since they are severely limited in terms of energy while demanding a rather high computational capability. As such, they represent the prime examples of the need for high computing power capabilities in very low power mobile robots. In fact, emerging mobile robot applications will be expected to achieve the same, if not more, level of local autonomy while relying on their embedded computing systems.

In this chapter, we present parallel/pipeline (wavefront/systolic) algorithms for efficient and very low-power implementation of the SV computation on a many-core MIMD architecture, the SEAforth 40C18 architecture [16]. This architecture is a 2D array of 40 cores that can deliver a performance of up to 25 GOPs. The maximum power consumption of the architecture is 250mW when all 40 cores run simultaneously at full speed. This represents a performance of 100 GOPS per Watt, making the SEAforth 40C18 architecture a very attractive candidate for very low-power embedded applications. For SV computation of images of 384x288 resolution, we have achieved a performance of up to 25 frames per second (fps) while consuming 75mW. To our knowledge, this seems to be one of the best performances in terms of fps per watt implementation results for the SV computation. It should be mentioned that multiple SEAforth 40C18 architectures can be coupled together, providing even a higher performance in terms of processing rate. We have also developed and implemented a simple Obstacle Avoidance (OA) algorithm based on the analysis of the computed depth map that clearly shows the high flexibility of this MIMD architecture. We have achieved a performance of 21 steering maneuvers per second while consuming 72mW of power. We have deployed the whole obstacle avoidance algorithm on a small mobile robot as the proof of concept. This very limited power consumption, for implementing both the SV and the OA computations, indeed enables the use of solar cells as the main source of power.

The main advantage of the SEAforth 40C18 architecture is that it provides a very high power efficiency, actually much higher than low-power FPGAs [21] while also offering flexibility and (to some degree) programmability like other many-core MIMD architectures. We have presented a comparison of the SEAforth 40C18 architecture with other available

low-power many-core MIMD architectures in [1]. However, the real challenge in its high performance application is in the design of efficient algorithms to optimize both the computation speed and the power consumption. To achieve such a goal, a deep understanding of the architecture and its capabilities and limitations is needed prior to any algorithm design. We discuss and show that this architecture is indeed very suitable for implementation of wavefront/systolic type of algorithms [15]. We have designed such algorithms for fast and low power computation of SV on this architecture.

This chapter is organized as follows. In Section 2, we discuss the stereo vision computation. In Section 3, we present and discuss the SEAforth 40C18 architecture with emphasis on some of its salient features which are exploited in our implementations and the challenges for developing efficient applications. In Section 4, we present the details of our parallel/pipeline implementation of SSD/SAD algorithms on the SEAforth 40C18 architecture. In Section 5, the developed OA algorithm is presented. The practical implementation results and performance of our developed algorithms are discussed in Section 6. The improvement of the OA algorithm is discussed in Section 7. An application of the developed algorithms and computing architecture as a wearable navigation system for visually impaired people is briefly discussed in Section 8. Finally some concluding remarks and directions for future works are presented in Section 9.

2. Stereo vision computation

SV has been extensively investigated and a great variety of algorithms have been developed for its computation. An extensive overview of stereo vision algorithms is presented in [6]. In general, dense stereo vision methods can be categorized into two classes: local and global. In local methods, disparity map is computed using a winner-takes-all (WTA) strategy, i.e. disparity of each pixel is calculated without considering disparity assignment of other pixels. In contrast, global methods formulate stereo matching as a global optimization problem. They make smoothness assumption, while preserving sharp discontinuity that may exist at object boundaries. It is shown that even for the simple discontinuity-preserving energy function, finding the global minimum is NP-hard [7]. Obviously, the more accurate is the depth estimation, the greater is the computational complexity of the algorithm. For real-time applications, local algorithms and in particular the Sum of Squared Difference (SSD) algorithm and the Sum of Absolute Difference (SAD) algorithm (a slightly simpler version of SSD) algorithm have been considered for implementation on various architectures, mainly due to their rather low computational cost.

2.1. SSD and SAD algorithms

The SSD algorithm is a local, window-based technique, to obtain the disparity map from a pair of rectified stereo images. Let $L_{n,m}$ and $R_{n,m}$ denote the intensity of pixels located at row m and column n in the left and right images, respectively. The input parameters of the algorithm are ω, the window size, and β, the disparity range value.

Assuming the left image as the reference, the disparity for each pixel *(n, m)* in the left image is calculated as follow:

- Consider a window centered at *(n, m)* in the left image
- Consider a window centered at *(n-k, m)* in the right image where $0 \le k < \beta$
- Calculate convolution of the windows in the right and left images as

$$\mathrm{SSD}_{n,m,k} = \sum_{i=n-\frac{\omega-1}{2}}^{n+\frac{\omega-1}{2}} \sum_{j=m-\frac{\omega-1}{2}}^{m+\frac{\omega-1}{2}} \left[L_{i,j} - R_{i-k,j}\right]^2 \quad (1)$$

The pixel that minimizes $SSD_{n,m,k}$ is the best match. That is,

$$k^* = \arg\, min_{0 \le k < \beta} \mathrm{SSD}_{n,m,k}\,,\ d_{n,m} = k^* \quad (2)$$

Briefly, the SSD algorithm consists of the following three steps:

1. Calculating the squared differences of intensity values for a given disparity.
2. Summing the squared differences over square windows.
3. Finding two matching pixels by minimizing the sum of squared differences.

The SAD algorithm is basically the same as SSD except that the sum of absolute differences operation is performed instead of sum of squared differences operation as:

$$\mathrm{SAD}_{n,m,k} = \sum_{i=n-\frac{\omega-1}{2}}^{n+\frac{\omega-1}{2}} \sum_{j=m-\frac{\omega-1}{2}}^{m+\frac{\omega-1}{2}} \left|L_{i,j} - R_{i-k,j}\right| \quad (3)$$

2.2. Previous works on fast implementations of SSD and SAD algorithms

For real-time applications, local algorithms and in particular the SSD and SAD algorithms have been considered, mainly due to their rather low computational cost. There are a number of reports in the literature focusing on real-time implementation of SSD and SAD algorithms on various architectures, ranging from General Purpose Processors (GPP) to FPGA implementation [8]-[13]. However, these approaches do not meet our requirements, particularly for very small robots, in terms of performance and power consumption. A more related work is the Tyzx DeepSea G2 Vision System [14]. The core of the system is a specific ASIC that performs patented, pipelined implementation of the Census stereo correlation algorithm. This architecture can achieve a processing rate of 200 frames per second (fps) for 512x480 images with a disparity range of 52 while consuming less than 1W. However, it should be emphasized that since the implemented algorithm is different with respect to conventional algorithms, it is not possible to benchmark it with SSD/SAD implementations.

3. The SEAforth 40C18 architecture and algorithmic design challenges

In this Section, we briefly discuss some of the salient features of the SEAforth 40C18 architecture which are exploited in our implementations. More detailed discussion can be found in [16]. We also discuss the challenges in designing efficient algorithms for this architecture. As stated before, we have presented a comparison of the SEAforth 40C18 architecture with other available low-power many-core MIMD architectures in [1].

3.1. A brief review of the architecture

The SEAforth 40C18 is a scalable embedded multi-core architecture consisting of a 2D array of 40 cores [16]. Each core, denoted as C18, is a complete processor with its own ROM, RAM, and inter-processor communication mechanism. Together, the 40 cores can deliver a performance of up to 25 GOPS. The maximum power consumption of the chip is 250mW when all 40 cores run simultaneously at full speed.

Figure 1. Block diagram of SEAforth 40C18, showing I/O and direction ports [16].

Each C18 core is identical to the others in terms of opcodes and architecture. Individual cores have different I/O options, and their ROM-based firmware differs slightly as well. Each core is a 18-bit processor which is a Forth stack machine [16]. Its instruction set consists of 32 basic opcodes. It uses a data stack for parameters and a return stack for control flow. The available programmers RAM for instructions and data for each core is only 64 words length. Each core runs asynchronously, at the full native speed of the silicon. Each step of a generic instruction is completed in about 1.6ns. Certain cores on the boundaries of the array (the colored cores in Fig. 1) have special I/O capabilities, and thus, can communicate with external devices.

Interior cores can communicate with their four immediate neighbors by using up, down, left and right ports. The ports between cores are bidirectional, half duplex asynchronous communication devices. The function of a communication port is to move words of data (and instruction) from one core to another with the minimum possible overhead in each individual transmission. For that reason when one core writes to another port, the binary values are asserted (not stored) until the receiving core reads them. A core waiting for data from a neighbor goes to sleep, dissipating less than 1μW. Likewise, a core sending data to a neighbor which is not ready to receive it goes to sleep until that neighbor accepts it.

With the implemented mechanism for moving data among cores, the synchronization happens automatically, thus making the development of algorithms and software much easier. In fact, this data driven nature of the SEAforth 40C18 architecture is exactly the same as that of Wavefront array architectures [15] wherein for each processor the arrival of data from neighboring processor is interpreted as a change in state and initiate some actions. This data driven mechanism substitutes the requirement of correct timing by correct sequencing

of the computations and hence eliminates the need for global control and global synchronization [15].

The programming language of SEAforth 40C18 is Forth which is a stack-oriented language [18]. Since many opcodes obtain their operands directly from the stacks, they are known as zero-operand opcodes. All opcodes are 5 bits in length, allowing multiple opcodes to be packed into and executed from a single 18-bit instruction word.

I/O ports on the SEAforth 40C18 are highly configurable since they are controlled by firmware. The 4-wire SPI port, the 2-wire serial ports, and the single-bit GPIO ports can be programmed to perform a large variety of functions. Ports can be programmed to support I2C, I2S, and asynchronous serial or synchronous serial communications. Serial Interfaces can reach a speed of up to 30Mbit/s. SEAforth 40C18 has also 2 parallel 18bit ports, which are usually used as address and data bus for accessing external memory, but they can also be used separately to perform double parallel data access to the chip. Speed of parallel interfaces can reach up to 100Mword/s.

Another important feature of this architecture is that two cores (core 1 and 31) provide fast intra-chip communication capability. These two cores employ a SerDes mechanism, i.e., a high speed Serializer/Deserializer that enables transfer of 18bit data and/or instructions between different SEAforth architecture with a speed of 400MHz [16]. Such a capability enables connecting multiple SEAforth 40C18 chips together resulting in larger arrays of cores with higher computing capability.

3.2. Challenges in algorithm design for SEAforth architecture

Exploiting the features of the SEAforth 40C18 architecture is even more challenging than it is for other many-core MIMD architectures, in terms of developing efficient algorithms and applications. To see this, note that the excellent power efficiency of the SEAforth 40C18 architecture is achieved at the cost of some rather drastic simplifications in the core architecture. In fact, in addition to the very simple and asynchronous nature of each core, there is also a very limited memory for each core, consisting of only 64 words for both instruction and data. This very limited memory space might indeed seem as a major bottleneck in developing algorithms and applications for efficient implementation on this architecture. Also the limited instruction set seems to be a limitation feature in developing efficient applications. These features of the architecture clearly indicate that it is more suitable for exploitation of a very fine grain parallelism in the computation, needing simple operations.

However, we believe that the features of SEAforth 40C18 architecture can be better exploited by using a more appropriate model of computation. In fact, from a computing point of view, the SEAforth 40C18 architecture can be considered as a Wavefront Array architecture [15]. Similar to Systolic Arrays [15], each core of the SEAforth 40C18 architecture is a rather simple processor with limited instructions set and memory and with a very fast nearest neighbor communication capability. However, the asynchronous nature of communication

and the data driven mechanism deployed in the SEAforth 40C18 architecture make it very similar to a Wavefront Array architecture. This similarity can in fact be exploited for efficient implementation of a large body of rather old algorithms and applications, originally developed for Wavefront Array, on the SEAforth 40C18 architecture. However, in adopting such algorithms, the constrained nature of the I/O capability in the SEAforth 40C18 architecture should be also taken into account. Another key and challenging issue is the optimization of required memory space as much as possible in the computation.

4. The developed stereo vision algorithms

In this Section, we analyze the main computational kernels of the developed and implemented SV algorithms on the SEAforth S40C18 Architecture. For our developed SV algorithms, we use the left image as the reference image. We also consider $\omega = 3$, as the window size, and $\beta = 16$, as the disparity range value. This means that each output pixel of the resulting depth map image is in the range (0,15), i.e., we have 16 different pixel values.

For a disparity of $\beta = 16$, we consider 16 cores (denoted by P_j, $0 \leq j < \beta$) constituting the *computation chain*. They are used as Processing Elements (PEs) and their code is optimized for performing nearest neighbor data movement and SSD/SAD computations.

Each PE in the computation chain has essentially the same code, with slight differences for P_0 and P_{15} (see Algorithm 1), with mainly three different phases that are iteratively repeated. The three phases are:

- Right Image Acquisition and Shifting,
- Left Image Acquisition and SSD/SAD Computation,
- Computed SSD/SAD Data Delivery.

For a window size of $\omega = 3$, the computation chain of PEs will be continuously fed by columns of 3 pixels of the right and left images.

When the computation chain is full it acts as a pipeline for each 2 (left/right) columns of 3 (ω) elements in the input of the computation (see Figure 2).

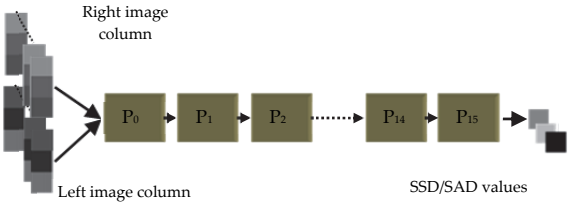


Figure 2. The Computation Chain acting as a Pipeline.

In the following, more details of the three processing phases of each PE are discussed.

4.1. Right image acquisition and shifting phase

This phase is necessary to accomplish the right image pixels movement in the computation chain. Considering a generic PE, say P_j, the previously acquired 3 right pixels (and already computed in the P_j, see below) are moved to the adjacent P_{j+1}. Next, the new 3 right pixels to be computed are received from the P_{j-1}. The data movement in this phase is different for P_{15}, since it doesn't deliver computed right pixels to any PE.

4.2. Left image acquisition and SSD/SAD computation phase

This phase is the most expensive in terms of number of operations and hence time. Here, 3 Square of Difference (SD) are computed and added to obtain the Column Sum Square of Difference (CSSD).

Let us analyze the algorithm in more details. By considering the pixel *(n,m)*, the inputs of P_j, are (see Figure 3):

- the right pixels column, just received in previous phase and then stored, i.e. the column *n+1-j*,
- the left pixels column, i.e. the column *n+1*.

Both columns are centered in the *m* row.

As described in the Algorithm 1, each left pixel is distributed among all the PEs of the computing chain. This is essential for making PEs of the computation chain working in a full parallel/pipeline mode as much as possible. Indeed, when each PE has both the left and right pixels it can then start to compute the SDs.

Once the left pixel is delivered, it is then used together with the first right pixel of the stored column to compute the $SD_{n+1-j,m-1}$ as:

$$SD_{n+1-j,m-1} = \frac{\left(L_{n+1,m-1} - R_{n+1-j,m-1}\right)^2}{2} \quad (4)$$

Note that each Square of Difference is divided by two, i.e., the result is scaled, to prevent the overflow, due to the internal 18 bits precision.

Right Image Acquisition and Shifting phase

wait until pixel $R_{n+1-j,m-1}$ is received
if (j!=15) **send** pixel $R_{n-j,m-1}$ to P_{j+1}
wait until pixel $R_{n+1-j,m}$ is received
if (j!=15) **send** pixel $R_{n-j,m}$ to P_{j+1}
wait until pixel $R_{n+1-j,m+1}$ is received
if (j!=15) **send** pixel $R_{n-j,m+1}$ to P_{j+1}

Left Image Acquisition and SSD/SAD Computation phase

wait until pixel $L_{n+1,m-1}$ is received

if (j!=15) **send** pixel $L_{n+1,m-1}$ to P_{j+1}

compute $(L_{n+1,m-1}-R_{n+1-j,m-1})^2/2=SD_{n+1-j,m-1}$

wait until pixel $L_{n+1,m}$ is received

if (j!=15) **Send** pixel $L_{n+1,m}$ to P_{j+1}

compute $(L_{n+1,m}-R_{n+1-j,m})^2/2=SD_{n+1-j,m}$

wait until pixel $L_{n+1,m+1}$ is received

if (j!=15) **send** pixel $L_{n+1,m+1}$ to P_{j+1}

compute $(L_{n+1,m+1}-R_{n+1-j,m+1})^2/2=SD_{n+1-j,m+1}$

compute $SD_{n+1-j,m+1} + SD_{n+1-j,m} + SD_{n+1-j,m-1} = CSSD_{n+1-j,m}$

compute $CSSD_{n-1-j,m}+ CSSD_{n-j,m}+ CSSD_{n+1-j,m}= SSD_{n-j,m}$

Computed SSD Data Delivery Phase

If (j!=*0*)

for (i=j; i>0; i--)

wait until SSD from P_{j-1} is received

send received SSD value to P_{j+1}

end for

send $SSD_{n+j,m}$ to P_{j+1}

Algorithm 1. Pseudo Code for generic PE P_j *(0≤j<β=16)*

This process is repeated three times, till we have the 3 computed SDs that are needed to calculate the Column Sum of Square Difference (CSSD) of pixel *(n+1-j,m)*, i.e.:

$$CSSD_{n+1-j,m}=\frac{\left(L_{n+1,m-1}-R_{n+1-j,m-1}\right)^2}{2}+\frac{\left(L_{n+1,m}-R_{n+1-j,m}\right)^2}{2}+\frac{\left(L_{n+1,m+1}-R_{n+1-j,m+1}\right)^2}{2} \quad (5)$$

Having previously stored inside the Pj, two *CSSD* (i.e. $CSSD_{n-1-j,m}$ and $CSSD_{n-j,m}$), we can then obtain the SSD of pixel *(n-j,m)* by simply adding the three CSSD contributions, i.e.:

$$SSD_{n-j,m}= CSSD_{n-1-j,m} + CSSD_{n-j,m} + CSSD_{n+1-j,m} \quad (6)$$

As shown in Figure 3, for each left and right columns as the input to the computation chain, the 16 cores compute the convolution of the 3x3 left window centered in (n,m) with the 16 3x3 right windows centered in (n,m), (n-1,m), (n-2,m), …(n-15,m). The 16 $SSD_{n,m,k}$ with $0≤k<\beta$ are then computed according to Eq. 1.

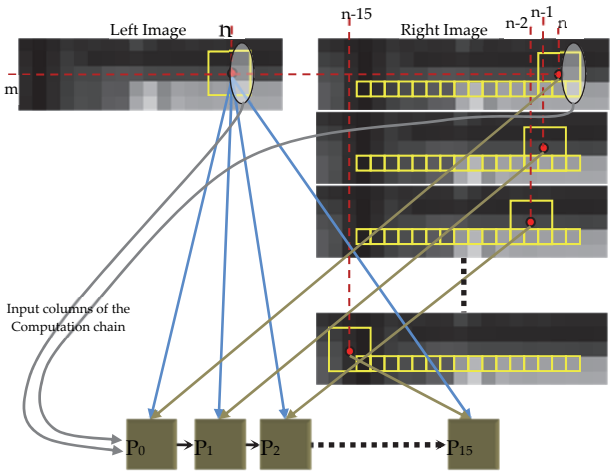


Figure 3. The resulting convolution computations performed by the 16 cores constituting the computation chain.

4.3. Computed SSD data delivery phase

In this phase, each P_j shifts its computed SSD value to its adjacent P_{j+1}. All the 16 computed SSD values need to be delivered to the core where the index of the PE with minimum SSD value will be calculated as the pixel value of the resulting Depth Map image. The SSD value computed by P_0 is the first SSD value received by the cores that search the minimum value. As soon as the each PE has transmitted its own SSD value it can then repeat the iterative cycle of Algorithm 1.

4.4. Mapping of the developed stereo vision algorithm onto SEAforth S40C18 architecture

In this Section we describe the mapping of the developed SV algorithm onto the SEAforth S40C18 architecture and our approach for improving the computational efficiency.

As shown in Figure 4, different cores of the 2D array have been used for performing different tasks as commonly practiced in any MIMD architecture. Essentially 4 different regions of cores can be highlighted and a total of 38 cores are used:

a. Cores constituting the computation chain. These 16 cores are shown in brown color and have been discussed before. We can further increase the processing rate by using the SAD algorithm instead of the SSD algorithm. Note that, the multiplication is significantly more expensive than calculation of absolute value on the SEAforth 40C18. The squared difference computation takes 56 steps, while the absolute difference computation takes only 19 steps to be processed.

b. Cores used to serially read and write the pixels from/to the external devices (the blue colored). The cores Rin and Lin are used for reading the Right and Left pixels respectively. The core Dout is used for writing the computed depth map pixels to the external devices.
c. Cores used to store 4 rows of pixels (2 rows for the left image and 2 rows for the right image). These 15 cores are highlighted in green color. Since the computation chain is continuously fed by sequences of columns of 3 pixels (of the Right and Left images) and since the images are both scanned row by row, the upper two pixels of each column (i.e. two rows) can be stored into the SEAforth S40C18 architecture for reducing the input (and the output) data rate. In this way we can read from the external cameras one pixel at a time and the column of 3 pixels is automatically built by using the 2 previously stored rows of pixels. As mentioned before, one of the key challenges for mapping any computation on the SEAforth S40C18 architecture is the very limited memory space of each core. We have used this scheme to circulate the data in the architecture and use some cores as data buffer to overcome this limitation.
d. Cores used to perform the search of the pixel with minimum disparity. These cores are highlighted in pink color. Each one of the 4 cores performs minimum searching process among 4 different values computed by PEs. This synergic searching is much more efficient in terms of time with respect to performing it in only one core.

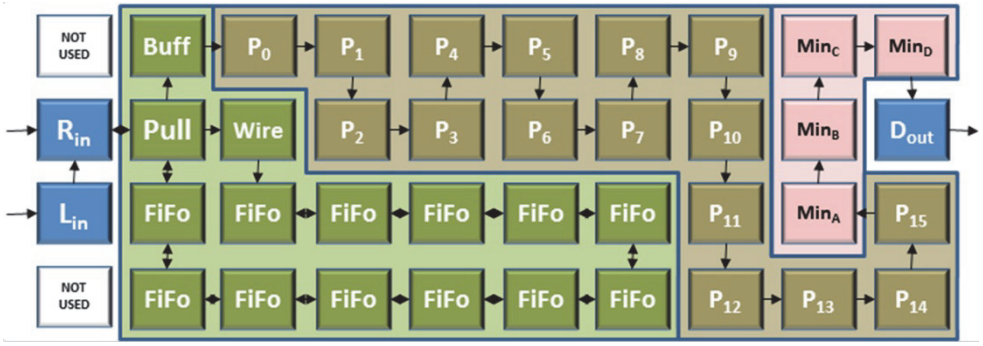


Figure 4. Mapping of the developed SV algorithm onto the internal cores of the SEAforth 40C18 architecture.

4.5. Stereo vision algorithm performance results

The simulation results of the developed SV algorithms are here summarized and discussed. The developed SV algorithms have been tested by using the Tsukuba set [20] that has been widely used in the computer vision literature (Figure 5).

The algorithms have been verified and simulated by using the VentureForth development tool. This includes compiler, debugger, and simulator [19]. In order to evaluate the performance of the algorithms, we have used the performance profiler which is included in the debugging tools provided to the user.

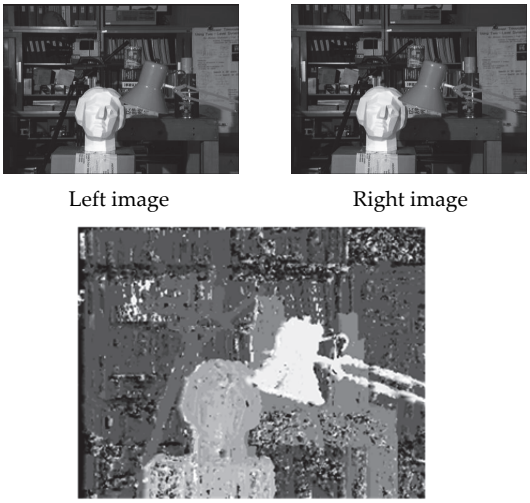


Figure 5. The original Tsukuba images and the resulting depth map of the SAD algorithm.

The performance profile is automatically generated and shown as a heatmap picture in Figure 6.

The heatmap shows the activity of each core during the simulated run time with colors and the data movement by arrows. Black colored core means 0% activity; while red colored core means 100% activity (i.e. core never went into sleep state). At the end of each computation, the heatmap accurately reports the total number of steps required for performing the overall computation and the total load of the computed task (data movement included) in terms of percentage of the maximum power dissipation.

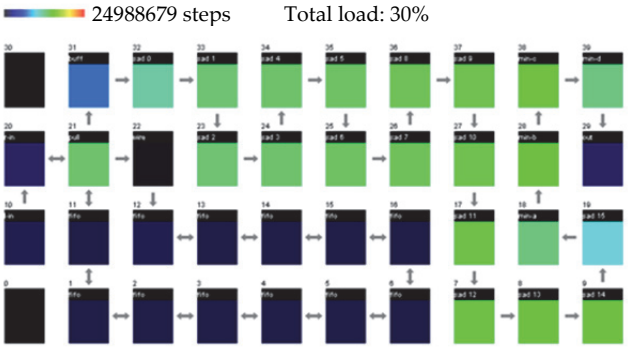


Figure 6. Heatmap of the developed SAD Algorithm.

By using these figures reported in the heatmap, an accurate estimation of computing time and power consumption can be obtained. Results for SSD and SAD computation are shown

in Table 2 and in Table 3. In the SAD computation, the total number of steps, for processing one depth map image of 384x288 pixels, with a disparity range of 16, by using our parallel algorithm, is 24988679. Since, as stated in Section 3, each step takes 1.6ns, the overall computation time is then ≈40ms. This computation time represents a processing rate of ≈25 fps. The total power consumption during the SAD algorithm computation is 30% of the maximum power dissipation, i.e., 75mW. The input and output data rates for sustaining this computation rate of 25fps are obtained as 22Mbit/s.

Developed SV Algorithm	fps	Power consumption	Input Data Rate	Output Data Rate
SSD	14fps	81mW	13Mbit/s	13Mbit/s
SAD	25fps	75mW	22Mbit/s	22Mbit/s

Table 1. Developed SV Algorithms performance results for 384x288 pairs images with disparity=16 and window=3x3 (vendor simulator)

With SAD computation we can then achieve close to real-time computation by using only 75mW of power. Moreover the input and output data rates are suitable for hardware implementation.

Table 3 shows the sustained performance. The sustained performance column of Table 3 has been computed taking into account the images resolution, the window size, the disparity value, the fps value and the number of operations for computing the algorithms.

Developed SV Algorithm	Sustained Performance [MOPs]	Sustained Performance per Watt [GOPs/W]	Mega Pixels Disparity per second [MPDs]
SSD	345.27	4.26	24.8
SAD	351.12	4.68	44.2

Table 2. Developed SV Algorithms sustained performance results for 384x288 pairs images with disparity=16 and window=3x3 (by using the vendor simulator).

In the metric of Pixels x Disparity measures per second (PDS), the SAD algorithm achieves a performance of 44.2MPDS. The achieved sustained performance per Watt is of 4.68 GOPs/W.

5. The developed obstacle avoidance algorithm

Systems that base their mobile capability on visual sensors such as stereo cameras usually use analysis of their computed depth map to avoid obstacles. For example, [22] describes the entire autonomous driving software architecture, including the stereo vision and obstacle avoidance architecture, for MER rovers. Beside the depth map computation, our objective is also to develop a very power efficient algorithm for obstacle avoidance suitable for autonomous low power mobile robots.

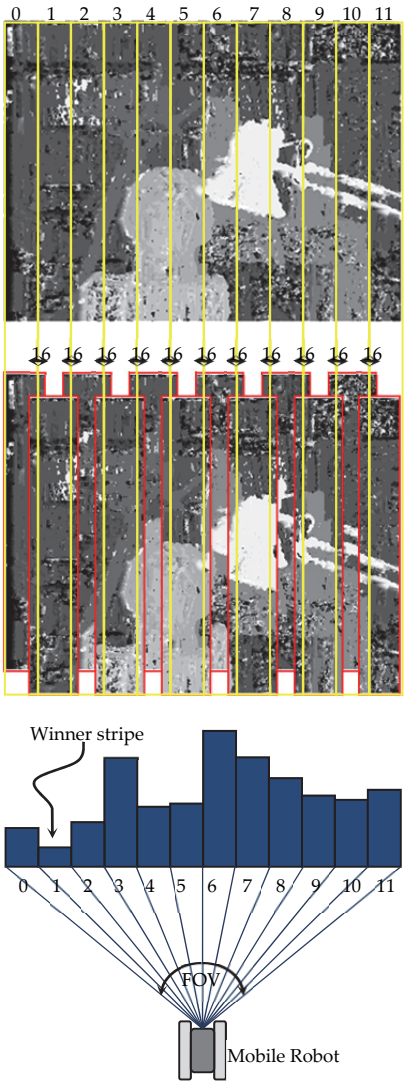


Figure 7. The 12 overlapped stripes used by the implemented Obstacle Avoidance Algorithm.

The Obstacle Avoidance (OA) algorithm described here is based on the analysis of the depth map image computed as discussed in the Section 4. It enables the mobile robot to avoid any existing obstacle in its environment.

The proposed algorithm is inspired by the work in [4]. In our implementation, the depth map image is divided in 12 stripes as shown in Figure 7. For each one of the 12 stripes, we compute the summation of white pixels (i.e. pixels of the objects that are closer to the camera) that are inside their boundaries. For making the algorithm more reliable, we have

considered overlapping stripes, that is, adjacent stripes overlap by 16 pixels. The overlapping of stripes is needed for efficient detection of objects close to the boundaries of the stripes.

The final results of the analysis of the computed depth map image are 12 values that are the summation of white pixels for each stripe. The decision making for navigating the robot is simply based on choosing the stripe with minimum value and then moves the mobile robot in the direction of that stripe. For instance, in the case of the Tsukuba images (Figure 7), the stripe with minimum value is the number 1. This means that the robot has to turn left proportionally of 5/12 of the Field Of View (FOV) of the camera.

5.1. Mapping the obstacle avoidance algorithm onto SEAforth S40C18 architecture

In this Section, we describe mapping of the developed OA algorithm onto the SEAforth S40C18 architecture.

Our aim has been to develop a power efficient OA Algorithm based on the analysis of the computed depth map image. For this reason, our strong requirement has been to use only one S40C18 chip for both SV and OA computation. This assumption has meant that we had to change the arrangement of the computation of the developed SV algorithm, as described in Section 4, in a way to also include the computation of the OA algorithm. The developed code for our OA algorithm is small enough to be fitted in only two cores. To reach this goal, we have modified the algorithm used to perform the minimum index search, to fit into two cores instead of four. In this way, the whole OA algorithm fits into 38 cores, as is shown in Figure 8.

The analysis of the computed depth map image is performed by two cores:

a. One core ("Ob Avoid" in Figure 8) is used to compute the white pixels summations in the 12 overlapped stripes. As soon as the whole depth map has been analyzed, it delivers the 12 values to its adjacent core.
b. One core ("Min Stripe" in Figure 8) is used to search the index value of the stripe that corresponds to the minimum value of the 12 received values from the "Ob Avoid" core. The index of the stripe with minimum value is then delivered to the core D_{out}. This core transmits the stripe index to the external device for steering maneuvers.

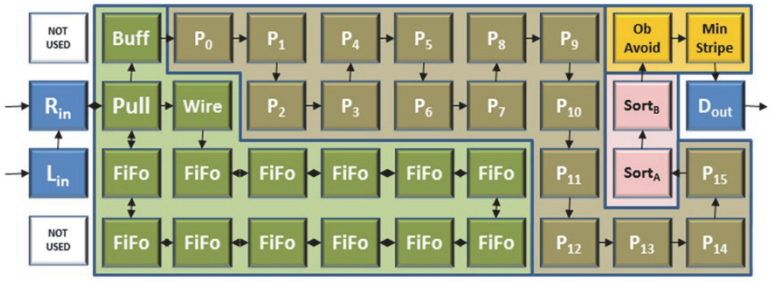


Figure 8. Map of the developed SV algorithm onto the internal cores of the SEAforth 40C18 architecture.

5.2. Obstacle avoidance algorithm performance results

The simulation results of the developed obstacle avoidance algorithm are here summarized and discussed. Similar to the depth map simulation results, we have used the heatmap obtained in the simulation, shown in Figure 9, for a complete analysis of the performance in terms of both power and computation time.

Due to the fact that the numbers of cores used to perform the search for the pixel with minimum disparity are now 2 instead of 4 (i.e. less parallelism means less performance in terms of execution time), now the computation of the depth map is a little slower. However, we are able to obtain a winner stripe value each 29534069 steps, i.e. 1 steering maneuver each 47.2ms or, ≈21 steering maneuvers per second. The power consumed for performing both the SV and the OA algorithm is ≈72mW (i.e. 24% of the maximum power dissipation).

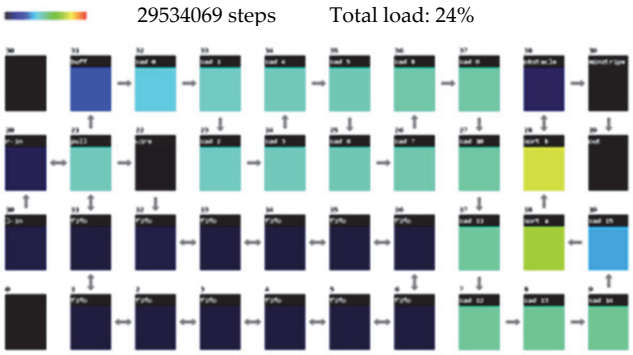


Figure 9. Heatmap of the developed obstacle avoidance Algorithm.

The Table 4 summarizes the obtained performances. For determining the steering maneuvers data rate we suppose to use a simple UART protocol with 8 bit of data, 1 Stop bit and no parity.

Developed SV Algorithm	Steering maneuvers	Power consumption	Input Data Rate	Steering maneuvers Data Rate
SAD	21 maneuvers/s	72mW	22Mbit/s	210 bit/s

Table 3. Developed OA Algorithm performance results for 384x288 pairs images with disparity of 16, 3x3 window and SAD SV Algorithm (vendor simulator)

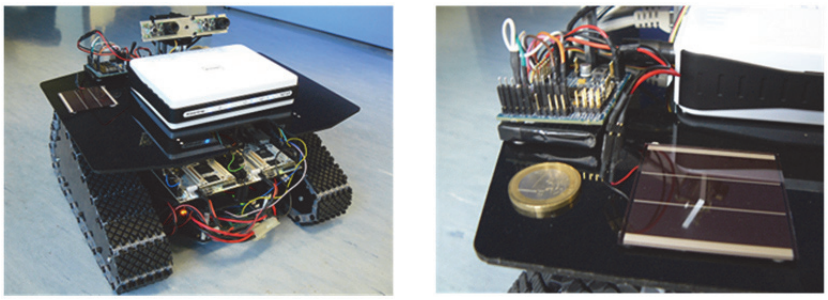
6. Practical implementation results

We have successfully tested the performance of the developed SAD algorithm in hardware by measuring the time spent to obtain 1 pixel of the resulting depth map. We considered that the Left and Right image were stored in the chip; in this way we are sure to measure the

computation time of the developed SAD algorithm without taking into account the I/O issues. We measured that the time spent to obtain 1 pixel by the SAD algorithm was ≈360ns. This value fully agrees with the simulation results of Section 4, achieving ≈40ms for computing a complete 384x288 depth map image.

As the proof of concept, we have also deployed the developed OA algorithm on a small self-powered mobile robot, called iCrawler, shown in Figure 10. This mobile robot has a stereo cameras unit installed on-board and wifi router for wireless communication. By using the OA as described in previous Section, the iCrawler is able to avoid obstacle that are in the FOV of the stereo camera as shown in Figure 11.

The iCrawler uses 2 ARM boards. Each one is able to: acquire images from USB camera; perform jpeg decompression of the images; perform images rectifications; serially access the SEAforth S40C18 architecture and send computed maneuvers to motors actuators to avoid obstacles.



(a) (b)

Figure 10. (a) the iCrawler mobile robot used for proof of concept. (b) the small 3.7v, 4.6Wh LiIon battery (under the S40C18 development board) and the solar cell used as main source of power are shown.

Because of the limitations in terms of speed of the ARM boards for performing the tasks described above, we have achieved 3 fps as the best performance to compute the depth map images and 3 steering maneuvers per second to avoid obstacles. With this limitation in terms of data rate feeding into IntellaSys chip, we have measured a consumed power of only ≈8mW. In fact, since the IntellaSys S40C18 architecture is a data driven architecture, by lowering the input data rate feed, the power consumption for performing the SV algorithm decreases consequently.

It should be emphasized that by using only a small 3.7v 4.6Wh LiIon battery, the on-board S40C18 architecture can compute steering maneuvers for obstacle avoidance consecutively for more than 20 days.

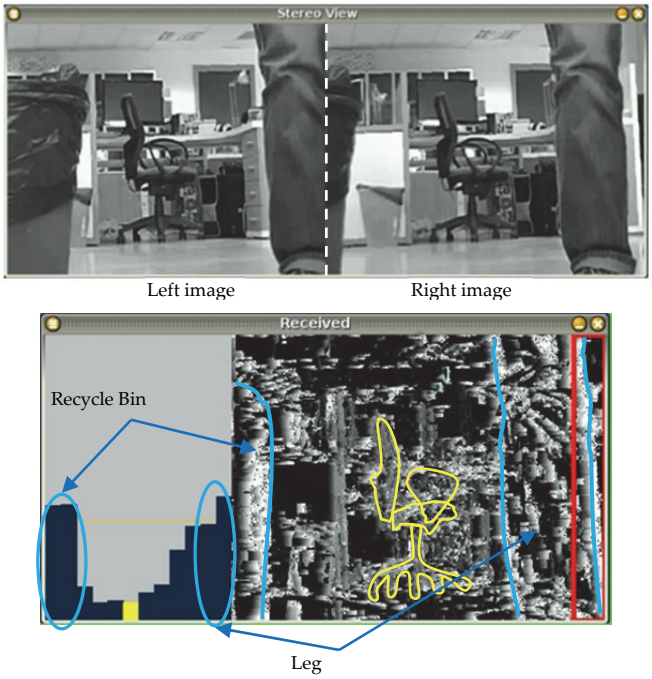


Figure 11. Example of acquired stereo scene and the computed depth map with the computed histogram. The recycle bin and the leg in the foreground are well detected and shown. The winner stripe (the yellow colored in the histogram) is also shown.

From the power consumption point of view, our current implementation demonstrates that the architecture can also be fully powered by using a solar cell panel as the main source of energy for recharging the battery for the whole architecture. The very low power consumption and consequently the possibility to adopt solar cells becomes a key feature, for example, for unmanned vehicle for planetary exploration as in MER [23] or any other ultra low power autonomous robot.

7. Further improvement of the obstacle avoidance algorithm

As described in the Section 5, the obstacle avoidance algorithm is based on the vertical slicing of the depth map. If an object was in the top of the scene, and then, it was not an obstructive object, the OA algorithm would have treated it as an obstacle to avoid. For example, even a scene of a bridge, could cause problems in the OA described in Section 5, by not permitting the rover to pass under it. See for instance Figure 12.

We can overcome this limitation by dividing the depth map into tiles. We implemented an OA algorithm based on the depth map divided into 4x12 tiles, as shown in Figure 13. Here, instead of having a histogram of values proportional to the distance of the object from the cameras, we have a 2D map of such values.

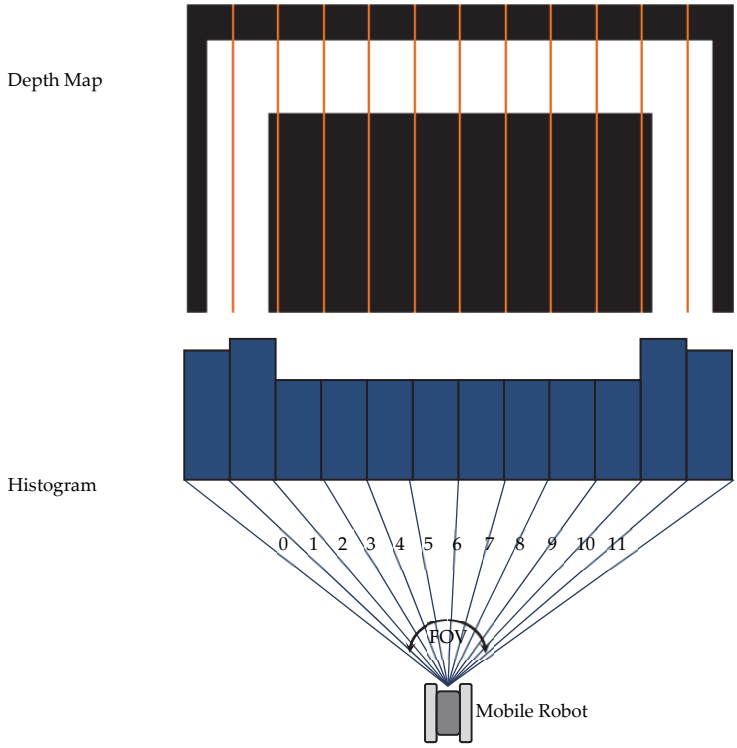


Figure 12. Depth map and histogram of an acquired scene of a "bridge".

So, the OA algorithm by simply using the 4x12 values of this map can make decision about its steering maneuvers, not considering obstacles that are detected on the top row of the map (of course this depends on the focal length of the lens of the camera, the height of the rover and so on). For instance, the case of the "bridge" is not anymore an issue for this improved version of the OA algorithm as shown in Figure 14.

As in the previous OA algorithm implementation, the analysis of the computed depth map image in the improved OA algorithm is performed by two cores:

a. One core ("Ob Avoid" in Figure 8) is used to compute the white pixels summations for each 12 tiles constituting 1 row (the depth map is sliced into 4 horizontal rows). As soon as a row has been analyzed, it delivers the 12 values to its adjacent core.
b. One core ("Min Stripe" in Figure 8) is used to deliver the 12 values for each row to the core D_{out} and also to accumulate the 12 values of each row constituting the depth map in a way to always to deliver the information as the previous OA algorithm in terms of winning vertical stripes.

So, the data sent to the host performing the steering maneuvers are a total of 49 data, 48 of them are relative to the 2D map of the analyzed depth map, and one is the winner stripes as

before. The result of the improved OA algorithm, when applied to the Tsukuba images, is shown in Figure 15.

Based upon these values, the host can simply modify the trajectory of the rover to avoid real obstacles in front of it. See for instance the rover behavior in case of a "bridge" in Figure 14.

In terms of performance of the improved OA algorithm, we have measured a slight increase in the power consumed as shown in Table 4. The increase in power consumption is mainly due to the increased activity of the D_{out} Core. Indeed, in the improved OA algorithm, it has to deliver out 49 values instead if only one (see Figure 15).

Developed SV Algorithm	Steering maneuvers	Power consumption	Input Data Rate	Steering maneuvers Data Rate
SAD	21 maneuvers/s	74mW	22Mbit/s	210 bit/s

Table 4. Improved OA Algorithm performance results for 384x288 pairs images with disparity of 16, 3x3 window, and SAD SV Algorithm (vendor simulator)

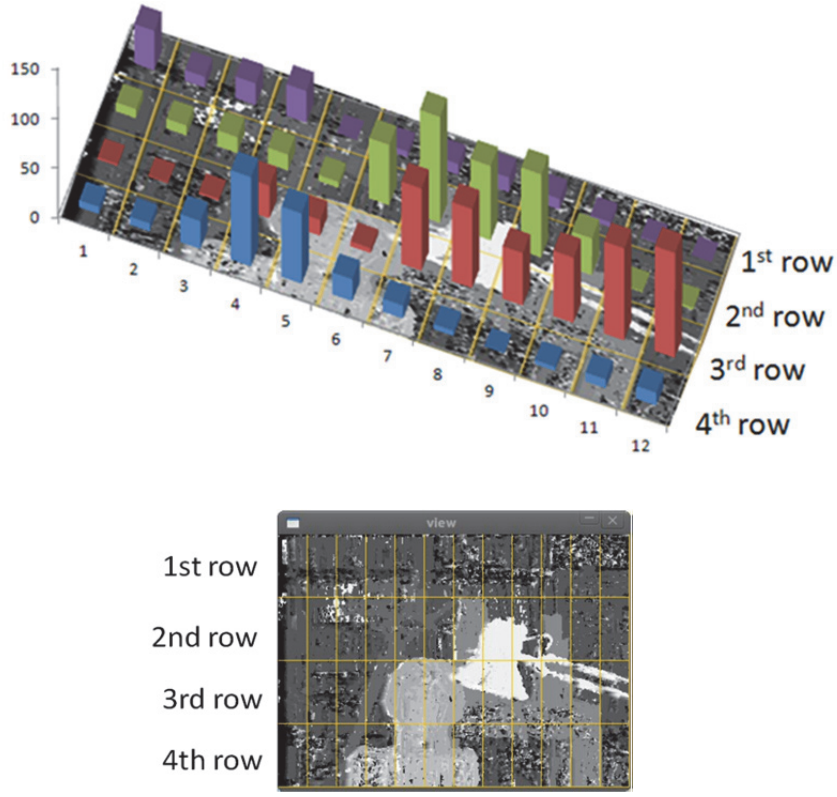


Figure 13. Depth map and 2D map of the Tsukuba images.

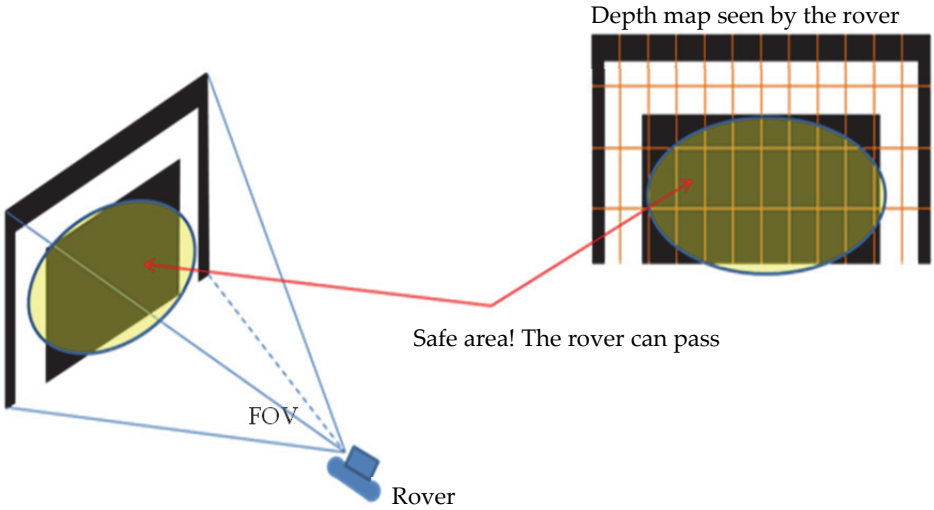
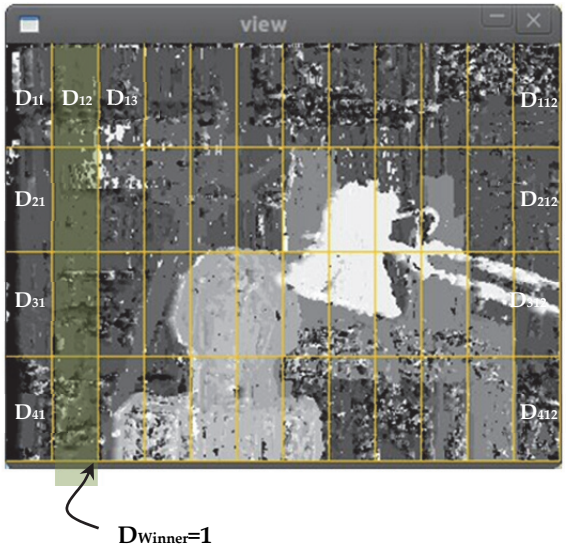


Figure 14. The "bridge" test case for the improved OA algorithm.



The values delivered to the host that takes decisions in terms of steering maneuvers are:
{D11, D12, D13, ... D112, D21, ... D212, D31, ... D312, D41, ... D412, 1}

Figure 15. Improved OA algorithm data communication to the host for Tsukuba images.

8. Wearable navigation system for visually impaired people

The excellent performance of our developed OA algorithm, in terms of speed of computation and, more significantly, the power consumption indeed enables other applications. Note that, a consequence of such a low-power system is that it drastically reduces the size and weight of the overall system since a very small and light weight power source, e.eg, a small battery, can be used.

One application that we are currently investigating is the development and deployment of our OA system as a navigation aid for visually impaired people. Note that, the use of visual sensors for navigation aid has been previously proposed (see, for example [25, 26]. However, the proposed systems require rather heavy equipments.

In this section, we briefly describe our proposed system which can be used as a special glass to aid navigation of visually impaired people. This proposed system, shown in Figure 16, consists of two cameras, the circuitry for computing depth map and OA, as described before, and the batteries. With respect to other similar stereo vision systems it will be really wearable due to its low power consumption and particularly light weight. Furthermore, the system would be able to perform required computations for a long period of time without any need for recharging the batteries.

The solar cells used in place of lenses are used to increase the batteries life and to fully supply the IntellaSys module. The innovative low cost, hybrid solar cells based on colloidal inorganic nanocrystal and DSSC technology [27] can be used for such a purpose.

The earphones are used by the visually impaired people to hear the commands in way to detect obstacles in front of him. The WiFi module adds communication capability to the glasses.

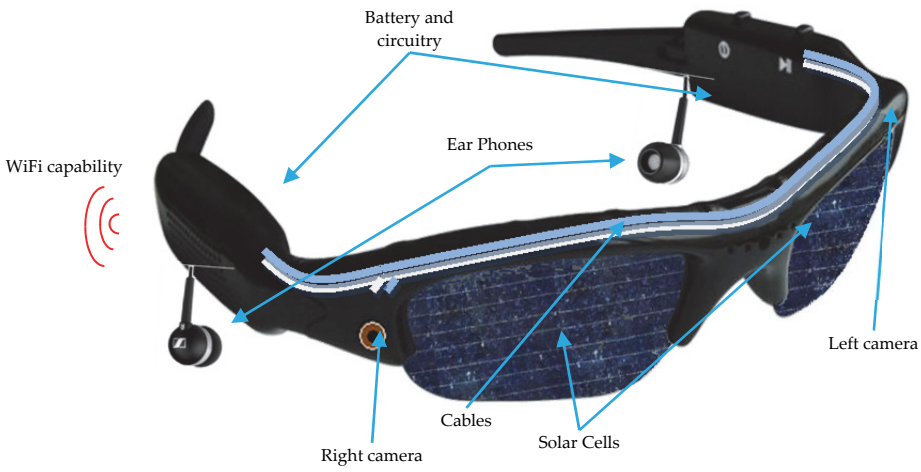


Figure 16. Proposed Wearable Vision System for visually impaired people

The proposed system can be used to directly implement the OA scheme, described before.

Another alternative is to use it synergistically with the smart phone, to help blind people in self navigation tasks. The real scene in front of impaired vision person is acquired and translated by the proposed system into a depth map image. The depth map is sent, via Wifi, to a smart phone. The smart phone uses the maps freely available from internet and merges the information in such a way to guide the person to the desired destination while avoiding local obstacles (see Figure 17).

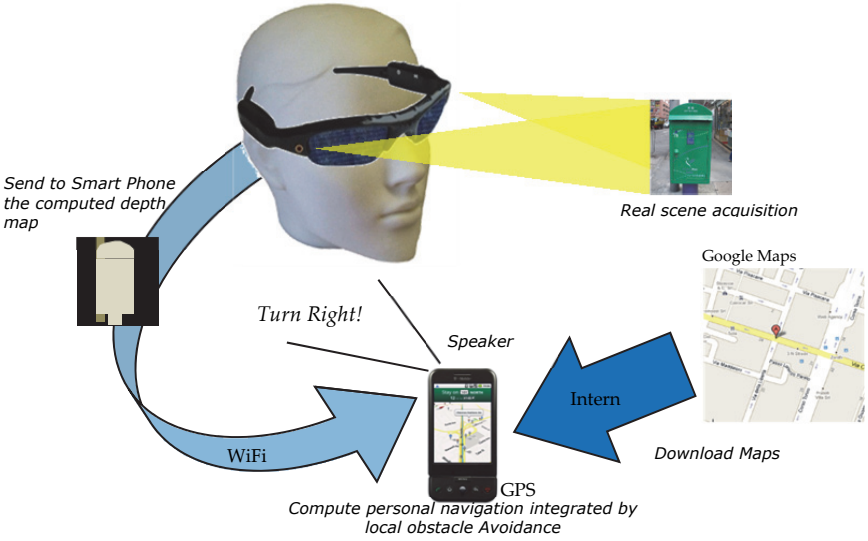


Figure 17. Proposed Personal Navigator for impaired vision people using the Wearable Communicative Stereo Vision Glasses

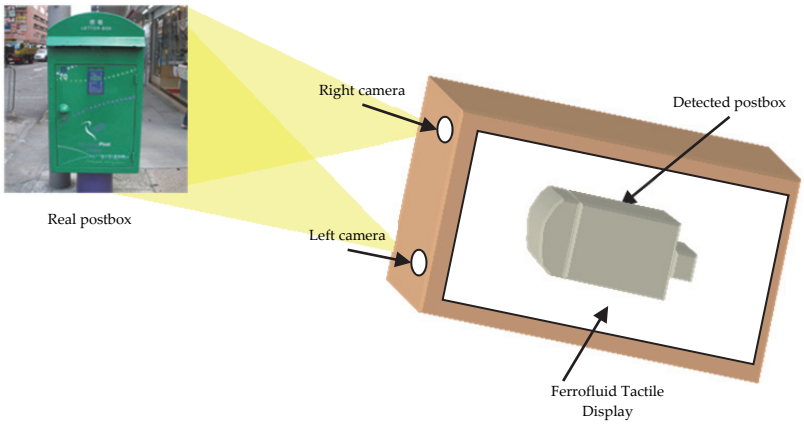


Figure 18. Prototype of Mobile 3D environment detector for Blind People

Another possibility is to use the system as a Mobile 3D environment detector as shown in Figure 18. By using a ferrofluid tactile display [28], and our improved OA scheme, it can generate tactile information based upon the scene detected in front of the device.

9. Discussion and conclusion

In this chapter, we presented an efficient parallel/pipeline implementation of SSD/SAD stereo vision algorithm on a novel and innovative many-core MIMD computing architecture, the SEAforth 40C18. We discussed the details of our parallel/pipeline implementations. To our knowledge, our results, obtained through simulation and verified in practical hardware implementation, seem to be among the best results in terms of performance per watt in computation of the SSD/SAD algorithms. Our results show that the depth map computation can be performed close to real-time (25fps) by consuming only 75mW. We also developed a novel obstacle avoidance algorithm that, by using the computed depth map, enables safe navigation and obstacle avoidance. This algorithm has also been successfully deployed as the proof of concept on a small mobile robot.

We showed that by using an appropriate model of computation, similar to Wavefront Arrays while also exploiting the asynchronous and MIMD features of this architecture, it is possible to efficiently implement algorithms for very low power mobile robots. As an example, in our OA implementation, 16 cores are used as a computing chain to perform the computation exactly as in a Wavefront Array. In fact, the Algorithm 1 represents a Wavefront algorithm. Other cores are used for performing other tasks such as buffering data (12 cores), performing the search for the pixel with minimum disparity (2 cores), analyzing the computed depth map (2 cores) and serial data communication (3 cores). Also, 3 cores are used for redirecting data, providing a communication path between cores which are not nearest neighbor. We should emphasize that the fact that the SEAforth 40C18 architecture represents an efficient and even more flexible practical implementation of Wavefront Arrays paves the way for developing other new efficient applications. This can be achieved by leveraging the rather large body of applications and algorithms originally (and mainly theoretically) developed for Wavefront Array processing [15].

Better performance in terms of fps in stereo vision algorithms can be achieved by using multiple SEAforth 40C18 architectures. For example, for implementation of SSD/SAD algorithm, the image can be divided into 4 stripes, each assigned to and computed by a SEAforth 40C18 architecture. This division would involve a very small overhead due to the overlapping of the boundary data but can lead to a speedup very close to 4. That is, a processing rate of about 100 fps can be achieved while consuming about 300mW. Similarly, multiple SEAforth 40C18 architectures can be coupled together to compute depth map images with bigger size and/or with higher depth range.

This very limited power consumption, for implementing both the SV and the OA computations, indeed enables the use of solar cells as the main source of power for the computing architecture. Such a high performance and low power computing system can enable new capabilities and applications. As an example, we briefly presented and

discussed the wearable navigation system for visually impaired people, by using our developed algorithms and computing architecture.

Author details

Francesco Diotalevi and Giulio Sandini
Robotics, Brain and Cognitive Sciences Department, Istituto Italiano di Tecnologia, Genova, Italy

Amir Fijany
SAAE SysTech, Inc., Los Angeles, CA, USA

10. References

[1] F. Diotalevi, A. Fijany, M. Montvelishsky and J-G. Fontaine,"Very Low Power Parallel Implementation of Stereo Vision Algorithms on a Solar Cell Powered MIMD Many Core Architecture," Proc. IEEE Aerospace Conf., Big Sky, MO, March 2011.

[2] W. van der Mark and D.M. Gavrila, "Real-time dense stereo for intelligent vehicles," IEEE Transactions on Intelligent Transportation Systems, Vol. 7(1), pp. 38-50, 2006.

[3] S. Fleck, F. Busch, P. Biber, H. Andreasson, and W. Straßer, "Omnidirectional 3d modeling on a mobile robot using graph cuts, " Proc. IEEE ICRA '05, pp. 1748-1754, April 2005

[4] L. Nalpantidis, I. Kostavelis and A. Gasteratos, "Stereovision-Based Algorithm for Obstacle Avoidance," in International Conference on Intelligent Robotics and Applications, ser. Lecture Notes in Computer Science, Vol. 5928. Singapore: Springer-Verlag, 2009, pp. 195–204

[5] F. Tang, M. Harville, H. Tao, and I.N. Robinson, "Fusion of Local Appearance with Stereo Depth for Object Tracking," In Computer Vision and Pattern Recognition Workshop, IEEE Computer Society Conference, pp. 1–8 (2008)

[6] D. Scharstein, R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," International J. of Computer Vision, Vol 47(1-3), pp. 7-42, 2002.

[7] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23(11), pp. 1222-1239, 2001.

[8] H. Hirschm, P.R. Innocent, J. Garibaldi,"Real-time correlation-based stereo vision with reduced border errors," International J. of Computer Vision, Vol. 47(1-3), pp. 229-246, 2002.

[9] R. Yang and M. Pollefeys, "A versatile stereo implementation on commodity graphics hardware," J. of Real-Time Imaging 11(1), pp. 7-18, 2005.

[10] H. Sunyoto, W. vander Mark, and D.M. Gavrila, "A comparative study of fast dense stereo vision algorithms," Proc. Intelligent. Vehicle Symposium, pp. 319-324, 2004.

[11] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," Proc. Third International

Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), pp. 798-805, Washington, DC, USA, 2006.

[12] J. Woodfill, B.V. Herzen, "Real-time stereo vision on the parts reconfigurable computer," Proc. IEEE Workshop FPGAs for Custom Computing Machines, pp. 242-250, 1997.

[13] C. Murphy, D. Lindquist, A.M. Rynning, T. Cecil, S. Leavitt, M.L. Chang, "Low-cost stereo vision on an FPGA," Proc. 15th IEEE Symp. on Field-Programmable Custom Computing Machines, pp. 333-334, 2007

[14] J. Woodfill, et al, "The Tyzx DeepSea G2 Vision System, A Taskable, Embedded Stereo Camera," Proc. of the IEEE Computer. Society Workshop on Embedded Computer Vision, Conference on Computer Vision and Pattern Recognition, June 2006.

[15] S.Y. Kung, VLSI Array Processors. Prentice Hall, 1988.

[16] IntellaSys, SEAforth 40C18 Data Sheet. Version 9/23/08, available on web: http://www.intellasys.net

[17] E. Rather and the technical staff of IntellaSys, "VentureForth Programmers Guide", available on web: http://www.intellasys.net

[18] E.D. Rather and E.K. Conklin, "Forth Programmer's Handbook", 3rd Edition.

[19] IntellaSys, Venture Forth Compiler and Simulator, rev. 1.4.0, available on web: http://www.intellasys.net

[20] Head scene images: http://www.csd.uwo.ca/~yuri/Gallery/stereo.html

[21] http://www.actel.com/products/pa3l/default.aspx, 2010

[22] J.J. Biesiadecki and M.W. Maimone, "The Mars exploration rover surface mobility flight software: Driving ambition," Proc of IEEE Aerospace Conference, vol. 5, Big Sky, MT, Mar. 2005

[23] L. Matthies et al, "Computer Vision on Mars," International J. of Computer Vision 75(1), pp. 67-92, 2007.

[24] M. Maimone, A. Johnson, Y. Cheng, R. Willson and L. Matthies, "Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission," in Proc. 9th International Symposium on Experimental Robotics (ISER), Singapore, June 2004.

[25] N. Molton, S. Se, J. M. Brady, D. Lee and P. Probert "A stereo vision-based aid for the visually impaired", Image and Vision Computing Volume 16, Issue 4, 4 March 1998, Pages 251-263G. Balakrishnan, G. Sainarayanan, R. Nagarajan and Sazali Yaacob, "Wearable Real-Time Stereo Vision for the Visually Impaired", Engineering Letters, 14:2, EL_14_2_2 (Advance online publication: 16 May 2007)

[26] G. Balakrishnan, G. Sainarayanan, R. Nagarajan and Sazali Yaacob, "Wearable Real-Time Stereo Vision for the Visually Impaired", Engineering Letters, 14:2, EL_14_2_2 (Advance online publication: 16 May 2007)

[27] http://cbn.iit.it/research-platforms/energy/research-activities/dssc.html

[28] Y. Jansen, T. Karrer, J. Borchers, "MudPad: tactile feedback and haptic texture overlay for touch surfaces," Proc of ITS'10, ACM International Conf. on Interactive Tabletops and Surfaces, November 7–10, 2010, Saarbr¨ucken, Germany, pp. 11-14.