Part One
Verilog HDL

1

Introduction

This chapter addresses the status of the vision architectures in four major fields: computer architectures, vision algorithms, vision devices, and design methodologies. Computer architecture, which is characterized by serial, parallel, pipelined, and concurrent computation, must be tuned to the underlying computational structures – parallel, iterative, and neighborhood computation – that are used in intermediate computer vision. Vision algorithms, which have evolved from heuristic methods to generic structured algorithms at each level of computer vision from low level to high level, must be investigated in terms of computational structures. The vision devices, ranging from CPUs to very-large-scale integration (VLSI) chips, must be investigated in terms of their flexibility and computational complexity. Finally, the design flow from vision to chip, which is not well-defined, must be defined and delineated using a general methodology.

1.1 Computer Architectures for Vision

Vision architectures are special forms of more general computer architectures. In the early 1970s, a general point of view on computer architecture was to see it as an information flow of data and instructions into a processor (Figure 1.1). Flynn's taxonomy (Flynn 1972) is the most universally accepted method of classifying computer systems. The instruction stream is defined as the sequence of instructions performed by the processing unit. The data stream is defined as the data traffic exchanged between the memory and the processing unit. According to Flynn's classification, the instruction stream and data stream can both be either singular or multiple in nature.

Flynn's taxonomy classified architectures into single instruction single data stream (SISD), Single instruction multiple data stream (SIMD), multiple instruction single data stream (MISD), and multiple instruction multiple data stream (MIMD). In this classification system, an SISD machine is the traditional serial architecture where instructions and data are executed serially. This is often referred to as the Von Neumann architecture. An SIMD machine is a parallel computer, where one instruction is executed many times with different data in a synchronized manner. An extreme example is the *systolic array* (Kung and Leiserson 1980; Kung 1988; Leiserson and Saxe 1991). In an MISD machine, each processing unit operates on the data independently via independent instruction streams. This computational technique is also called *pipelining*. A set of pipelined vectors is referred to as a superscalar. An MIMD machine is a fully parallel machine where multiple processors execute different instructions and data independently.

This concept can be formalized by a set of state machines, such as the Moore machine or the Mealy machine. Suppose a processing element (PE) in a state Q_k receives date D_k and instruction I_k and

Architectures for Computer Vision: From Algorithm to Chip with Verilog, First Edition. Hong Jeong.

Companion Website: www.wiley.com/go/jeong

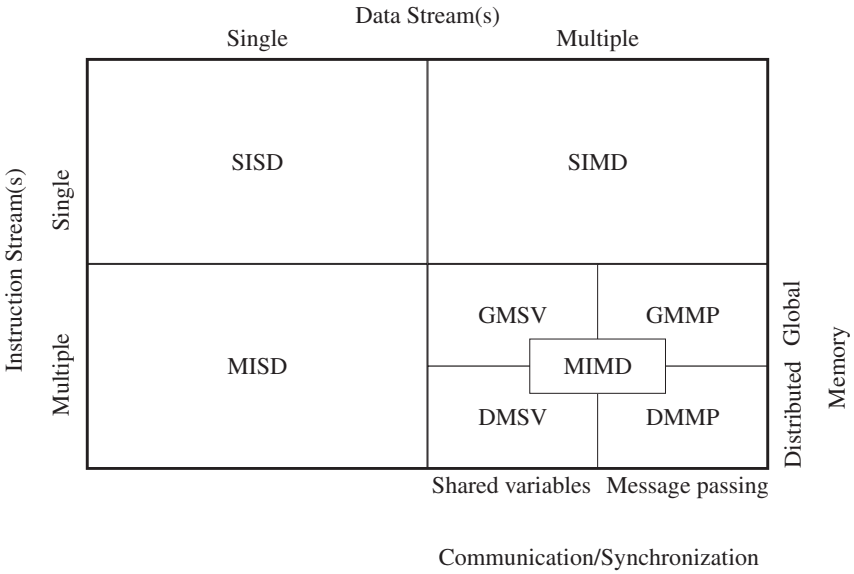

Figure 1.1 Flynn–Johnson taxonomy of computer architectures

generates output O_k, $(k = 0, 1, \ldots)$ according to the state transition $T(\cdot)$ and output generation $H(\cdot)$. Then, the SISD machine is modeled by a Mealy machine:

$$\begin{cases} Q_{k+1} = T(Q_k, D_k, I_k), \\ O_k = H(Q_k, D_k, I_k), \quad k = 0, 1, 2, \ldots. \end{cases} \tag{1.1}$$

Other machines can be modeled by a set of PEs by combining data and instructions in various ways. As such, an SIMD machine is modeled as a set of identical PEs, operating on different data but controlled by the same instruction set:

$$\begin{cases} Q_{k+1}^l = T\left(Q_k^l, D_k^l, I_k\right), \\ O_k^l = H\left(Q_k^l, D_k^l, I_k\right), \quad \forall l \in [0, N-1], \end{cases} \tag{1.2}$$

where N denotes the number of PEs. The MISD machine is modeled as

$$\begin{cases} Q_{k+1}^l = T^l\left(Q_k^l, O_k^{l-1}, I_k^l\right), \\ O_k^l = H^l\left(Q_k^l, O_k^{l-1}, I_k^l\right), \quad \forall l \in [0, N-1], \end{cases} \tag{1.3}$$

where the data input is $O_k^{l-1} = D_k^l$ and the output is O_k^{N-1}. The MIMD machine is a set of different machines:

$$\begin{cases} Q_{k+1}^l = T^l\left(Q_k^l, D_k^l, I_k^l\right), \\ O_k^l = H^l\left(Q_k^l, D_k^l, I_k^l\right), \quad \forall l \in [0, N-1]. \end{cases} \tag{1.4}$$

Nowadays, the MIMD category includes a wide variety of different computer types and as a result, taxonomies have been added to the MIMD class. The Flynn–Johnson taxonomy, which is one of many classification methods, proposed a further classification of such machines based on their memory structure (global or distributed) and the mechanism used for communications/synchronization (shared variables or message passing).

A global memory shared variable (GMSV) machine is a machine with shared memory multiprocessors. A global memory message passing (GMMP) machine is a machine that uses global memory and message passing. This type of machine is rarely used. In distributed memory shared variables (DMSV) machines, memory resources are distributed to the processors and the variables are shared. In distributed memory message passing (DMMP) machines, memory resources are distributed and message passing is used. In this classification system, DMSV and DMMP machines are loosely coupled machines, and GMSV and GMMP machines are tightly coupled machines. In addition to data and instructions, this classification system introduces two more variables: memory M and message m,

$$\begin{cases} Q_{k+1}^l = T^l\left(Q_k^l, D_k^l, I_k^l, M_k^l, m_k^l\right), \\ O_k^l = H^l\left(Q_k^l, D_k^l, I_k^l, M_k^l, m_k^l\right), \quad \forall l \in [0, N-1]. \end{cases} \tag{1.5}$$

The differences between the four machine types are based on the various combinations of the memory M, shared by a set of processors, and the messages m, passed between a set of processors.

Modern computer processors, such as central processing units (CPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs), embedded processors (EPs), and graphics processing units (GPUs), tend to evolve into huge systems that use more computational resources – more pipelines, multi cores, more shared memory, more distributed memory, and multi-threading.

For computer vision, the computational architectures depend on the levels of vision (early, intermediate, and high-level vision) and the algorithms (relaxation, graph cut, belief propagation, etc). Starting from serial algorithms for general computers, we usually search for more structured algorithms and architectures for better implementation. The passage from low to intermediate vision is characterized by high-levels of resource usage for numerical computations and memory space, and the structures that are used are usually parallel, repetitive, or local neighborhood structures. High-level vision, on the other hand, is characterized by high-level resource usage for symbolic computation, and the structures that are used are usually concurrent, heterogeneous, modular, and hierarchical computational structures. The intermediate level is closer to the regular, SIMD, and MISD architectures and the high level is closer to the general, SISD, and MIMD architectures. From early to intermediate level, the computational structures are characterized by pixel or local neighborhood computations, recursive updating, and scale-dependent (hierarchical or pyramidal) and parallel computations. We will concentrate on designing architectures for such early- to intermediate-level operations that are parallel and iterative and rely on neighborhood computations.

Similar to general computer architecture, vision architecture can be modeled by state machines. Assume an image plane $\mathcal{P} = \{(x, y) | x \in [0, N-1], y \in [0, M-1]\}$ and an image defined over $\mathcal{P}$, $I = \{I(p) | p \in \mathcal{P}\}$. A neighborhood N_p is a set of (topologically) connected pixels around $p \in \mathcal{P}$, and a window A_p is a set of pixels around $p \in \mathcal{P}$. A typical operation is to update the state of a pixel using the neighborhood values along with the image input. The operation is repeated for all pixels until the values converge. The outputs are the pixel states in equilibrium. The state equation for parallel iterative neighborhood computation is defined as follows:

$$\begin{cases} Q^{(k+1)}(A_p) = T(Q^{(k)}(N(A_p)), I(A_p)), & k = 0, 1, \ldots, K-1, \\ O(A_p) = Q^{(K-1)}(A_p), & \forall p \in \mathcal{P}, \end{cases} \tag{1.6}$$

where the superscript denotes the iteration and K denotes the maximum number of iterations. The parallelism can be achieved by the set of window A_p. The iteration means the recursive computation of the states. The neighborhood computation is represented by the local neighborhood function $N(A_p)$. This is a basic computational structure in low to intermediate vision that generally uses large amounts of spatial and temporal resources. The run-time is $O(MN\mathcal{A}\mathcal{N}K)$, which is proportional to the image size MN, the window size A, and the number of iterations. The required space is $O(MN)$, which is proportional to the image size MN.

1.2 Algorithms for Computer Vision

With regards to stereo vision, there are survey papers (Brown *et al*. 2003; Kappes *et al*. 2013; Scharstein and Szeliski 2002; Szeliski *et al*. 2008) and a site (Middlebury 2013), where state-of-the-art algorithms are listed and test datasets are stored.

The algorithms can be categorized into local matching and the global matching methods. Local matching algorithms use matching costs from a small neighborhood of target pixels. Most local matching algorithms adopt a three-step procedure for matching – cost computation, aggregation, and disparity computation. The first and the third steps are common both in local and global matching methods. The aggregation step gathers measured matching costs in pre-defined matching support, whose definition varies depending on the kind of algorithm. On the other hand, global matching algorithms use matching costs from every pixels on the image to determine single-pixel correspondence. Global matching methods commonly consist of the optimization step instead of the aggregation step. In the optimization step, the algorithm aims to search acceptable solution that minimizes or maximizes some kind of *energy function*. Energy functions have various constraints about geometries and appearances of two views.

Though some of the local matching algorithms show acceptable performance based on error rate and processing speed, most top-ranked algorithms are global matching methods, such as belief propagation (BP) and graph cuts (GC), which are based on energy functional models. In general, global matching methods require more computations and more memory than local methods.

The vision algorithms consist of a set of basic general algorithms that can be combined in various manners. Some of the general algorithms are listed in Table 1.1.

Conceptually, the exhaustive search is a kind of benchmark for the global solution, ignoring all the other practical requirements such as time and space requirements. The Gauss-Seidel and Jacobi methods and relaxation algorithm are the fundamental algorithms in iterative optimization methods. The dynamic programming algorithm is an efficient algorithm for divide-and-conquer type problems. The simulated annealing (SA) algorithm is an intelligent sampling strategy for statistical optimization,

Table 1.1 Comparison of major vision algorithms

Technology	Performance	Parallelism	Time	Flexibility
Exhaustive Search	Ultimate goal	No	NP-hard	General
Gauss–Seidel	Low	Serial	Fast	General
Jacobi	Low	Parallel	Fast	General
Relaxation	Medium	Parallel	Medium	General
DP	Good	Parallel	Fast	1D problem only
SA	Good	No	Slow	General
BP	Near best	Parallel	Slow	General
GC	Best	Serial	Slow	General

cf. DP: dynamic programming, SA: simulated annealing, BP: belief propagation, and GC: graph cuts.

which is guaranteed to converge to the global minima if some ideal conditions, such as *generation probability*, *acceptance probability*, and *annealing schedules*, are satisfied. The BP is the result of the long evolution of stochastic relaxation that determines marginal distributions iteratively in a Bayesian tree. The GC algorithm has evolved from max-flow min-cut problems to the current swap move and expansion move algorithms (Boykov *et al*. 2001). Owing to the BP and GC algorithms, the performance of vision algorithms has improved dramatically. Some research has even reported that due to the two algorithms there may be no more margins than a few percent to the global optimum. The two algorithms are general problem solvers but require vast resources for computation and space. The GC algorithm requires global communication that may in turn require serial computation. The BP algorithm is based on MRF and neighborhood computation that may require parallel computation.

There are also common optimization techniques in computer vision fields such as expectation maximization (EM), Bayesian filtering, Kalman filtering, particle filtering, and linear programming relaxation (LPR).

1.3 Computing Devices for Vision

The lifespan of hardware systems is very short compared to the life spans of algorithms and software systems. In spite of the poor documentation and the difficulty of surveying the field, we can get a feel for the state of the field from the survey lists for device types, performance, and trends (Lazaros *et al*. 2008; Nieto *et al*. 2012; Tippetts *et al*. 2011, 2013).

In image processing, the most dedicated devices are FPGAs and ASICs (refer to the books on FPGAs and image processing (Ashfaq *et al*. 2012; Bailey 2011; Gorgon 2013, 2014; Samanta *et al*. 2011). They are fast enough for real-time processing, yet small enough for portability and mass production.

The flow from algorithm to architecture to device is characterized by the constraints and requirements. The algorithms and architectures are closer to concepts, and the device is the reality with a smaller degree of freedom. Therefore, any algorithm, targeted for implementation, must be developed from the beginning so as to satisfy the device constraints.

Although there are numerous types of devices, they can be classified into roughly six categories: Generic CPUs, EPs, DSPs, GPUs, FPGAs, and ASICs. All of these six platforms, except dedicated digital circuits, are common tools for hardware realization. The major differences between these platforms are speed, flexibility, and development time. In addition, there may be other relevant factors such as cost, power consumption, and time required for updates.

The major devices and factors are summarized in Table 1.2. With regards to performance and cost, single-purpose, dedicated ASICs are rated best, and the generic processors, which are targeted for general application, are rated lowest. However, with regards to flexibility, which deals with major updates or modifications, the order is reversed: ASICs are rated lowest and the generic processors are the best.

Table 1.2 Comparison of major devices

Technology	Performance/cost	Time until running	Speed	Flexibility
ASIC	Very High	Very long	Very high	No
FPGA	Medium	Long	High	Low
GPU	High	Medium	High	Medium
DSP	High	Medium	Medium	High
EP	Low	Short	Low	High
Generic CPU	Low to medium	Very short	Very low	Very high

cf. EP: Embedded Processor.

In terms of development time, ASICs take the longest amount of time and the generic processors only require programming time.

We want to explore the pure computational structures of the vision algorithms and use them as much as possible in developing dedicated architectures. Because of this, we exclude algorithms that need sophisticated software control. Devices such as GPU, DSP, EP, and generic processor are heavily dependent on software. This book focuses on the FPGA/ASIC because they are genuinely dedicated hardware solutions that do not require software interventions.

1.4 Design Flow for Vision Architectures

The task of researchers is to define vision problems, explore vision algorithms, and build software or hardware systems. For hardware realization, the vision algorithm can be coded into devices such as embedded processors, GPUs and CPUs with hyper-threading, and streaming SIMD extensions (SSE). ASICs, FPGAs, and programmable logic devices (PLD) are more dedicated devices. The chip design process is separate from the vision algorithms. The algorithms cannot be applied directly to chip design. The vision algorithm is inherently serial and the chip is inherently concurrent.

There must be an intermediate stage between the vision algorithm and the chip design. This stage, called vision architecture, must integrate the vision algorithm, which is mostly serial, into the design architectures, which are concurrent. The overall design flow is illustrated in Figure 1.2.

The design flow consists of the three parts: vision, architecture, and chip design. The vision part means the ordinary vision research and the algorithms for programming. In the vision architecture, the vision algorithms are analyzed in terms of computational structure, and redesigned using processing elements, memory, and connections into architecture. Given the architecture and specifications, the chip

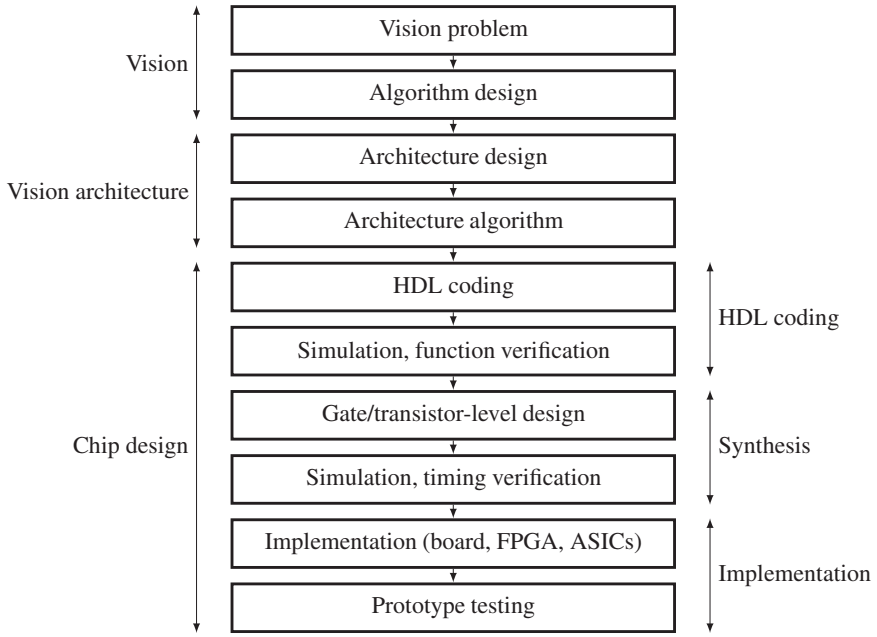

Figure 1.2 Design flow from vision to chip

design progresses sequentially from hardware description language (HDL) coding, to synthesis, and then to implementation. The HDL coding programs the architecture description into the register transfer language (RTL) format. This code is converted into circuits, i.e. net list, in the synthesis stage. Each stage is a loop consisting of design and testing. There are variety of potential realizations for the net list, such as FPGA (i.e. programming) and ASICs (i.e. hard copy) and full custom.

The FPGA design consists of the following stages.

1. Define a new project and enter the design using VHDL or Verilog HDL languages. The design can also be entered using schematic diagrams that can be translated to any HDL.
2. Compile and simulate the design. Find and fix timing violations. Obtain power consumption estimates and perform the synthesis.
3. Download the design to the FPGA using either a parallel port or a USB cable. Designs can also be downloaded via the Internet to a target device.

Once an FPGA design is verified, validated, and used successfully, there is an option to migrate it to a structured ASIC. This option is known as hard copy. Using hard copy, FPGA design can be migrated to a hard-wired design removing all configuration circuitry and programmability so that the target chip can be produced in high volume. Hard-copied chips use 40% less power than FPGAs and the internal delays are reduced.

Vision algorithms can be implemented using computational structures that are either serial or parallel and either iterative or recursive. Therefore, we have to reinterpret the vision algorithms in terms of architectural modalities and describe them in architectural terms. To expand the vision research to architecture, vision engineers have to learn two principles:

- Architecture design: Given an algorithm, analyze the computational structures in terms of data structures – memory, queue, stack, and processing – and express them in a hardware algorithm.
- HDL coding: Code the algorithm in HDL and test.

The first task is to convert vision algorithms into hardware algorithms. Vision algorithms are free from the constraints of a specific realization and as a result can rely on generic programming. The hardware, on the other hand, must be described in terms of memory, data structure, communication, and control. Likewise, the vision algorithm can be coded in a high-level programming language, but the architecture must be designed using the lower-level HDL programming language.

There are integrated programming environments such as Quatus by Altera, Inc. and ISE by Xilinx, Inc. They are the software tools for synthesis and analysis of HDL designs, which enable the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device. We will use Quartus tools as a reference design tool and Verilog as a design language. Verilog is preferred in this book because it closely resembles C, which is one of the most prominent languages in vision research.

Problems

1.1 [Architecture] Explain the advantages and disadvantages of SISD. How are problems addressed in DSP?

1.2 [Architecture] What are the pros and cons of SIMD and MISD?

1.3 [Architecture] What are the problems with shared and distributed memory systems?

1.4 [Architecture] For GMSV and GMMP, how must Equation (1.5) be defined?

1.5 [Architecture] Express an average operation (one-pass) in terms of Equation (1.6).

1.6 [Architecture] Express a difference operation (one-pass) for $\frac{\partial}{\partial x}I$ and $\frac{\partial}{\partial y}I$ in terms of Equation (1.6).

1.7 [Devices] Explain the processors – CPU, DSP, EP, GPU, FPGA, and ASIC – in terms of their core functions and specifications. In addition, what are the state-of-the-art technologies for each device?

1.8 [Algorithm] Understanding vision algorithms in terms of computational structure is very important. Name some vision algorithms that make use of (1) one-pass/multi-pass, (2) neighborhood operation, (3) iteration, and (4) hierarchical structures.

1.9 [Algorithm] The labeling problem assigns labels $l \in [0, L-1]$ to the pixels in $\mathcal{P} = \{(x, y)|x \in [0, N-1], y \in [0, M-1]\}$. How many cases are there in labeling? If the labels of the neighbors are equal, how many cases are there? Discuss the role of constraints in the labeling problem.

1.10 [Design] Download Altera Quartus or Xilinx ISE and acquaint yourself with the tools. What are the major functions of these tools? How can vision algorithms be designed into circuits?

References

Ashfaq A, Hameed T, and Mehmood R 2012 *FPGA Based Intelligent Sensor for Image Processing: Image Processing with FPGA*. Lambert Academic Publishing.

Bailey DG 2011 *Design for Embedded Image Processing on FPGAs*. Wiley-IEEE Press.

Boykov Y, Veksler O, and Zabih R 2001 Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.* **23**(11), 1222–1239.

Brown M, Burschka D, and Hager G 2003 Advances in computational stereo. *IEEE Trans. Pattern Anal. Mach. Intell.* **25**(8), 993–1008.

Flynn M 1972 Some computer organizations and their effectiveness. *IEEE Trans. Computer* **C-21**, 948–960.

Gorgon M 2013 *FPGA Imaging: Reconfigurable Architectures for Image Processing and Analysis*. Springer.

Gorgon M 2014 *FPGA Imaging: Reconfigurable Architectures for Image Processing and Analysis*. Springer.

Kappes JH, Andres B, Hamprecht FA, Schnorr C, Nowozin S, Batra D, Kim S, Kausler BX, Lellmann J, Komodakis N, and Rother C 2013 A comparative study of modern inference techniques for discrete energy minimization problems *EMMCVPR 2013*.

Kung H and Leiserson C 1980 Algorithms for VLSI processor arrays In *Introduction to VLSI Systems* (ed. Mead C and Conway L) Addison-Wesley Reading, MA pp. 271–291.

Kung S 1988 *VLSI Array Processors*. Prentice-Hall, Englewood Cliffs, NJ.

Lazaros N, Sirakoulis GC, and Gasteratos A 2008 Review of stereo vision algorithms: from software to hardware. *International Journal of Optomechatronics* **2**(4), 435–462.

Leiserson C and Saxe J 1991 Retiming synchronous circuitry. *Algorithmica* **6**(1), 5–35.

Middlebury U 2013 Middlebury stereo home page http://vision.middlebury.edu/stereo (accessed Sept. 4, 2013).

Nieto A, Vilarino D, and Sanchez V 2012 *Towards the Optimal Hardware Architecture for Computer Vision* InTech chapter 12.

Samanta S, Paik S, and Chakrabarti A 2011 *Design & Implementation of Digital Image Processing using FPGA: FPGA-based digital image processing*. Lambert Academic Publishing.

Scharstein D and Szeliski R 2002 A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision* **47**(1-3), 7–42.

Szeliski RS, Zabih R, Scharstein D, Veksler OA, Kolmogorov V, Agarwala A, Tappen M, and Rother C 2008 A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. *IEEE Trans. Pattern Anal. Mach. Intell.* **30**(6), 1068–1080.

Tippetts BJ, Lee DJ, Archibald JK, and Lillywhite KD 2011 Dense disparity real-time stereo vision algorithm for resource-limited systems. *IEEE Trans. Circuits Syst. Video Techn* **21**(10), 1547–1555.

Tippetts BJ, Lee DJ, Lillywhite K, and Archibald J 2013 Review of stereo vision algorithms and their suitability for resource-limited systems http://link.springer.com/article/10.1007%2Fs11554-012-0313-2 (accessed Sept. 4, 2013).