Proceedings of the IEEE Computer Architectures for Machine Perception, Villa Olmo, Como, Italy, September 18-20, 1995

Real-Time Quantized Optical Flow

Ted Camus*
National Institute of Standards and Technology†

Abstract

Algorithms based on the correlation of image patches can be robust in practice but are computationally intensive due to the computational complexity of their search-based nature. Performing the search over time instead of over space is linear in nature, rather than quadratic, and results in a very efficient algorithm. This, combined with implementations which are highly efficient on standard computing hardware, yields performance of over 5 frames per second on a scientific workstation. Although the resulting velocities are quantized with resulting quantization error, they have been shown to be sufficiently accurate for many robotic vision tasks such as time-to-collision and robotic navigation. Thus, this algorithm is highly suitable for real-time robotic vision research.

1 Introduction

Currently two major limitations to applying vision to real robotic vision tasks are robustness in real-world, uncontrolled environments, and the computational resources required for real-time operation. For example, D. Touretzky et al. (1994) note *"Even something as simple as calculating real-time optic flow requires more processing power than is practical for a mobile robot."* ([19]). In particular, many current robotic visual motion detection algorithms (optical flow) may not be suited for practical applications such as collision detection on a mobile robot because they either require highly specialized hardware or up to several minutes on a scientific workstation. For example, [21] quotes 4 minutes on a Sun workstation and 10 seconds on a 128-processor Thinking Machines CM5 supercomputer. Although good quantitative results are reported, such an algorithm would take approximately 16 years for the 1000-fold increase in performance necessary for real-time rates on an ordinary workstation, given that computer processing power increases approximately 54% per year [18]. In addition, many such algorithms depend on the computation of first and in some cases higher numerical derivatives, which are notoriously sensitive to noise. In fact the current trend in optical flow research is to stress accuracy (often under idealized conditions such as modeling the noise as Gaussian) over computational resource requirements and robustness against noise, both of which are essential for real-time robotics. Another trend is to compute only *normal* flow [14] instead of the true optical flow. Although normal flow can be computed much more easily than full optical flow and may have sufficiently robust statistical properties for many useful tasks (e.g., [13, 9]) there will be many cases where full flow is preferable, if it can be computed efficiently and robustly.

Many techniques for optical flow exist (e.g., [10, 21]; see also [2, 16] for reviews and discussions of several techniques). Although many of these techniques can perform very well for certain sequences of images, there are very few that are currently able to support real-time performance without special-purpose hardware. One obvious reason calculating optical flow is so computationally intensive is that images are composed of thousands of pixels (which often motivates massively parallel implementations). One option is to only calculate optical flow at areas of high contrast where the flow measurements are more reliable. However, many applications of optical flow require dense outputs which are difficult to compute in areas of low contrast. For the purposes of real-time robotic vision, it is desirable to find a method of calculating optical flow that is less sensitive to noise in the imaging process, gives a dense output independent of the structure in the image, and is computationally efficient.

2 Correlation-based Optical Flow

In correlation-based flow such as in [3, 12, 15] the motion for the pixel at [x,y] in one frame to a successive frame is defined to be the determined motion of the patch P_ν of $\nu * \nu$ pixels centered at [x,y], out of $(2\eta+1)*(2\eta+1)$ possible displacements. We determine the correct motion of the patch of pixels by simulating the motion of the patch for each possible displacement of [x,y] and considering a match strength for each displacement. If ϕ represents a matching function which returns a value proportional to the match of two given features (such as the absolute difference between the two pixels' intensity values), then the match strength M(x,y;u,w) for a point [x,y] and displacement (u,w) is calculated by taking the sum of the match values between each pixel in the displaced patch P_ν in the first image and the corresponding pixel in the actual patch in the second image:

$$\forall u, w : M(x, y; u, w) = \sum \phi(E_1(i, j) - E_2(i + u, j + w)), (i, j) \epsilon P_\nu \quad (1)$$

The actual motion of the pixel is taken to be that of the particular displacement, out of $(2\eta+1)*(2\eta+1)$

*This research was conducted while the author held a National Research Council Research Associateship at NIST.

†Author's address: Intelligent Systems Division, Bldg 220 Rm B-124, Gaithersburg, MD 20899 USA. Email: tac@cme.nist.gov. URL: http://isd.cme.nist.gov/staff/camus/

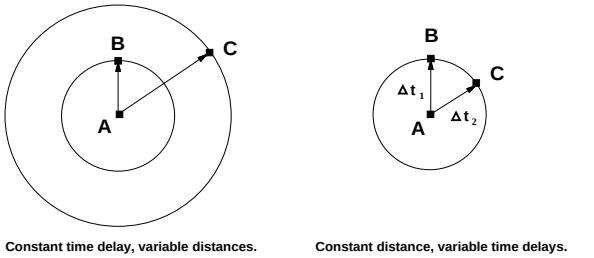

Figure 1: As the maximum pixel shift increases linearly, the search area increases quadratically. However with a constant shift distance and variable discrete time delays, search over time is linear.

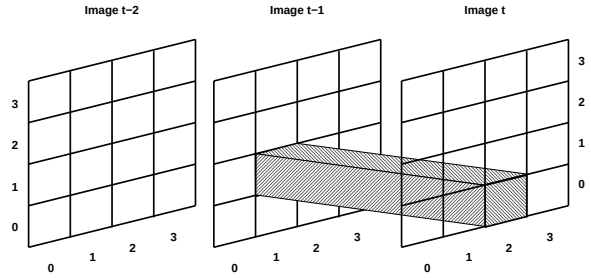

Figure 2: Visualization of motion of pixel (1,1) in image $T-1$ to pixel (2,0) in image T, an optical flow of (1,-1) pixels per frame.

possible displacements, with the maximum neighborhood match strength (equivalently minimum patch difference); thus this is called a "winner-take-all" algorithm.

This algorithm has many desirable properties. Due to the two-dimensional scope of the matching window, this algorithm generally does not suffer from the aperture problem except in extreme cases [4], and tends to be very resistant to random noise. Since the patch of a given pixel largely overlaps with that of an adjacent pixel, match strengths for all displacements for adjacent pixels tend to be similar (except at motion boundaries), and so the resultant optical flow field tends to be relatively smooth, without requiring any additional smoothing steps. Conversely, any noise in a gradient-based method usually results in direct errors in the basic optical flow measurements due to the sensitivity of numerical differentiation. In fact the correlation-based algorithm's "winner-take-all" nature does not even require that the calculated match strengths have any relation whatsoever to what their values should theoretically be, it is only necessary that the best match value correspond to the correct motion. For example, a change in illumination between frames would adversely affect the individual match strengths, but need not change the best matching pixel shift. Conversely, a gradient-based algorithm's image intensity constraint equation would not apply since the total image intensity does not remain constant. Finally, since one optical flow vector is produced for each pixel of input (excepting a small $\eta + \lfloor \nu/2 \rfloor$ border where flow may not be calculated), optical flow measurement density is 100 percent.

3 Time-Space Tradeoff

One limitation with the traditional correlation-based algorithm described in Section 2 is that its time complexity grows quadratically with the maximum possible displacement allowed for the pixel [4, 8]; see the left of Figure 1. Intuitively, as the speed of the object being tracked doubles, the time taken to search for its motion quadruples, because the area over which we have to search is equal to a circle centered at the pixel with a radius equal to the maximum speed we wish to detect.

However, note the simple relationship between velocity, distance and time:

$$velocity = \frac{\Delta distance}{\Delta time}.$$

Normally, in order to search for variable velocities, we keep the inter-frame delay δt constant and search over variable distances (pixel shifts):

$$\Delta v = \frac{\Delta d}{\delta t}, \quad d \leq \eta.$$

However, we can easily see from Figure 1 that doing so results in an algorithm that is quadratic in the range of velocities present. Alternatively, we can keep the shift distance δd constant and search over variable time delays:

$$\Delta v = \frac{\delta d}{\Delta t}. \tag{2}$$

In this case, we generally prefer to keep δd as small as possible in order to avoid the quadratic increase in search area. Thus, in all examples δd is fixed to be a single pixel. (Note however, there is nothing preventing an algorithm based on both variable Δd and Δt's). Since the frame rate is generally constant, we implement "variable time delays" by integral multiples of a single frame delay. Thus, we search for a fixed pixel shift distance $\delta d = 1$ pixel over variable integral frame delays of $\Delta t \in \{1, 2, 3, ...S\}$. S is the maximum time delay allowed and results in the slowest motion detectable, $1/S$ pixels per frame. For example, a $1/k$ pixels per frame motion is checked by searching for a 1-pixel motion between the current frame t and frame $t-k$. Thus our pixel-shift search space is fixed in the 2-D space of the current image, but has been extended linearly in time. As before, the chosen motion for a given pixel is that motion which yields the best match value of all possible shifts.

For example consider Figure 2 and Figure 3. Here we are trying to calculate the optical flow for pixel (1,1) at the current frame, image T. In Figure 2 the optimal optical flow for pixel (1,1) from image $T-1$ to image T is calculated to be a pixel shift of (1,-1). This is only a temporally local measurement however;

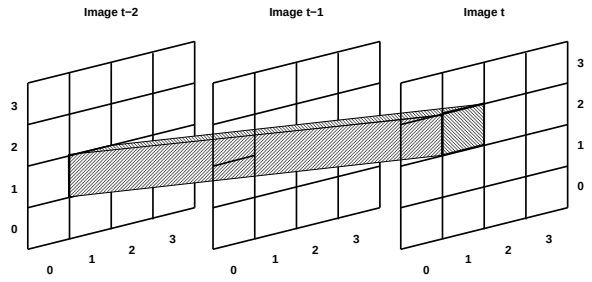

Figure 3: Visualization of motion of pixel (1,1) in image $T-2$ to pixel (1,2) in image T, an optical flow of (0,1/2) pixels per frame.

it may not be the final chosen motion. In Figure 3 the same search is performed, except using image $T-2$ as the first image. In this case the calculated motion happens to be a pixel shift of (0,1) pixels over 2 frames, equivalently (0,1/2) pixels per frame motion. If the maximum time delay $S = 2$, then the procedure would stop here, otherwise we would continue processing frames until finally the optical flow from image $T-S$ to image T was calculated as well. The best of all these motions is taken to be the actual motion. This approach does assume that a motion of $1/S$ pixels per frame is continuous for at least S frames before it can be registered; failure of this assumption can lead to temporal aliasing and the *temporal aperture problem* [5, 6]. However, the temporal anti-aliasing heuristic described in [5, 6] imposed a latency of a single frame before producing results. Although this is much less than techniques that smooth with a temporal Gaussian, any latency is undesirable when dealing with real-time robotics, thus this technique is not employed in the examples used in this paper.

It should be stressed once again that although there are *performance* advantages in keeping the search radius small (i.e., with a minimum of one pixel), there is nothing preventing an algorithm based on both variable Δd and Δt's, in which case this algorithm can be viewed as being similar to that of [3], but extended in time. Each pixel shift is treated no differently that any other. A search radius of two pixels consists of 25 pixel shifts instead of 9, and a search radius of three pixels consists of 49 pixel shifts; adding one to a search radius of $\eta - 1$ increments the search cost by $8*\eta$. Thus it is not *easier* to compute flow that occurs within a limited radius, it is merely *faster*. This situation is different from a simple gradient technique, which is in general limited to nearby pixels, unless multi-resolution techniques are employed. Multi-resolution techniques may also be used in correlation-based optical flow such as in [1], and could be used in conjunction with this algorithm as well. Pyramid techniques can be used to good effect when large areas of the image move coherently, but are less effective when there are multiple velocities present [2], which is more likely to occur at coarser resolutions [5]; in any event hierarchal techniques are not inconsistent with the methods described in this paper. However, a fast optical flow

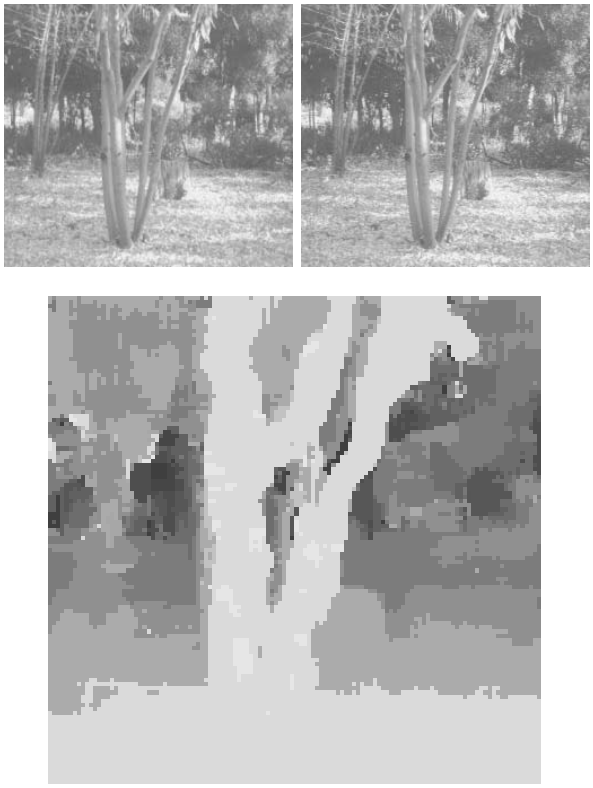

Figure 4: Two images from the SRI tree sequence along with a grey-level image indicating the magnitude of the resulting optical flow.

algorithm can support faster frame rates where any given motion per frame would be less than if a slower frame rate were used, in which case it would not be necessary to track motion over several pixels in the first place.

Figure 4 shows two images from the SRI tree sequence, subsampled to 128x128 pixels in size, along with a grey-level plot of the magnitude of the resulting flow (in this example all flow results from the translation of the camera orthogonal to the line of sight). For all sequences in this paper, $S = 10$ and $\nu = 7$.

This time-space tradeoff reduces a quadratic search in space into a linear one in time, resulting in a very fast algorithm: optical flow can be computed on 64x64 images, calculating 10 speeds per frame, at over 5 frames a second on a 50 MHz Sun Sparcstation 20[1]. This algorithm has been successfully used on many real and synthetic image sequences for a variety of real-time robotic vision tasks [8, 11, 7]. [11] reports being able to use this optical flow algorithm to instantiate some fly-like control laws [20] in a small mobile

[1] Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST nor does it imply that the materials or equipment identified are necessarily the best for the purpose.

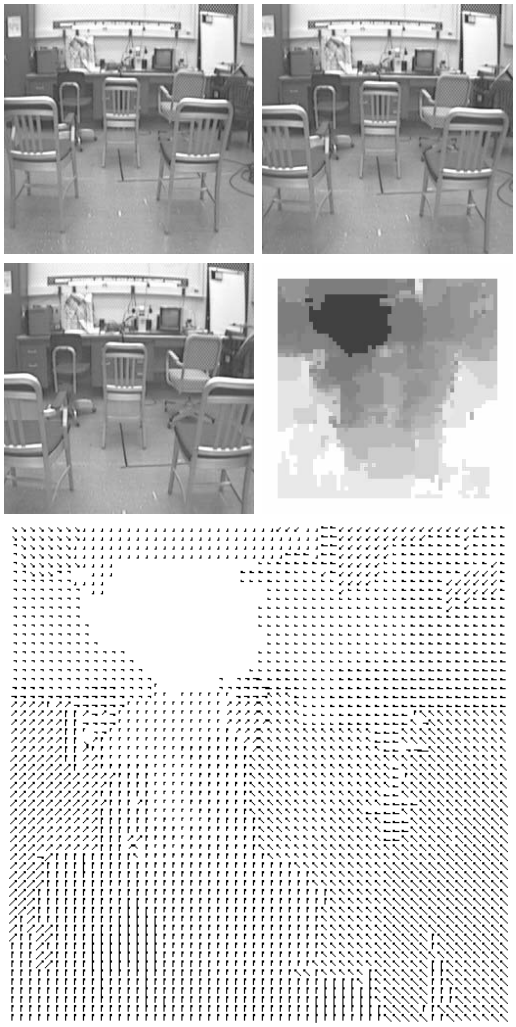

Figure 5: Three frames showing an approach to metal chairs, with magnitude and optical flow "needle" plots for the third frame. The two peripheral chairs show clearly in the magnitude plot.

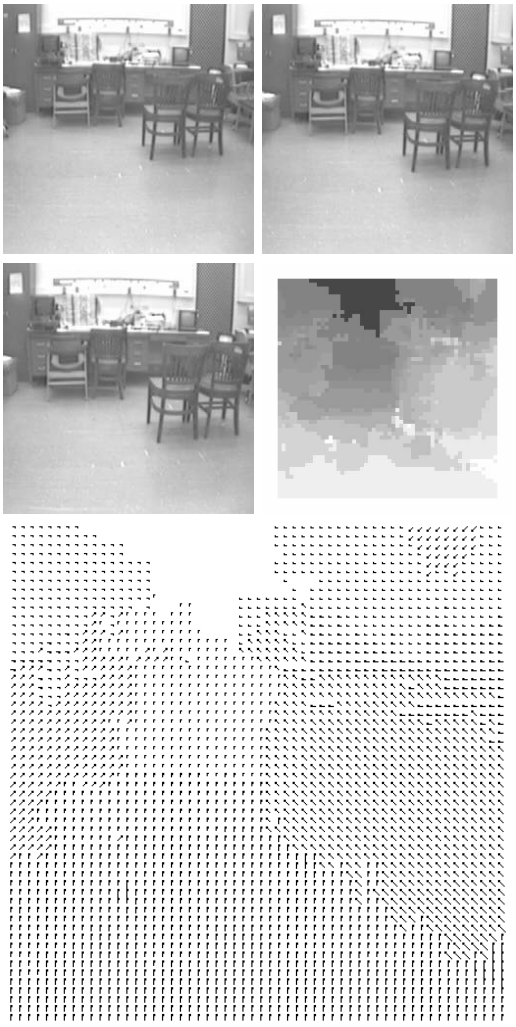

Figure 6: A more difficult sequence showing 3 frames of an approach to wooden chairs, with magnitude and flow plots. The chairs on the right show clearly despite their slow image motion. The flow on the floor also shows clearly despite a lack of texture and sharp edges.

robot. With a frame rate of 4-5 frames per second and the robot moving at 4-5 cm per second, the robot is able to successfully maneuver through an unmodified office environment.

One disadvantage of the patch-matching approach is that the basic motion measurements are integer multiples of pixel-shifts. Although the algorithm discussed in this paper does calculate sub-pixel motions, it still computes velocities that are basically a ratio of integers, not a truly real-valued measurement. Calculating real-valued optical flow measurements by using interpolation is currently being researched. In any event, remarkable accuracy has already been shown in applications such as calculating time-to-collision to sub-frame precision despite these deficiences [5, 7].

A primary application for this algorithm is real-time obstacle avoidance. Figure 5 shows 3 frames (spaced 10 frames apart) of an approach to some metal chairs, with the magnitude and optical flow "needle" plots for the third frame shown. The two peripheral chairs are clearly seen in the magnitude plot. Figure 6 is a more difficult sequence showing 3 frames of an approach to wooden chairs, whose edges are not as sharply defined against the background. The pair of chairs on the right are clearly differentiated in the magnitude plot despite the relatively slow motion. In addition note that the optical flow on the floor is also very well defined despite the lack of texture and sharp edges. These images were taken with a frame rate of 5 frames per second, with the robot moving at 12 centimeters per second. The central 256x256 pixels (shown) were taken from a 512x512 image using a 115 degree field-of-view camera, and were subsampled to

64x64 pixels in size using a basic block subsampling technique. Subsampling by averaging each NxN block of pixels into a single 8-bit value can be done extremely quickly without special hardware (although I/O may be a problem on some systems); no other temporal or spatial smoothing was performed. The flow plots are generated and displayed in real-time simultaneously with the image flow computation; a special routine converts real-valued optical flow values into one of 161 possible 8x8 bitmaps. The resulting flow plots are 56x56 pixels in size, due to the $\eta + \lfloor \nu/2 \rfloor$ border. The real-time display typically slows down the frame rate by about 20%.

Example animations are available via the WWW at the URL listed in the author's address.

4 Real-Time Implementation

A major advantage of the optical flow algorithm described here is that it can be implemented using only simple integer arithmetic such as adds, subtracts and integer compares, which can execute very quickly compared to floating point calculations. About two-thirds of the computational time of this algorithm is consumed during the patch-matching (equation 1) stage during which the best matching pixel shift is found. A key part of this stage is known as a *box filter*. This single operation is so important to real-time operation that it will be described in some detail here.

The first step of the patch-shift-match operation is the shifting stage [3]. In a massively parallel machine such as the Connection Machine, it might be desirable to implement this as an explicit image shift so that corresponding pixels are located in the same processing node. In the case of a serial processor however, this can be implemented at negligible cost by using normal addressing arithmetic on certain microprocessors. Once the base addresses of corresponding shifted rows are calculated, a single index register can be updated with each new pixel in many computer architectures.

After the shift stage (whether explicit or implicit) each pair of corresponding pixels is compared. This comparison could be either the sum of absolute differences (SAD) of each corresponding pair of pixels' intensity values, or the sum of their squared differences (SSD). Both sum-of-absolute differences and sum-of-squared differences could be implemented with no difference in computational time by using lookup tables for the absolute-value and squared-value functions. If a lookup table is not convenient (as it might not be in certain custom hardware), absolute value can be easily calculated on an N-bit 2's complement machine by the expression:

```
abs(x) = (x^(x>>(N-1)))-(x>>(N-1))
```

where '^' is the standard C operator for bitwise XOR and '>>' represents an arithmetic right shift.

The next step is referred to as the "Excitation" stage in [3], or the local neighborhood summation stage, as demonstrated in Figure 7. Here each element of the absolute difference (or squared difference) array is replaced with the sum of all elements in a local neighborhood $\nu * \nu$ in size. In Figure 7, the neighborhood is 3x3 in size. Note that the final box-sums

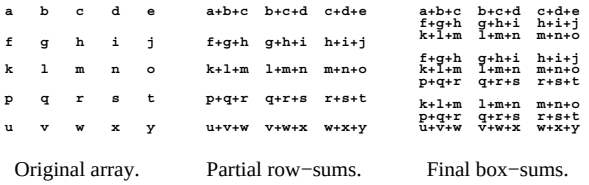

Figure 7: Demonstration of a 3x3 box filter.

array is not defined on the very edges of the image, since that would require pixels not available in the image itself. A naive implementation would perform an explicit $\nu * \nu$ summation for each pixel location; such an algorithm would be quadratic in ν. A much better way is to perform a box filter, whose running time is independent of the neighborhood size ν. A box filter is performed in two passes, one along the rows of the absolute difference array and another along the columns[2]. The first pass creates the partial row-wise summations, which are then added along the columns to yield the full neighborhood summations. Each pass is performed by taking the value for the previous location, adding one new absolute difference (corresponding to the new scope of the $\nu * \nu$ neighborhood), and subtracting an old absolute difference (corresponding to the neighborhood of the previous location). This takes advantage of the fact that the neighborhoods of adjacent locations overlap completely except for the most extreme elements. Therefore, a complete box summation consists of only a constant four simple computations, two along the element's row and two along the element's column (except for the very first element of each row or column which initializes the running summations), and this is independent of the size of ν. The final "winner-take-all" stage is then done based on the match values.

A remarkable fact is that the neighborhood summations can be speeded up significantly by calculating multiple sums in a single register. Normally, a single neighborhood summation would be accumulated into a single register. If we are summing absolute differences of unsigned 8-bit values (the usual representation for images), with a maximum neighborhood size $\nu = 15$, we are guaranteed that the entire sum of the $\nu * \nu$ absolute differences in the neighborhood patch will fit into 16 bits. Furthermore, we are also guaranteed than this sum will never be negative during any point of the summation. Thus, we can load two such values into a register at once, and add or subtract entire registers in a single cycle, effectively doubling the patch summation bandwidth for vertical passes. On 64-bit architectures, 4 such summations could proceed each cycle. It appears that this optimization may only be performed on column-wise summations, where multiple adjacent values may be loaded simultaneously into a single register. For example, a block load of 2 16-bit summations, when stored in row-major order, can be effected by a single load of one 32-bit integer.

[2] The "two-pass" implementation of the constant-time box filter was first pointed out to this author by Jim Little.

If it were possible to quickly store the first-pass partial summations in column-major order, then this optimization could be performed a second time, however attempting to do so appears to introduce too much overhead to be worthwhile.

A fully optimized implementation can calculate optical flow on 64x64 images, calculating 10 speeds per frame, at over 5 frames a second on a 50 MHz Sun Sparcstation 20. Naturally, some care must be taken to insure that these optimizations are properly implemented. A proper understanding of pointer arithmetic is essential for optimal performance.

Using the technique of [4] (the same as described in Section 2) on a Datacube MaxVideo 200, [15] calculates optical flow within a limited radius of +/- 3 pixels horizontally and +/- 2 pixels vertically using a 7x7 correlation window at 10 Hz on 128x120 images. Since this implementation uses the same basic correlation measure as that described in this paper, it is instructive to compare the performance of the two. We will use single-pixel-shift units as a basic measure, i.e., the comparison of a single given pixel with a single hypothesized pixel shift. Recall that in the traditional algorithm [3, 4] only two frames are used, and a pixel is constrained to lie within a certain neighborhood, in their case 7x5. Motion is calculated everywhere except for a border region where motion calculation would require pixels not available in the image itself. This border region is the sum of the maximum pixel displacement in any direction plus half the correlation window; thus the "active" image area is (128-2(3+3))*(120-2(2+3)) = 116*110. Given a frame rate of 10 Hz, this is 7*5*116*110*10 = 4,466,000 single-pixel-shift units per second. (As has been noted, the correlation window size, which is 7x7 for both algorithms, does not affect computational complexity.) In the linear-time algorithm, the local neighborhood is +/- 1 pixel for a search neighborhood of 3x3. In addition, there are multiple time delays, which adds another factor. The active image area is (64-2(1+3))*(64-2(1+3)) = 56*56. When calculating 10 speeds the frame rate is 5 frames per second on a 50 MHz Sun Sparcstation 20. This results in 3*3*10*56*56*5 = 1,411,200 single-pixel-shift units per second. Thus the workstation implementation is only 3.2 times slower than the implementation on the dedicated image processor. It is expected that by the time of this printing, further increases in CPU performance will have closed this gap [5].

Acknowledgments

The translating tree sequence was taken at SRI. The remaining images were taken by the author. This research was conducted while the author held a National Research Council Research Associateship.

References

[1] P. Anandan, A Computational Framework and an Algorithm for the Measurement of Visual Motion, *International Journal of Computer Vision*, 2:283-310, 1989

[2] J. Barron, D. Fleet, S.S. Beauchemin, Performance of Optical Flow Techniques, *International Journal of Computer Vision*, 12(1):43-77, 1994

[3] H. Bülthoff, J. Little, T. Poggio, A Parallel Algorithm for Real-time Computation of Optical Flow, *Nature* 337(6207):549-553, 9 Feb 1989

[4] H. Bülthoff, J. Little, T. Poggio, A Parallel Motion Algorithm Consistent with Psychophysics and Physiology, *IEEE 1989 Workshop on Visual Motion*, 165-172, March 1989

[5] T. Camus, *Real-Time Optical Flow*, PhD Thesis, Brown University Technical Report CS-94-36, 1994

[6] T. Camus, Real-Time Optical Flow, SME Technical Paper MS94-176, MVA/SME Applied Machine Vision June 1994, Minneapolis Minnesota

[7] T. Camus, *Calculating Time-to-Contact Using Real-Time Quantized Optical Flow*, National Institute of Standards and Technology NISTIR 5609, March 1995

[8] T. Camus, H. Bülthoff, Space-Time Tradeoffs for Adaptive Real-Time Tracking, Mobile Robots VI, William J. Wolfe, Wendall H. Chun ed., *Proc. SPIE 1613*, p.268-276, Nov. 1991

[9] D. Coombs, M. Herman, T. Hong, M. Nashman, Real-Time Obstacle Avoidance Using Central Flow Divergence and Peripheral Flow, Fifth International Conference on Computer Vision, Cambridge MA, June 1995.

[10] A. Del Bimbo, P. Nesi, Optical Flow Estimation on the Connection Machine 2, p.267-274, Workshop on Computer Architectures for Machine Perception, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[11] A. Duchon, W. Warren, Robot Navigation from a Gibsonian Viewpoint. 1994 *IEEE International Conference on Systems, Man and Cybernetics* San Antonio, Texas. October 2-5, 1994. IEEE, Piscataway, NJ, pp 2272-2277.

[12] R. Dutta. C. Weems, Parallel Dense Depth from Motion on the Image Understanding Architecture, *Proceedings of the IEEE CVPR*, p.154-159, 1993

[13] L. Huang, Y. Aloimonos, Relative Depth from Motion Using Normal Flow: An Active and Purposeful Solution, *IEEE Workshop on Visual Motion*, p.196-204, 1991

[14] B. Horn, P. Schunck, Determining Optical Flow, *Artificial Intelligence* 17:185-203, August 1981

[15] J. Little, J. Kahn, A Smart Buffer for Tracking Using Motion Data, p.257-266, *Workshop on Computer Architectures for Machine Perception*, New Orleans Louisiana, IEEE Computer Society Press, Los Alamitos CA, 1993

[16] J. Little, A. Verri, Analysis of Differential and Matching Methods for Optical Flow, *IEEE 1989 Workshop on Visual Motion*, 173-180, March 1989

[17] R. Nelson, J. Aloimonos, Obstacle Avoidance Using Flow Field Divergence, *IEEE PAMI-11* no. 10, p.1102-1106, October 1987

[18] D. Patterson, J. Hennessy, *Computer Organization and Design: the Hardware/Software Interface*, Morgan Kaufmann, San Mateo CA, 1994

[19] D. Touretzky, H. Wan, A. Redish, Neural Representation of Space in Rats and Robots, in J.M. Zurada and R.J. Marks, eds., Computational Intelligence: Imitating Life. *Proceedings of the symposium at the IEEE World Congress on Computational Intelligence*, IEEE Press, 1994

[20] W. Warren Jr., Action Modes and Laws of Control for the Visual Guidance of Action, in *Complex Movement Behavior: The motor-action controversy*, p.339-380, O. Meijer and K. Roth eds., Elsevier Science, B.V. (North-Holland), 1988

[21] J. Weber, J. Malik, Robust Computation of Optical Flow in a Multi-Scale Differential Framework, *Fourth International Conference on Computer Vision*, p.12-20, 1993