Stereovision-based 3D obstacle detection for automotive safety driving assistance

Nicolas Ventroux, Renaud Schmit and Stéphane Guyetant

CEA, LIST
Embedded Computing Laboratory
Mail Box 94 - F91191 Gif-sur-Yvette Cedex
Email: nicolas.ventroux@cea.fr

***Abstract*—This paper describes the implementation of a real-time architecture dedicated to obstacle detection in the automotive domain, and more particularly to pre-crash situations. The method, based on stereovision, is of high complexity and can not run in real-time on standard processors. Therefore, the application is accelerated with the use of special purpose hardware; in particular, a highly parallelized disparity engine is presented. A prototype board was realized, which achieves a performance of 460 GOPS and computes the application at the rate of 22 frames per second, thus reaching road safety constraints.**

I. Introduction

The Advanced Driver Assistance Systems (ADAS) goal is to provide more security and comfort for road users. Usual ADASs comprise driver drowsiness warning, lane departure detection, road signs identification, vulnerable human (such as cyclists or pedestrians) detection and avoidance, and at last, obstacle detection and pre-crash triggering. These applications are mostly based on image processing using one or several image sensors: in particular, stereovision allows to extract the depth of a scene in order to analyze it. Some ADAS methods are based on the recognition of the scene elements, such as road signs, pedestrians, or cars. In this article we will only focus on pre-crash systems that can detect close obstacles at high speed, without the need of identifying them.

Embedded ADASs inside vehicles are subject to stringent constraints, such as safety, robustness, real-time execution and power consumption. The constraints that we consider in this article are the real-time execution of an obstacle detection application: the road security implies a high reactivity, well under one second, so that brake signals can be sent to actuators as soon as possible, and a high frame rate of 15 to 30 frames per second.

The next section of this article presents a quick overview of existing driver assistance systems. In section III the obstacle detection algorithm is described. Then, section IV discusses the major processing stages and the resulting partitionning. The following section V exposes the harware architecture that will execute this application. The hardware prototype system is then presented in section VI. Finally in section VII validations are given.

II. Related work to systems for obstacle detection

Many systems exist for measuring the depth of a scene, which are based on radar, ultrasound or time-of-flight. In our work, we consider only vision based methods for short range pre-crash applications.

Several processors exist on the market that are dedicated to automotive applications, such as IMAPCAR[1] and EYEQ[2], which at the moment do not provide dedicated stereovision capabilities. IMAPCAR is a SIMD processor running at 100 MHz, composed of 128 processing elements and embeds 300 Ko of memory, enough to store a VGA frame (640 columns by 480 lines). A stereovision application has been ported on an earlier version of this processor, called IMAPVISION [3]: 100 ms were necessary to compute the depth map of regions of 128 by 128 pixels. Concerning EYEQ, a second version will be available in 2009 that includes a stereovision engine.

Lots of experiments are conducted for real-time applications using an embedded hardware running lower complexity stereovision algorithms, and non real-time applications using a PC. As an example, [4] is a SIMD implementation of stereo algorithms using the SIMD capabilities offered by the multimedia extensions of general purpose processors, such as the SSE2 instruction set of the Pentium family. Other prototyping platforms are studied, such as [5], which presents an architecture for obstacle detection based on stereovision and the V-disparity algorithm. The platform consists of 30 homogeneous processors (MIPS R3000) in one multicore chip. Lastly, Deepsea [6] is a processor that can achieve a bandwith of 200 frames by second on 512 by 480 pixels, using a simple correlation function.

III. Obstacle detection algorithm

The image sensors used are automotive grade sensors ECK100 from Sensata Technologies [7] with progressive scan and a maximum refresh rate of 30 frames per second (fps). The sensors output 12 bits grey data or 8 bits color planes at the VGA resolution. These sensors have a high dynamic range of 120 dB that allows real road operating conditions, such as tunnels, night, bright days without saturating the images.

The first part of the algorithm consists of computing, off-line, an index of rectification, then rectifying input images on the fly using this index. The rectification is a geometrical interpolation which results in two images for which the pixels of a horizontal line are in the same image line, called the epipolar line.

The second part of the algorithm carries out the disparity map. The disparity calculation is based on the best matching of a pixel neighborhood between the reference image (arbitrarily the left one) and a search zone on the other one (see Figure 1). The search zone on the right image is centered on the same line, and corresponds to the scan of the correlation window over a horizontal segment. The length of this segment is limited by the maximum disparity that is geometrically possible, 256 in our case. The correlation window is a 7 by 7 pixels kernel; the sizing of this window is mandatory for the quality of the algorithm and is a compromise between the computational load, the ability to detect small obstacles and the precision of the obstacle contouring.

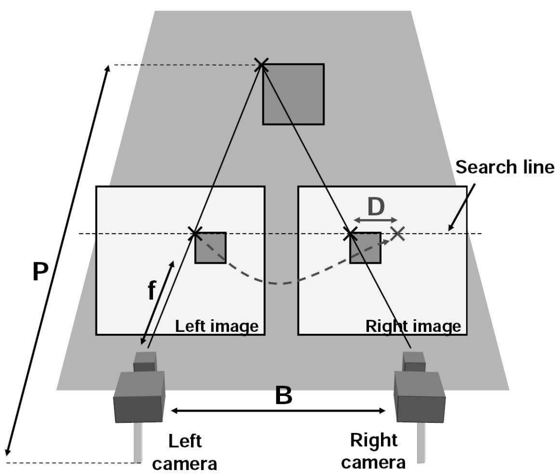

Figure 1. Principle of the disparity calculation: a point of interest (here the corner of a box), is searched on the same line into both images by correlating a 7 by 7 pixels neighborhood. The depth P is given by calculating B.f/D, f beeing the focus distance and B the base between the two cameras.

Several correlation methods are used for computing the dissimilarity score between each pair of windows; the Zero-mean Sum of Square Differences (or ZSSD), shown in Table I, was selected for its good balance between computational load and its robustness to the different illuminances of the two stereo images. For each pixel (u,v) of the reference window N(x,y), the mean illuminance from each window (I1 for the left one and I2 for the right one) is subtracted from the respective pixel value, then the difference of these two pixels is squared in order to boost the differences between the windows. All these values are summed over the window to create the ZSSD score. Then for each left window, the smallest score (the best correspondance) out of the 256 computed ones is selected as the disparity value at this position.

$$S_{ZSSD} = \sum_{(u,v)\in N(x,y)} \left((I_1(u,v) - \bar{I}_1) - (I_2(u+d,v) - \bar{I}_2)\right)^2$$

Table I
FORMULA OF THE ZSSD SCORE.

Two enhancement passes are applied in order to get better results. First, as the correlation method leads to poor results for low textured regions (lot of errors due to homogeneous correlation values), a gradient filter is applied on the incoming images: this filter is a simple edge detector that is not expensive in term of computing power and it stresses the border of objects or signs on the road.

Secondly, an optional feature of the algorithm is to tilt the left window by an angle so that inside the window the road texture is viewed with the same angle from the left and right cameras. This option allows a better detection of the road, under the hypothesis that the road is plan; but for obstacles at the camera height, this correction is wrong; thus when using this option, the best correlation score for both the normal and the tilted reference window must be computed in order to give the disparity at this position.

Finally, the last part of the algorithm is the obstacle detection. This step is performed in a v-disparity space, where the disparities calculated at the preceding step are cumulated in a 2D space: the x-axis are the disparity values and the y-axis are the lines of the source image. In this space (an example is shown in Figure 2), the road is represented as a line that can be extracted thanks to a Hough transform [8] or a Ransac[9]. Any object over the road surface will be seen as a vertical segment in the v-disparity space, and the crossing point of this segment and the road line determines precisely the distance of the targeted object. In the resulted image shown in Figure 7, the boxes that delimitate the objects are set vertically thanks to the v-disparity method, and horizontally with the u-disparity, which is exactly the same method but with the y-axis representing the columns of the source image. Further details on this kind of application can be found in [10].

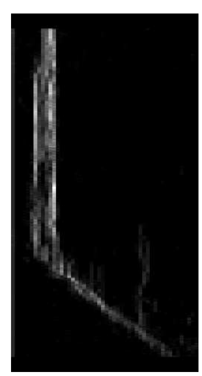

Figure 2. V-disparity space of a road scene: the diagonal bar is the road representation, and the vertical segments are obstacles.

IV. HW/SW PARTITIONING

Table II shows the profiling result of the obstacle detection application on a standard PC, with a non real-time performance of two images per second. This algorithm has a very high level of complexity, thus its implementation on an embedded system requires a careful study of the computational needs, in order to accelerate the algorithm and achieve real-time performance.

Function	Time (ms)	Distribution
Rectification	96.0	18.8%
Optical distortion	50.0	9.8%
Image reduction	3.2	0.6%
Movement prediction	0.01	0.001%
Disparity	318.64	62.3%
Moving estimation	1.3	0.25%
Obstacle detection	42.0	8.2%
Total obstacle detection	**511.15**	100%

Table II
OBSTACLE DETECTION APPLICATION PROFILING ON A PENTIUM 4 HT AT 2.4 GHZ.

First of all, the rectification and the optical distortion correction need floating point arithmetics. Consequently, their hardware acceleration would generate an area-consuming solution. Moreover, the obstacle detection part of the algorithm is an irregular processing since it only consists of complex control. To execute the software part of the algorithm, we looked for the most powerful floating-point Digital Signal Processor (DSP) available on the market, which is the TigerSharc ADSP-TS201S from Analog Device [11]. It can deliver 4800 MMACs at 600 MHz and has a 24 Mb of internal scrathpad memory. Moreover, it only needs 4 cycles to access the external memories, which helps to reduce the usual memory bottleneck in image processing.

Then, the profiling of the application shows that 62% of the application time is spent in the computation of the disparity. While this function is particularly regular, its parallelization is therefore possible and a specific hardware component must be designed to accelerate this critical part.

Finally, for the other low-intensive parts of the application, a hardware acceleration would bring too little benefits and a programmable processor is consequently the best choice.

The execution time on the TigerSharc processor of the software parts of the algorithm is presented in Table III. The rectification and distortion correction stage is accelerated by a factor of 4 compared to the PC version; the reduction and prediction stage is slightly accelerated, but is not relevant because of its small impact; finally the detection stage is not accelerated and is the limiting factor of the algorithm. Most important is the fact that these tasks are independant and can be pipelined over different processors, running in parallel. The largest stage of the algorithm, named “disparity processing”, requires specific acceleration that is described in the next section.

Function	Exec. Time (ms)
Rectification & optical distorsion	37
Image reduction & Movement prediction	2
Moving estimation & Obstacle detection	40-45

Table III
IMPLEMENTATION RESULTS ON THE TIGERSHARC DSP FOR ONE IMAGE (LEFT OR RIGHT). THE OBSTACLE DETECTION DOES NOT TAKE INTO ACCOUNT THE DISPARITY COMPUTATION.

V. DISPARITY PROCESSOR

The disparity function is a very compute-intensive function, with a simple arithmetic and a regular processing.

A dedicated hardware accelerator is best suited to accelerate such functions, but as the fabrication of an ASIC component of 15 Mega-gates is too costly and time-consuming, we decided to look for the most powerful and complex FPGA device in the Xilinx reconfigurable family at the moment: the Xilinx Virtex 4 FX140 platform [12] was selected for its numerous integrated DSP blocks. The largest parallel architecture that can be synthesized on this FPGA was evaluated: we found that at most 8 scores can be computed simultaneously per each pixel of the left image.

The disparity processor can be divided into two parts: the control and the computation part. The control part selects the necessary pixels for the computation of the disparity and manages the whole disparity computation process. As shown in Figure 3, the control part integrates a bidirectional circular buffer of 6 lines of pixels in order to maximize the data locality. To compute 8 scores, we need the selection of 8 right windows and a left window. The tilt of the left window is optional and is a parameter of the architecture. Besides, during the computation of the current line, additional input buffers are used to enable the load of the next incoming line, and the means of windows and the gradient are computed along the next line.

At each cycle, 8 means and gradients are simultaneously computed in the right image. Because we consider 7 by 7 kernels and that some pixels are common, the control part provides 14 by 7 pixels, 8 means and 8 gradients to the computation part at each cycle with 8 cycles latency. Then, it moves from 8 pixels on the right and provides another set of values to the computation part. This is repeated 32 times to process the 256 pixels of the search window. As the search window length depends on the distance between the current pixel and the right border of the right image, the control part automatically computes these border effects and manages the computation part. Finally, the output memory controller gets back the computed scores, which manages the minimum of these scores and applies the gradient selection.

The computation part brings together all of the architecture operators. It mainly computes the previously presented ZSSD function. The architecture of the left and right mean computation is fully-parallel and the division by 49 (7 by 7 pixels) is simplified by a multiplication with the value 1337 followed by a 16-bit right shift (actually this is a division by

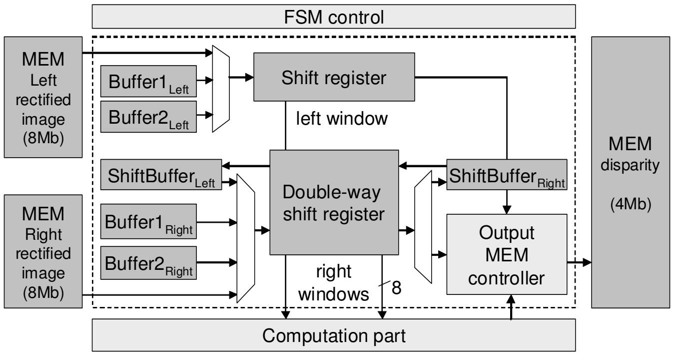

Figure 3. Control part of the disparity processor.

49.017, that is to say a 0.03% deviation). Following the ZSSD formula, these means are substracted from the pixels values, and the right and left results are again substracted. Then, each of these 392 results (49 by 8) are squared.

The square operator is obtained by two different techniques. First, we use 189 embedded DSP blocks (that include fast multipliers) out of the 192 available into the selected FPGA. Then, for the 203 remaining square operators, we use another approach in order to minimize the number of slices. By exploiting symmetry properties of numerical values and the incremental computation of squares presented in [13], it is possible to drastically decrease the hardware cost of a square operator. The corresponding hardware architecture is derived from a simple conversion of existing incremental scalar multipliers. With this approach, our 18-bit square operator needs only 112 slices with a 3 cycles latency, instead of 193 slices with a parallel multiplier (-42%).

The last step to get the score consists in summing 8 times the 49 squared values (3 cycles latency). Then, these operations are done 32 times and the 256 scores are compared together to find the minimum score according to the gradient values. This last step needs 7 cycles. Finally, the position in the search window corresponding to this score (the disparity), the score and the gradient are returned to the control part.

Thus, once the pipeline has been initiated, the architecture implemented into one FPGA component is able to deliver 8 scores per cycle and therefore a disparity value for each left pixel in only 32 cycles. For a 640-pixel width or a complete left pixel line, we need only 18,432 cycles. As shown in Table IV, because of the very high resource utilization, we used the Xilinx PlanAhead hierarchical floorplanner and we manually contributed to the place and route step. In this way, we reached a 100 MHz frequency. Therefore, the time spent to execute a whole line is about 184 μs. To respect real-time conditions, it is possible to cut the image according to its height and to implement the architecture as many times as we need. Indeed, the processing of each line is independent.

In the next section, we will present the whole architecture that has been implemented to support the application.

Logic	Utilization
I/Os	318 (41%)
Slices	53,880 (98%)
Slices FF	73,348 (58%)
4-input LUTs	87,663 (69%)
DSPs	189 (98%)
RAMB16s	450 (81%)

Table IV
SYNTHESIS RESULT OF THE DISPARITY PROCESSOR (XILINX VIRTEX 4 FX140-10

VI. ARCHITECTURE DETAILS

The targeted embedded platform for the application is a stereovision system. We use two logarithmic CMOS cameras and a synchronization box. We directly use digital output signals and interface them through an FPGA dedicated to image adaptation. This FPGA shapes input images to prepare the rectification. According to Figure 4, the acquired images are both saved in two output shared memories. Each of the 32-bit memories depicted in this figure is a two-bank dual-port memory of 512KB. A dual-port memory is used to avoid conflicting accesses between the input and the output data flows.

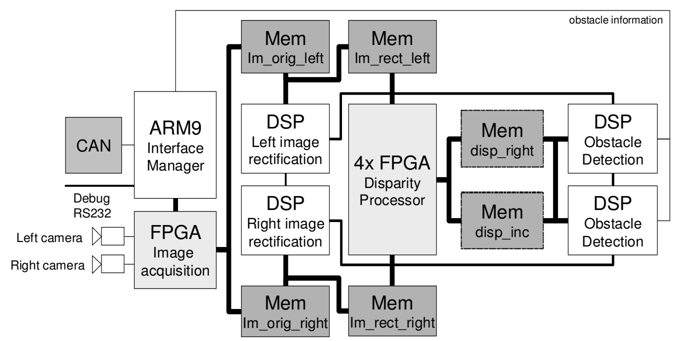

Figure 4. Synoptic view of our platform

As shown in Table III, even if the chosen DSP is powerful, the time spent for all this software processing remains important. However, our final system must reach between 15 and 22 fps with the highest reactivity. The only possible approach consists in pipelining the application to keep a good frame rate. In addition, the pipeline length must be as short as possible to reduce the latency. Hopefully, it is possible to duplicate the processors to simultaneously process each software part on the left and right input images. Indeed, each image processing is independent. Some information must be shared to validate and synchronize the rectification or the obstacle detection, but this has a very low inpact on parallelism.

According to our implementation results on the TigerSharc, the longest processing stage is the moving estimation with the obstacle detection. Therefore, the stage length must be equal to 45 ms and our pipeline must consist of three stages. The first pipeline stage is the rectification, the optical distorsion, the image reduction and the movement prediction. The second stage is the disparity, and the third stage carries out the moving estimation and the obstacle detection. Four TigerSharc DSPs are used for the first and last pipeline stages. Because the disparity execution of half an image needs 42 ms and the disparity for both a normal and a slopped left image must be computed, we integrated 4 FPGAs for the disparity computation stage.

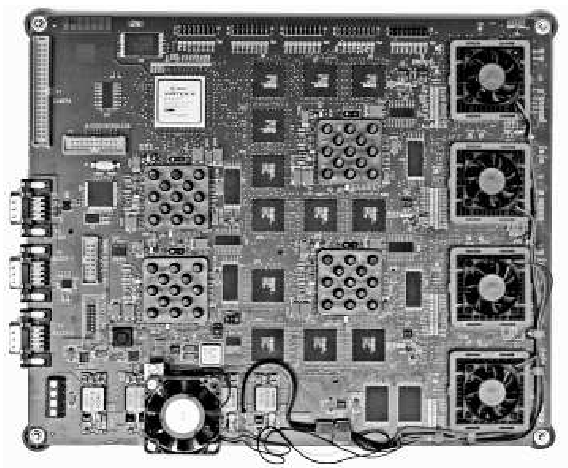

Figure 5. Obstacle detection platform board

In addition, a CAN interface is present to communicate with the driver and to send the confidence rating of the information, the number of detected obstacles and their 3D position. An RS232 serial link is also used as a debug interface to communicate with a host PC. A debug application has been carried out under Visual C++ to drive the board and manage all test and debug protocols. All these interfaces are managed by an ST microcontroller STR912 based on the ARM 9 [14]. Figure 5 shows the board of our complete platform: it is a 14-layer PCB (printed circuit board). The peak performance of our embedded system is about 460 GOPS and the board dissipates about 100 W when running the application at full rate.

VII. Validations

As shown in Figure 6, the left and right image rectifications begin whereas the image acquisition is not yet finished. All the tasks are activated as soon as input data are sufficient to begin the process. This is very important because the distance covered by the vehicle before detecting an obstacle depends on the latency of the whole processing. In the same manner, the disparity computation begins as soon as the 7 first lines of each image are ready.

Finally, the minimum time to get the obstacle information is about 135 ms (3 successive image acquisitions) i.e. 3.37 meters at 90 km/h. This represents about 10 times benefit compared to a human reaction latency. Then, we get an information every 45 ms. A detection result example is depicted in Figure 7: the wall on the left part which is a close object is not classified as an obstacle and all cars are identified as obstacles. This result is robust to various road conditions and illuminations.

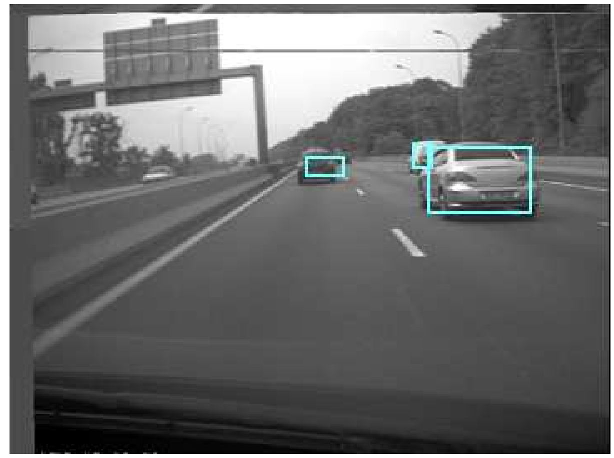

Figure 7. Detection result: obstacles on the road surface are identified into the rectangles

VIII. Conclusion

This article presents a prototype system that achieves real time performance for a high-end obstacle detection application. Because of its high complexity, the application was partitionned onto several DSP processors. The disparity calculation, which is the most critical stage of the application because of its high computational load, is parallelized into several FPGA components: a dedicated accelerator was realized to run the disparity scores.

After the prototype was assembled and debugged, the obstacle detection was validated on the board with various real road conditions stereo videos; the prototype was also tested inside parking lots, and we still have to embed it into a car for demonstration purposes. With the technological nodes progress, a large Virtex-6 FPGA would be now enough to handle all the disparity calculation, so that the board would be simpler and dissipate much less power than the current implementation.

Acknowledgment

The authors would like to thank Pierre-Emmanuel Viel and Frédéric Pasquet for their active participation in HW and SW developments, and Charly Bechara for careful reading. Part of this research was funded by the French Innovation and Research Program for Terrestrian Tranportations (PREDIT).

References

[1] Website. http://www.necel.com/applications/en/automotive/imapcar.
[2] Website. http://www.mobileye.com.
[3] G. Kraft and P. Jonker. Real-time stereo with dense output by a simd-computed dynamic programming algorithm. In *PDPTA '02: Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 2002.

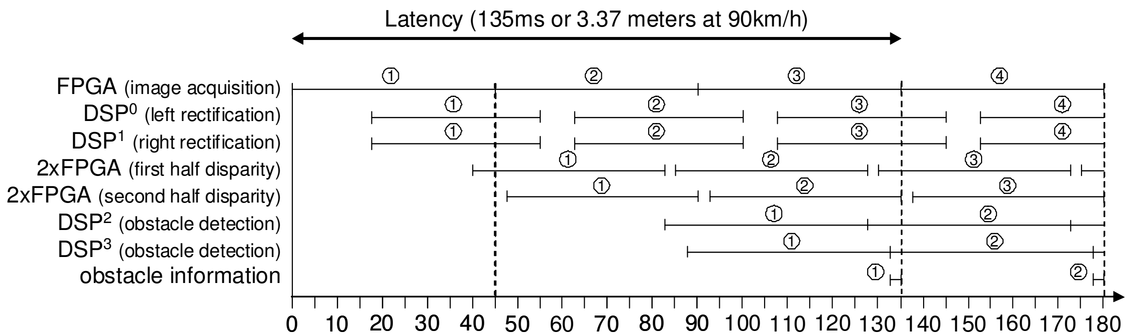

Figure 6. Time recorder and task distribution of the application onto the platform.

[4] W. VanDer Mark and D. M. Gavrila. Real-time dense stereo for intelligent vehicles. *IEEE Transactions on Intelligent Transportation Systems*, March 2006.

[5] A. Greiner, F. Petrot, M. Carrier, M. Benabdeni, R. Chotin-Avot, and R. Labayrade. Mapping an obstacles detection, stereo vision-based, software application on a multi-processor system-on-chip. *IEEE Intelligent Vehicle Symposium*, June 2006.

[6] J. I. Woodfill, G. Gordon, and R. Buck. Tyzx deepsea high speed stereo vision system. In *CVPRW '04: Proc. of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, 2004.

[7] ECK100 evaluation camera kit. Technical report, Sensata Technologies, July 2007.

[8] P. V. C. Hough. Machine analysis of bubble chamber pictures. In *Proc. Int. Conf. High Energy Accelerators and Instrumentation*, 1959.

[9] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM*, 24:381–395, June 1981.

[10] N. Hautiere, R. Labayrade, M. Perrollaz, and D. Aubert. Road scene analysis by stereovision: a robust and quasi-dense approach. In *ICARCV*, pages 1–6. IEEE, 2006.

[11] ADSP-TS201 TigerSHARC processor Hardware Reference. Technical Report v1.1, Analog device, December 2004.

[12] Virtex-4 FPGA User Guide. Technical Report v2.6, Xilinx, December 2008.

[13] S.G. Smith. Incremental computation of squares and sums of squares. *IEEE Transactions on Computers*, 38(9):1325–1328, September 1989.

[14] UM0216 - STR91xF ARM9-based microcontroller family Reference Manual. Technical Report rev1, ST Microelectronics, April 2006.