A fast algorithm for computing optic flow and its implementation on a Transputer array

Han Wang[†] Michael Brady[‡] Ian Page[†]

[†]Programming Research Group
Oxford University Computing Laboratory, 11 Keble Rd., Oxford, OX1 3QD
[‡]Robotics Research Group
Engineering Science Department, Parks Rd., Oxford, OX1 3PJ

This paper describes a parallel algorithm, MCMAP, to compute the optic flow along intensity change curves. The flow is computed locally at corners, determined as edge points of high curvature, by cross correlation over time. The (full) flow is then computed along edge contours by means of a combined wave/diffusion process. MCMAP has been implemented on a Transputer array using a mixed paradigm of a pipeline and a processor farm.

1 Introduction

The advance of parallel processing technology makes real-time image processing increasingly feasible. However, many existing vision algorithms are designed for sequential machines. Even those that are designed to be parallel are typically only implemented on sequential machines. Many algorithms are either not suitable for implementation on real parallel machines, or are not efficient. This paper describes a parallel algorithm for computing optic flow and its implementation on a Transputer array. Experimental results are presented and the performance on set of Transputers interfaced to a Sun workstation is analysed.

Assuming that the motion field is smooth, so that the intensity function $I(x, y, t)$ can be expanded up to first order to give the familiar motion constraint equation:

$$\nabla I \cdot \mu = -\frac{\partial I}{\partial t} \tag{1}$$

where $\mu = (u, v)$ is the optic flow. An early iterative algorithm was derived by Horn and Schunck [2]. The scheme works well on smooth portions of flow, eg. those generated by textured smooth surfaces. However, it fails at edges where moving objects occlude the background.

Hildreth developed an edge-based approach to computing the optic flow. First, the normal component of the flow is estimated everywhere along a zero-crossing contour using the motion constraint equation. Then the full flow is estimated by regularisation around the (closed) contours. The corresponding Euler-Lagrange equation was solved numerically using conjugate gradient descent, an intrinsically sequential algorithm. The results presented by Hildreth are promising (see also [9] for a close variant called oriented smoothness), though the conjugate gradient descent algorithm is extremely slow to converge as motion estimates are propagated around closed zero-crossing contours (see [11] for discussion). Hildreth's algorithm also gives unstable results near around junctions, since zero-crossing contours are only simply connected and her algorithm made no attempt to locate, or prevent propagation of flow estimates through junctions.

To improve the time performance, Gong and Brady [7] have developed an algorithm for recovering flow along zero crossing contours. This algorithm decomposes flow into tangential and normal components where the normal component is computed using the motion constraint equation. The tangential flow is obtained only at certain locations along the contour using a certain second-order expansion of the intensity function called the Curve Motion Constraint Equation (CMCE). More precisely tangential flow estimates are obtained at locations where the image Hessian is well-conditioned. Typically, these are connected sets of contour points near corners. Local flows estimates are then propagated from seed locations along contours using the combined wave/diffusion process invented by Scott, Turner and Zisserman[12]. Note that the flow is prevented from propagating through junctions. The algorithm works on both closed and open contours. Gong and Brady [7] present a number of results with a serial implementation of the algorithm.

The curve motion constraint equation involves computing at each edge point the image Hessian which is:

$$\mathrm{H} = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix}$$

Since it involves second differentials, the Hessian is sen-

BMVC 1990 doi:10.5244/C.4.32

sitive to noise [9]. To overcome this problem, the tangential flow estimates in the contiguous set of contour locations were median filtered. Although [6] sketches a design for a parallel implementation, it was only implemented on a sequential processor.

Our goal was to implement a variant to the Gong-Brady algorithm on a set of Transputers. Results using the implemented algorithm are shown in the final section, together with timings. We call our algorithm MCMAP (Motion by Corner Matching and Propagation), since corner locations on contours are found after fitting local cubic B-Splines, and the corner motion is computed by correlation over time. MCMAP consists of the following steps:

(1) Edge Segmentation
Edges are computed using a Canny edge operator; Charkravarty's edge segmentation algorithm is used to segment and group edges obtained by the edge operator [3].

(2) Corner Detection
Corners are detected after fitting cubic B-Splines locally.

(3) Corner Matching
Local flow at corners is computed by matching edge contours over time by cross correlation algorithm.

(4) Local Flow Propagation
Local flow estimates are then propagated from corners using the wave/diffusion algorithm [12, 6].

2 The Algorithm — MCMAP

2.1 Edge Segmentation

We have experimented with two edge finders so far: the Canny edge operator and the LoG (Laplacian of Gaussian). Our experiments show that side effect of the Canny edge operator is that it produces fewer junctions than LoG does, which is desirable for the subsequent propagation process. We plan to experiment with the morphological edge detector developed by Noble.

The output of the Canny edge operator is a sparse matrix with edge pixels (edgels) set to 1 and non-edgels set to 0. Edgels need to be connected into contours that makes the connectivity between edgels explicit. This process is often called edge segmentation. An edge thinning algorithm is required following edge finding in order to turn 4-connected edges into 8-connected edges. This is because edge segmenting algorithms typically assume single pixel edges.

Many algorithms have been proposed for segmenting edge contours. Unfortunately, they are either recursive [2, pages 131–146], or need more than one pass [10], which are not suitable for parallel processing on distributed memory machines. Charkravarty [3] has designed an algorithm which segments edges in a single pass in raster order and junctions are processed effectively. We have implemented Chakravarty's algorithm.

Edgels are encoded as *lists* of Freeman chain links[2]. Lists are polarised as head and tail, as well as chain links. The data structure of a list contains an array of chain links and pointers. Lists are referred to each other by pointers. The algorithm starts from the top left corner of the edge map, and scans it in raster order. As it decodes a chain link, the developing contour lists are searched and a list corresponding to one of them is added. The rule for padding a link is that the head must join only with the tail. If the head of a link is found to be connected with the head of a list, then the link must be reversed, that is its head becomes tail and tail becomes head. The next step is to update lists to find if the newly changed list has a connection with other lists. If so, these two lists are joined together. The head-tail rule applies in the same way.

The most important feature of this algorithm is its ability to deal with junctions. Charkravarty defines some 20 patterns for decoding junctions. Whenever a junction is encountered the two ends of the connected lists are closed by setting a close mark. A closed end of a list can no longer be appended. Instead, a new list is created and this new list has its pointer point to the closed list.

After the edge segmentation process, a grouping procedure organises these lists into a data structure that defines the set of contours. Each contour consists of simply connected edgels. A filtering process discards lists with less than 10 edgels and without junctions.

2.2 Corner Detection

Corners are defined here as contour points where the curvature is either discontinuous or reaches a significant maximum. Again, many algorithms have been proposed to locate corners, at various scales. Asada and Brady [1] used Gaussian filters to smooth the curve and get both corners and joins. The emphasis of that work was on scale space interpretation of curvature changes. It worked well though slowly. Gong [6] surveyed several algorithms and suggests that least squares fitting gives acceptable results for computing optic flow. However, our goal is real-time execution. Hence, a less costly B-Spline fitting algorithm [8] is adopted and its performance is found to be satisfactory.

In the B-Spline approach, a curve S is represented parametrically by

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases}$$

so that the curvature κ is given by,

$$\kappa = \frac{\frac{df}{dt} \cdot \frac{d^2 g}{dt^2} - \frac{dg}{dt} \cdot \frac{d^2 f}{dt^2}}{((\frac{df}{dt})^2 + (\frac{dg}{dt})^2)^{3/2}}$$

The cubic B-spline approximation involves fitting parametric cubic polynomials locally:

$$\begin{cases} x &= f(t) &= a_1 t^3 + b_1 t^2 + c_1 t + d_1 \\ y &= f(t) &= a_2 t^3 + b_2 t^2 + c_2 t + d_2 \end{cases}$$

to each set of four adjacent points of (x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1}), and (x_{i+2}, y_{i+2}). Having found the approximating polynomials, the curvature at point i $(t = 0)$ is easily seen to be:

$$\kappa = \frac{2(c_1 b_2 - c_2 b_1)}{(c_1^2 + c_2^2)^{3/2}} \tag{2}$$

The parameters are given by the B-spline formulation. In particular,

$$\begin{cases} d_1 = x_{i-1}/6 + x_i/3 + x_{i+1}/6 \\ d_2 = y_{i-1}/6 + y_i/3 + y_{i+1}/6 \end{cases}$$

is the displacement of the approximating curves from the given data points. Medioni suggests repeating the B-spline fitting process on the displaced points and re-estimating the curvature. The bottom line is that the curvature is estimated using a smoothing procedure involving five points. This is fast to compute. Corners are determined to be contour locations where

1. the curvature is over a given threshold, and
2. the curvature is a local positive maximum or negative minimum

In Medioni's original algorithm, the displacement measure $(x_i - d_1, y_i - d_2)$ is used to determine high curvature points in addition to the corner points. This measure is not applicable in our case since we are interested only in the corners.

2.3 Corner Matching

Corner matching is used to determine the motion of corners. In our approach, matching is implemented using cross correlation. Henceforth, let I_t and $I_{t+\Delta t}$ denote image frames at time t and $t + \Delta t$. Assume a corner point p on a rigid body undergoes motion

$$v_p = (v_x, v_y)|_p = (\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})|_p$$

Normalising time sampling $\Delta t = 1$, we have

$$v_p = (\Delta x, \ \Delta y)|_p$$

where Δx, Δy are the offset at point p over time Δt, or in another words, the point p is to be found at $p' = p + v_p$. To find v_p, an image window α can be extracted from I_t at location p. Given a domain $\phi \subset I_{t+\Delta t}$, a window β of the same size as α is extracted from ϕ. Correlating α and β, the offset $(\Delta x, \ \Delta y)$ can be determined from the best match of α and β, that is,

$$\frac{\sum_{i=1,2...n} (\alpha_i - \bar{\alpha}) \cdot (\beta_i - \bar{\beta})}{(n-1) \cdot \sigma_\alpha \cdot \sigma_\beta}$$

is a maximum within the search area ϕ, where $\bar{\alpha}, \bar{\beta}$ are the mean, and $\sigma_\alpha, \sigma_\beta$ are the standard deviation.

The advantage of this corner matching strategy is (1) it gives *both* qualitative and quantitative measures of motion, and (2) the amount of computation is small since only corners are used for matching (typically, for a 128×128 image, there are 50~100 corner points). In our experiments, the window size used is 5×5 and the search area is 8×8. Larger search areas can be used in combination with pyramid processing to reduce the overall cost. Alternatively, and more interestingly, tracking algorithms (including the ubiquitous Kalman filter) can be used to provide prior motion estimates and validation gates for outlier rejection.

2.4 Propagation

The cross correlation technology applied at corners estimates the full flow at such points. The flow field along an edge can be reconstructed using a smoothness constraint. Applying the Motion Coherence Theory [13] in one dimension along the edge contour, Gong has shown that motion estimation along the contour can be propagated from corners by a diffusion process[6]. Unfortunately, a diffusion process is unacceptedly slow for reasonably long contours. A wave equation propagates data at a constant speed [12], but does not satisfy motion coherence theory. For these reasons a combined wave/diffusion process[12] is adopted by Gong to propagate local flows at reasonable speed, that is each iteration of the diffusion process is interleaved with a wave equation.

3 Parallel Implementation

MCMAP is implemented in parallel C on an array of Transputers. Parallel C adopts Hoare's communicating sequential process (CSP) model. There are two primitives for achieving parallelism—multi-tasks and multi-threaded tasks. A task is a collection of processes. Communication between tasks are strictly confined to channels. Threads both provide channels and the ability to share memory.

MCMAP was first run on a single Transputer. In a typical experiment, the times taken for each functional step was measured, Typically, for 128 × 128 images, timings measured are:

$$\begin{aligned}
&(1)\ \mathit{Edge\ operator + thinning} &&= 3sec\\
&(2)\ \mathit{Segment} &&= 0.5 \sim 0.7sec\\
&(3)\ \mathit{Corner\ Detection\ \&\ Matching} &&= 2.5 \sim 3sec\\
&(4)\ \mathit{Wave/diffusion} &&= 10 \sim 15sec
\end{aligned}$$

We are unconcerned about the time taken for edge detection and thinning, since we expect to transfer this computation imminently to a Datacube image processor, interfacing to the Transputers using a board developed by BAe Sowerby. Each procedure is dependent on the previous one, thus the processes can be pipelined. Note that the wave/diffusion process takes considerably longer than the other processes. However, it is naturally decomposable, since the propagation along different contours are completely independent of each other. To exploit this, we introduce a processor farm to share computations among processors. This suggests a mixed network topology of a pipeline and a processors farm [5]. Procedure (1), (2) and (3) are placed into three processors connected as a pipe. The wave/diffusion procedure is loaded into three other processors to compute edge contours independently. In this system, there is also a frame grabber to provide a sequence of images and a user interface and dealing with file I/O. Figure 1 is the connection diagram for MCMAP. Thereafter, the processor for computing edges will be referred to as edge processor, hence segment processor, corner processor and wave/diffusion processor. The frame grabber is not connected at present and is replaced by a normal Transputer called the server.

3.1 Pipeline

In addition to grabbing images, the server has two other jobs: (1) sending a sequence of images to both edge and corner processors, and (2) receiving computed contours. Two threads run simultaneously on the server to carry out these two tasks. In Figure 1, the output of segment processor is a group of edge contours of varying length. The corner processor needs data of both edge contours and intensity images to conduct the corner matching. There is a link between the corner processor and the server to short cut the path for sending images form server to the corner processor.

In each iteration of the sending thread in the server, an image is captured and sent down to both the edge processor and the corner processor. The receiving thread gets a number N of contours from the corner processor, and then gets N contours from the wave/diffusion processors.

3.2 A Processor Farm

In the wave/diffusion process, a single job corresponds to a contour. The complexity of a job can be approximated by $O(\frac{\ell}{c+1/log\ell})$, where ℓ is the length of the contour and c is the propagating speed of the wave. Often computations are skewed, for example, in our experiment (Figure 4) the time spent on computing the outline of the car (ℓ = 127) is greater than the sum of computing the rest of contours in one image. Hence, a dynamic scheduling strategy is adopted [4]. In this dynamic model, contours are sorted in descending order. An idle processor will ask for a contour; when the corner processor finished computing a contour, it will then send this contour to the requesting processor.

Figure 2 is the logical diagram for the processor farm. **w0, w1,** and **w2** are the wave/diffusion process; "+" and "−" are the multiplexer and the de-multiplexer. Edge contours come through the **in** channel and goes out from the **out** channel. Requests from idle processors are coming from the **request** channel.

Normally, a worker processor will have one job buffered when it is computing another one. In some cases, including the car experiment, the processor which gets the biggest job will dominate the entire computation time. Hence, it is not suitable to buffer another job for this processor.

4 Experiments and Discussion

In this section, we present two examples of using MCMAP on real image sequences. We have made several motion sequences each of thirty images. Video of the resulting tracking algorithm has been made. Figure 3(a) is a cup moving towards left on a static table background, 3(b) shows the edge segments after filtering and the local estimates of corner motion. Figure 3(c) is the propagated full flow along edge contours. Figure 4 shows two cars moving towards the top right corner of the image. It is obvious that one car is moving faster than the other (note the length of the flow vectors).

The following table gives timings (in seconds) of MCMAP on a Sun 3/75 workstation and T800 Transputers for the above examples. Images are $128 \times 128 \times 8$ bits.

Images	Sun3	1 T800	8 T800
cup	36.6	17.7	4.3
cars	28.3	12.9	2.7

Obviously, the current architecture scales easily as additional Transputers are made available. For example, edge detection can be shared by more than one processors using a closed ring. Inherently, MCMAP consists of two distinct parts, where the first part (edge finding, and edge segmenting) is data independent and the second part is not. We intend to use a Datacube to execute the first part and a Transputer array to implement the second part.

A comment contrasting MCMAP with Gong's curve motion constraint equation should be made. In Gong's algorithm, the normal flow $\mathbf{n}$ was computed using the motion constraint equation

$$\mathbf{n} \cdot \mu = \frac{\frac{dI}{dt}}{|\nabla I|}$$

where μ is the full flow. Gaussian smoothing was used before applying this equation. Let G denote the Gaussian function, and $\otimes$ denote convolution, then

$$I^{'} = G \otimes I$$

Since, in general,

$$\frac{\frac{dI}{dt}}{|\nabla I|} \neq \frac{\frac{dI^{'}}{dt}}{|\nabla I^{'}|}$$

$\mathbf{n}$ is only a *qualitative* measure. For the same reason, the measurement of the tangential flow cannot be accurate. However, this is *not* a problem in MCMAP where flows are not decomposed into tangential and normal components; instead local full flow estimates are propagated.

References

[1] H. Asada and J. M. Brady. The curvature primal sketch. *IEEE Trans. on PAMI*, PAMI-8(1):2–14, 1986.

[2] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, New Jersey, 1982.

[3] I. Charkravarty. A single-pass, chain generating algorithm for region boundaries. *Computer Graphics and Image Processing*, 15:182–193, 1981.

[4] P. M. Dew and H. Wang. Data parallelism and the processor farm model for image processing and synthesis on a transputer array. In *Proc. of SPIE Symposium*, volume 977, pages 212–220, Aug. 1988.

[5] P. M. Dew, H. Wang, and J. A. Webb. Apply: Machine independent image processing language and its implementation on a meiko computing surface. In *5th Alvey Vision Conference*, Reading, UK, Sept. 1989.

[6] S. Gong. *Parallel Computation of Visiual Motion.* PhD thesis, Dept. Engineering Science, Oxford University, Sept. 1989.

[7] S. Gong and J. M. Brady. Parallel computation of optic flow. 1990. to appear.

[8] G. Medioni and Y Yasumoto. Corner detection and curve representation using cubic b-splines. In *IEEE Int. Conf. on Robotics and Automation*, pages 764–769, San francisco, CA, 1986.

[9] H. Nagel. On the estimation of optical flow: Relations between different approaches and some new results. *Artificial Intelligence*, 33:299–324, 1987.

[10] T. Pavlidis. A minimum storage boundary tracing algorithm and its application to automatic inspection. In *IEEE Trans. Syst. Man Cybern*, volume SMC-8, pages 66–69, 1988.

[11] G. L. Scott. Fast recovery of the optic flow around a closed contour using stabilised regression onto fourier components. In *the Fifth Alvey Vision Conference*, Sept. 1989.

[12] G. L. Scott, S. Turner, and A. Zisserman. Using a mixed wave/diffusion process to elicit the symmetry set. In *Alvey Vision Conference*, Sept. 1988.

[13] A. Yuille. A motion coherence theory. In *IEEE Inter. Conf. on Computer Vision*, Tampa, Florida, Dec. 1988.

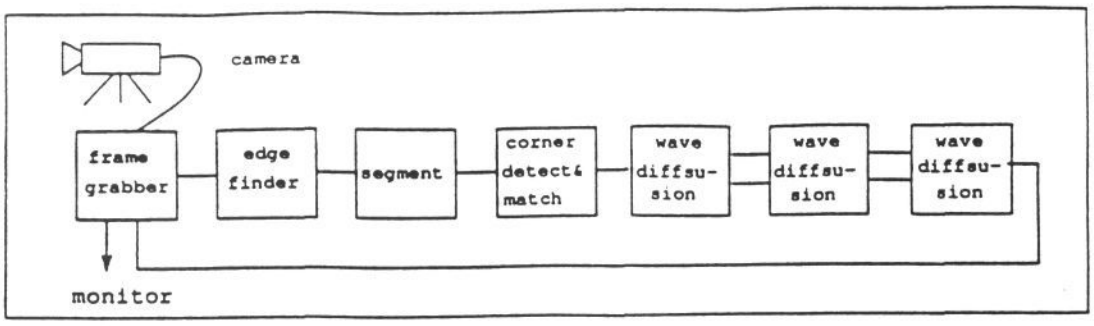

Figure 1. Hardware Configuration for MCMAP

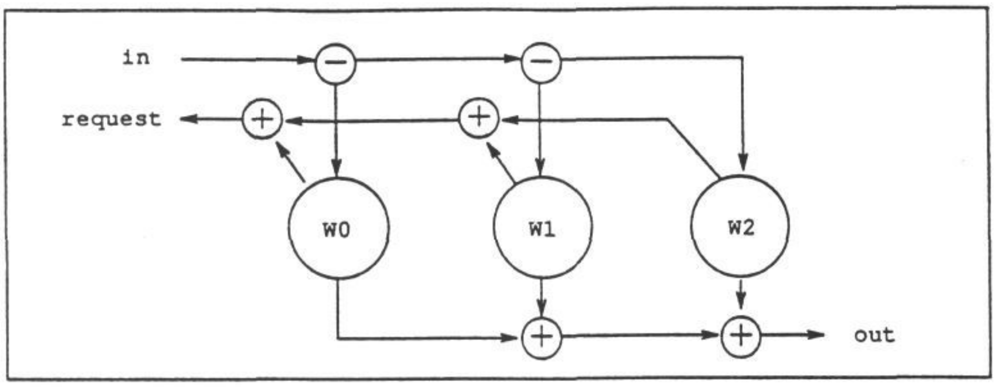

Figure 2. The Processor Farm

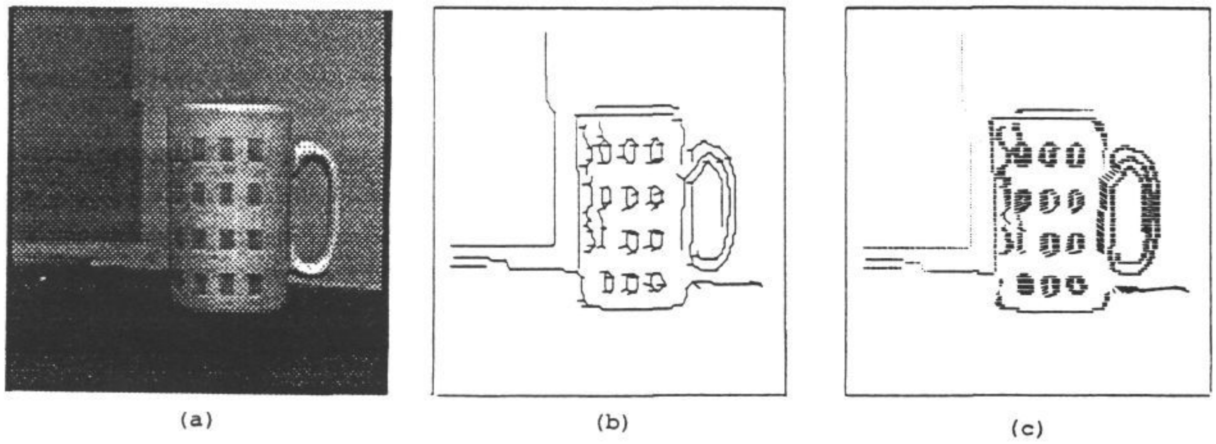

Figure 3. A cup moving left

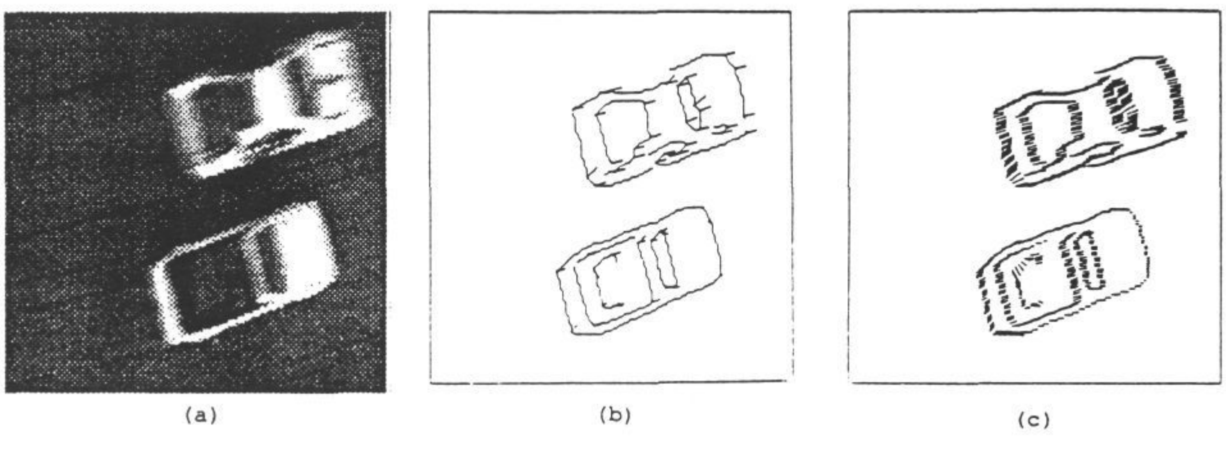

Figure 4. Two cars moving with different speeds