

Title of the paper:

DepthFinder, A Real-time Depth Detection System for Aided Driving

Authors:

Yi Lu Murphey¹, Jie Chen¹, Jacob Crossman¹, Jianxin Zhang¹, Paul Richardson² and Larry Csieh²

¹Department of Electrical and Computer Engineering
The University of Michigan-Dearborn
Dearborn, MI 48128-1491

²U. S. Army Tank-Automotive Command
ATTN: AMSTA-O
Warren, MI 48397-5000

Technical Category:

Special session: Military Applications and Current Research

Contact author

Yi Lu Murphey
Department of Electrical and Computer Engineering
The University of Michigan-Dearborn
Dearborn, MI 48128-1491
Voice: 313-593-4520
Fax: 313-593-9967
yilu@umich.edu

DepthFinder, A Real-time Depth Detection System for Aided Driving

Y. L. Murphey¹, J. Chen¹, J. Crossman¹, J. Zhang¹, P. Richardson² and L. Csieh

¹Department of Electrical and Computer Engineering
The University of Michigan-Dearborn
Dearborn, MI 48128-1491
Yilu@umich.edu

²U. S. Army Tank-Automotive Command
ATTN: AMSTA-O
Warren, MI 48397-5000

Abstract

Many military applications require the distance information from a moving vehicle to targets from video image sequences. For indirect driving, lack of perception of depth in view hinders steering and navigation. In this paper we present a real-time depth detection system and a thorough discussion in performance analysis is presented. DepthFinder can be used with a camera mounted either at the front or side of a moving vehicle. A real-time matching algorithm is introduced to improve the matching speed hundreds of times. A simple axial motion model is established by mounting a camera on the vehicle facing the moving direction. The experiment results and the performance analysis are presented in the paper.

Keywords depth detection, axial motion, monocular image sequences.

1 Introduction

The application of computer vision and image processing can derive significant advantage in the battlefield in global picture generation as well as aided driving. Real-time depth detection plays an important role in many outdoor applications, particularly in vision-aided driving. Much research in depth detection has been conducted using stereo vision techniques[5]. Stereo vision establishes correspondence between a pair of images acquired from two well-positioned cameras. In this paper, we present our research in depth finding from monocular image sequences.

Monocular vision is interesting to a number of military applications. For example, in order to provide a full view of entire surroundings to a tank crew, we need to have a suite of cameras mounted at the front, rear, and both sides of a moving tank and each camera covers a specific area of the scene. Due

to the limited bandwidth of communication channels within a vehicle and cost, depth finding using stereo camera is not practical. For indirect driving, lack of perception of depth in mono-scope view hinders steering and navigation. Furthermore depth-finding from a monocular image sequence can be used as a fault-tolerant solution when one camera in a stereo system is damaged.

This paper describes a real-time depth detection system developed for in-vehicle surveillance. A video camera can be mounted at the front of the vehicle or at the side of a vehicle. The major task of the depth finding system is that at any time if a user moves the mouse to any object in the scene and click, the distance from the object to the vehicle is displayed. Obviously high speed computation is a critical issue.

2 Real time depth finding

The computation of depth from a monocular image sequence obtained in the time domain is based on the geometric model shown in Figure 1. From geometric optics, we have,

$$\begin{cases} \frac{1}{R_1} + \frac{1}{I_1} = \frac{1}{f} \\ \frac{1}{R_2} + \frac{1}{I_2} = \frac{1}{f} \\ \frac{H}{D_1} = \frac{R_1}{I_1} \\ \frac{H}{D_2} = \frac{R_2}{I_2} \end{cases} \quad (1)$$

where f is the focal length of the camera lens. From these equations, we have,

$$R_2 = \frac{D_1(D_2 + H)}{H(D_2 - D_1)} L \quad (2)$$

where $L = R_1 - R_2$ is the distance that the vehicle has moved during the time period that the two image frames are captured and $D_2 - D_1$ is the disparity between the two images taken at time t_1 and t_2 . Due to

the compact size of the camera, we can assume that $R_2 \gg I_2$ (i.e. the distance between the object and the lens is much greater than the distance between the lens and the image plane inside the camera) and $H \gg D_2$ (i.e. the actual size of the object is much greater than its image on the image plane). Thus, Equation (2) becomes,

$$R_2 = \frac{D_1}{D_2 - D_1} L \quad (3)$$

According to equation in (3), the computation of the distance from an object to the vehicle involves two stages. First, search two sequential images to find matching objects. Finding matching objects gives us the disparity, $(\Delta x, \Delta y)$, or relative movement of the object from frame to frame. Second, we use the camera parameters and the disparity from steps 1 to calculate the depth for each object of interest. The first step is to find the correspondence of object match between the two image frames. There are a number of approaches being studied in the correspondence problem, matching edges, object contour, or corners. These approaches depend very much the outcome of the image feature extraction algorithms, which are also computationally demanding. In DepthFinder, we use intensity feature to match corresponding pixels in the two adjacent image frames. In order to have accurate and efficient matching, we derived a number of motion heuristics including maximum velocity change, small change in orientation, coherent motion and continuous motion [4]. Based on these heuristics, for a pair of images I_t and I_{t+1} , we define a matching window and a search window to compute the correspondence problem (see Figure 2). The match window, the smaller square in the figure, is used to compute the similarity between the two portions in I_t and I_{t+1} . The search window is used to limit the search for the possible location of a particular pixel in the image frame I_{t+1} .

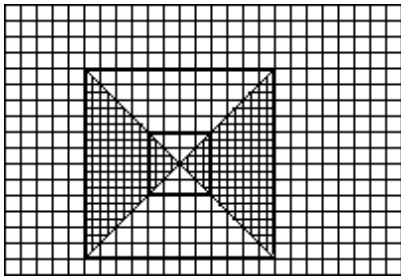


Figure 2. Illustration of search window and matching window. The smaller square is the matching window at the center pixel and the two shaded triangles form the search window.

The disparity between two images are computed as follows. For a pixel (x, y) in I_t , its corresponding location in I_{t+1} is found by using the following maximum likelihood function:

$$\Phi(u_i, v_j) = \sum_q \sum_p (I_t(x_i + p, y_j + q) - I_{t+1}(u_i + p, v_j + q))^2 \quad (4)$$

where p and q should be within the matching window, and, $\Phi(u_i', v_j') \geq \Phi(u_i, v_j)$ for all (u_i', v_j') within the search window. The use of the effective window reduces not only the computational time but also matching error. However, for an image of 540×480 and at the frame rate of 15 fps, there is a lot of computation in computing the disparity. A brute force implementation of the algorithm requires the worst computation on the order $O(L * H * D_x * D_y * p * q)$, where L and H are the image width and height, D_x and D_y are the maximum horizontal and vertical disparity values, and p and q are the matching region width and heights as defined above. For a 540×480 image with maximum disparity values $D_x = D_y = 4$ and an average-sized matching window of 32×32 , the number of comparisons (differences) that must be taken approaches 2 billion. For a Pentium II 400 MHz, the computation time is about 20 minutes. This is nowhere near real-time processing. We need to reduce the computational complexity by at least 1,000 times. To produce this speedup we use three techniques. First, we apply a dynamic programming algorithm to eliminate the matching window size (p, q) from the complexity. Second, we target the modern cache-dependent processor by localizing data access in the computation. Third, if the computation on a particular processor or image size is still not fast enough, we only calculate the depth of certain sub-region(s) of the image.

The dynamic programming algorithm is based on the fact that many of the computations (i.e. intensity difference calculations) are repetitive. We use Figure 3 to illustrate the algorithm. For the convenience of description, we superimpose a linear window (i.e. $q=1$) on images I_t and I_2 . We denote each difference and square operation with a pair of x coordinates, (x, x') where x is the coordinate in image I_t and x' is the corresponding coordinate in I_2 . As the matching window shifts across I_t , we re-calculate the square difference for the same pixel exactly $q-1$ times (ignoring the boundary conditions). When the window becomes rectangular (i.e. $p > 0$), we perform $(p-1)$ repetitive calculations in the vertical direction. Therefore, we can implement the matching algorithm as follows. We first vary the center pixel before varying the disparity (u_i, v_i) . This allows us to store the results for each square difference calculation in a table and look them up as needed. The data being stored are the sum and difference calculations for a single row of the image at a time (or possibly a $q \times L$), and the minimum $\Phi(u_i, v_i)$ and the associated disparity, (u_i, v_i) , in a table for every pixel in the image. This implementation reduces the computational complexity by a factor of the window size, $p \times q$, while

the extra storage is proportional to the size of the image.

The implementation of the dynamic programming fully utilizes the localization of main memory address. It is well known to professional code optimizers that localizing address requests to small regions of memory for long periods of time maximizes cache hits and can significantly increase performance. In our implementation we localize address requests in two ways. First, we make copies of the sub-region of the image needed in the calculation, and operate on those copies. Second, we exhaustively calculate all possible calculations (including saving intermediate results necessary for the dynamic method) on each row of the image before proceeding to the next row.

Combining the dynamic programming algorithm and cache-targeted optimizations we were able to reduce 20 minutes of computation time for a 240×700 image, $p = q = 32$, $D_x = 16$, $D_y = 0$, from 20 minutes to 2-3 seconds. For a small sub-region of the image (e.g. a 8×8 pixels square) our algorithm improved execution time from about 2-5 seconds to about 0.01 seconds. All tests were run on a Pentium II 400 MHz processor with 256MB of RAM.

The final calculation speed we require depends greatly on the application. If the video is being streamed in at 15-30 frames per second (fps) we need to calculate depth at least ever 2-5 frames and thus we cannot possibly calculate the depth of the entire image. Therefore we target our depth calculations on a small region surrounding the mouse (the size of the region can be set by the user), and regions surrounding specific targets chosen by the user (using some type of pointing device). Because the depth calculation includes motion estimation of image object, we have even been able to track objects from frame to frame while displaying their depth. If depth information is required at the rate of once every second or, as in the case of a slow moving robotic vehicle, we can calculate the depth across the entire image and display depths at user request for particular objects of interest.

3 Experimental Results and Conclusions

We have tested the DepthFinder system on many video image sequences and on a number of real-time driving experiments. During the real-time driving test, we have a camera mounted on top of a moving vehicle, a laptop computer that runs DepthFinder that controls the image capturing and displaying, computes and displays the depth of objects on the computer screen.

In our experiments, we selected outdoor scenes that contained the objects such as people, telephone pole, light poles, and trees that are particularly interesting to military applications. Figure 4 shows two image frames captured from one image sequence. Figure 4 (a) shows the first image, (b) shows the fourth image frame, which was taken about 0.3 second

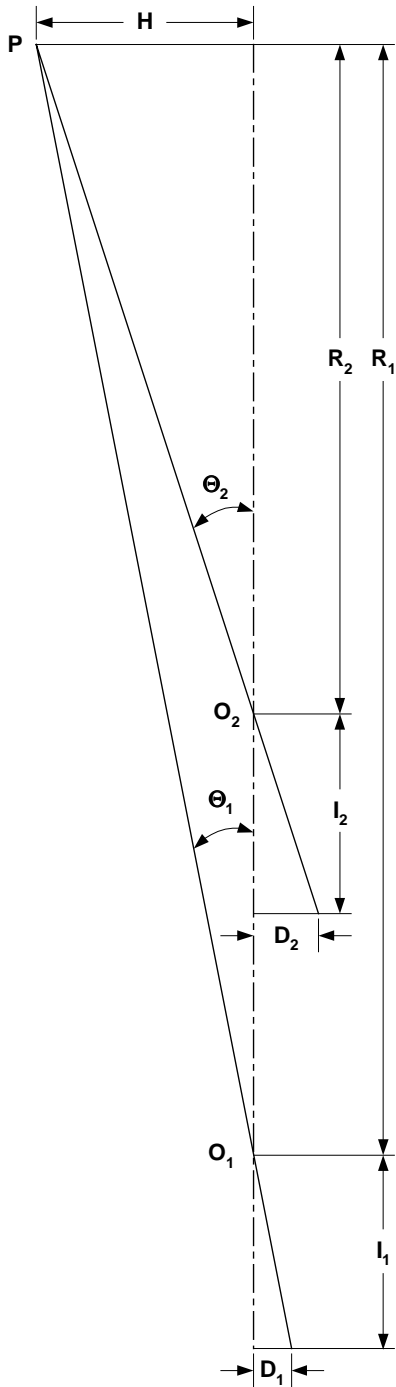
after the first image frame, at that time the camera was 6.5 feet closer towards the objects in the scene.

Table 1 shows one example of the results generated from the study. There are five image sequences taken at three different vehicle speed and using two different types of lenses, narrow and wide angle. Each image sequence was captured while the vehicle was moving at supposedly a constant speed, which is listed in the table. The frame rate was 10 frames per second except for the last one in Table 1, which was captured at 15 frames per second. In the table, L is the distance that the vehicle moved between two successive image frames, “†” indicates that the object is not in the image, “‡” indicates the object is in the image but too small to be measured. For each object, we manually measured the physical distance of the object in the scene that is listed as “measured” under “Depth of Object,” the distance calculated by DepthFinder is listed as “calculated,” and the “deviation” column lists the difference between the measured distance and the calculated distance in percentage. All the distance values in the table are in feet. All our experiments show that the DepthFinder system always predicts correct relative depth, namely, objects appear closer to the camera are shown shorter distance values by the system. For the absolute depth, namely the calculated distance of an object to the camera, the deviations shown in Table 1 are actually smaller than the theoretical estimation. The deviation of absolute depth is attributed to by the following factors:

- Due to the inconvenience of measuring long distances and the incoherence in depth for objects such as tree, the measured distances of the same object can vary within ± 10 feet or more.
- The limited camera resolution.
- The unparallelled camera motion.

In theory the triangular method widely used in the computer vision field assumes a camera to have infinitely high resolution. In practice, all cameras have only limited resolution. In the one-camera configuration, the error due to the limited camera resolution is a function of object angle relative to the center of the view. As the distance increases, the distortion increases. In the monocular image sequence, the baseline used in depth calculation is replaced by the estimation of the vehicle speed and the time interval between two successive image frames. During the camera motion if the optical axis of the camera is not parallel to its previous optical axis, the image of the object moves away from its desired location on the image plane. In one camera configuration, the optical axis of the camera at time t is most likely not parallel to the optical axis at $t+1$ and this is the major cause of the deviation shown in Table 1.

Currently, we are working on the enhancement of the corresponding algorithms and study the monocular vision system on axial and lateral views.



O_1 — The position of the lens at moment t_1 .

O_2 — The position of the lens at moment t_2 .

The vehicle is moving along O_1O_2 . O_1O_2 is also the common optical axis of the camera at both positions.

P — The object, or the point of interest on the object.

H — The distance between P and the optical axis.

R_1 — The projection of the distance between P and the lens at moment t_1 on the optical axis.

R_2 — The projection of the distance between P and the lens at moment t_2 on the optical axis.

I_1 — The distance between the lens and the image plane at moment t_1 .

I_2 — The distance between the lens and the image plane at moment t_2 .

I_1 and I_2 should be very close.

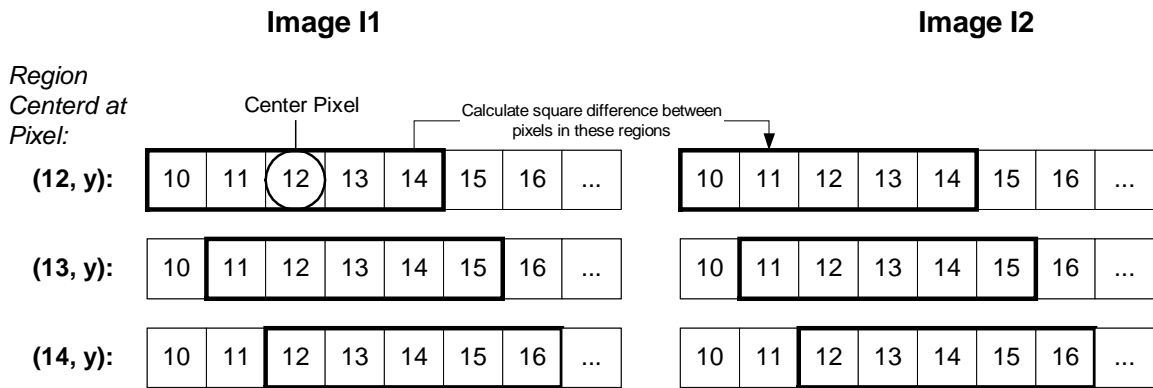
D_1 — The location of P on the image plane at moment t_1 .

D_2 — The location of P on the image plane at moment t_2 .

Θ_1 — The viewing angle of P in the image at moment t_1 .

Θ_2 — The viewing angle of P in the image at moment t_2 .

Figure 1. The geometric model of extracting depth from one camera.



Calculations: (x, x') represents $(\lambda(x, y) - \lambda(x', y'))^2$

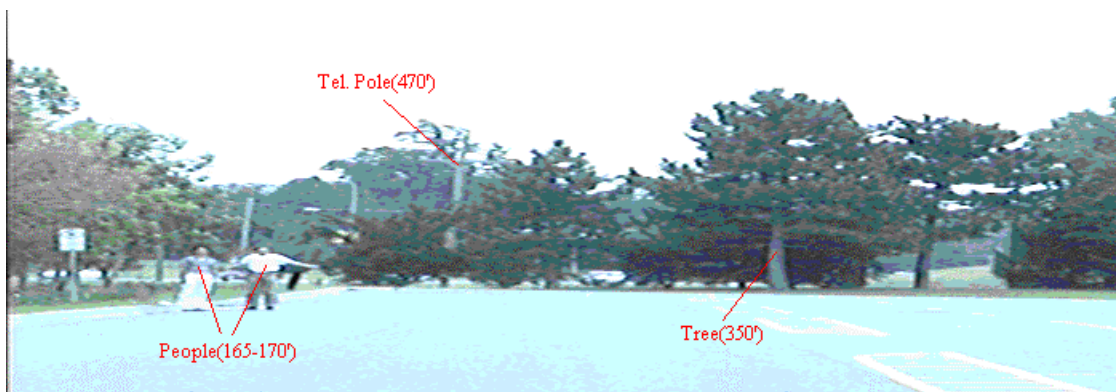
(12, y): $(10, 10') + (11, 11') + (12, 12') + (13, 13') + (14, 14')$

(13, y): $(11, 11') + (12, 12') + (13, 13') + (14, 14') + (15, 15')$

(14, y): $(12, 12') + (13, 13') + (14, 14') + (15, 15') + (16, 16')$

Indicates redundant calculations

Figure 3: Example of calculating $\Phi(u_i, v_i)$ for 3 regions centered at pixel's $(12,y)$, $(13,y)$, and $(14,y)$ respectively. The region size is $q = 5$, $q = 1$, and $u_i = v_i = 0$ is fixed. Note that each row of pixels in the figure is the same row (y) of the image; however, the region has shifted.



(a) The first image frame.



(b) The 4th image frame.

Figure 4. An example of depth finding in an outdoor driving test.

Vehicle Speed (mph)	Lens Type	L (feet)	Depth of Object #1 (People)			Depth of Object #2 (Pole)			Depth of Object #3 (Tree)		
			measured	calculated	deviation	measured	calculated	deviation	measured	calculated	deviation
15	narrow-angle	6.5	166	176	6%	470	444	5.5%	350	288	17.7%
25	narrow-angle	7.3	166	196	15.3%	†	†	†	350	277	20.9%
15	wide-angle	17.5	166	‡	‡	250	270	8.0%	350	455	30.0%
25	wide-angle	18.2	166	172	3.6%	250	256	2.4%	350	328	6.3%
35	wide-angle	17.0	166	‡	‡	250	225	10.0%	350	442	26.3%

Table 1. Examples of experimental results.

5 Acknowledgment

This work was supported in part by a contract from U. S. Army TACOM, Vetricons Department.

6 References

- [1] A. Mitiche, "Motion understanding: robot and human vision", Kluwer, Academic, Boston, pp. 81-99, 1988.
- [2] B. Sridhar and R. Suorsa, "Integration of motion and stereo sensors in passive ranging systems", The American Control Conference, 1990.
- [3] Arun K. Dalmia and Mohan Trivedi, "Depth extraction using a single moving camera: an integration of depth from motion and depth from stereo", Machine Vision and Applications, Vol.9, pp. 43-55, 1996.
- [4] Yi Lu et al., "Dyta: An Intelligent System for Moving Target Detection," International Conference on Image Analysis and Processing, Sept. 1999
- [5] Juyang Weng, Thomas S. Huang, Narendra Ahuja, Motion and Structure from Image Sequences, Springer-Verlag, 1993