Monitoring Weaving Sections

Prepared by:
Osama Masoud
Scott Rogers
Nikolaos P. Papanikolopoulos

Artificial Intelligence, Robotics, and Vision Laboratory
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

October 2001

Published by:

ITS Institute
University of Minnesota

CTS 01-06

Executive Summary

Traffic control in highway weaving sections is complicated since vehicles are crossing paths, changing lanes, or merging with through traffic as they enter or exit an expressway. There are two types of weaving sections: (a) single weaving sections which have one point of entry upstream and one point of exit downstream; and (b) multiple weaving sections which have more than one point of entry followed by more than one point of exit. Sensors which are based on lane detection fail to monitor weaving sections since they cannot track vehicles which cross lanes. The fundamental problem that needs to be addressed is the establishment of correspondence between a traffic object A in lane x and the same object A in lane y at a later time. For example, vision systems that depend on multiple detection zones simply cannot establish correspondences since they assume that the vehicles stay in the same lane.

The motivation behind this work is to compensate for inefficiencies in existing systems as well as to provide more comprehensive data about the weaving section being monitored. We have used a vision sensor as our input. The rich information provided by vision sensors is essential for extracting information that may cover several lanes of traffic. The information that our system provides includes (but is not limited to): (a) Extraction of vehicles and thus a count of vehicles, (b) Velocity of each vehicle in the weaving section, and (c) Direction of each vehicle (this is actually a trajectory versus time rather than a fixed direction, since vehicles may change direction while in a highway weaving section). The end-product of this research is a portable system that can gather data from various weaving sections. Experimental results indicate the potential of the approach.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

USER INTERFACE AND CALIBRATION TOOL

INTRODUCTION

This chapter describes the complete calibration tool and user interface that is used for calibrating transportation scenes. Processing of any image requires some knowledge of the pan and tilt angles of the camera, along with the mounting height and the distance of the camera from the scene being monitored. Accurate knowledge of these parameters can significantly impact the computation of such things as the vehicle velocities and how vehicles are classified. It does not affect the general vehicle counts. These parameters are not easily obtained. A user interface has been developed so that the user can point to some locations on the image (e.g., the endpoints of a lane divider line whose length is known). Then, the system can compute the calibration parameters automatically. An additional feature of the interface is that it allows the user to define traffic lanes in the video, and the direction of traffic in them.

PREVIOUS WORK

In this project, calibration of the algorithms required explicit knowledge of the camera position and orientation with respect to the road. These parameters are difficult to measure

directly, especially while one is away from the scene, which is often the case. It should be noted that even at the scene, the user may have difficulty determining these parameters. One way to compute them is to look at the known facts about the scene. For example, we know that the road, for the most part, is restricted to a plane. We also know that the lane markings are parallel and lengths of markings as well as distances between those markings are known numbers. We can use these to calculate the position and orientation of the camera with respect to those lane markings.

Previously the various points of the image were determined using the image editing software Paint Shop Pro and these values were then processed in Mathematica to determine the camera position and orientation. Aside from being a laborious and time consuming task, the Mathematica algorithm made several assumptions that are not reasonable in the more general case. The system described here combines a method of marking an image of the scene as well as using these marks to calculate the calibration values needed. The proposed system is easy to use and intuitive to operate, using obvious landmarks, such as lane markings, and familiar tools, such as a line-drawing tool.

DESCRIPTION

The Graphical User Interface (GUI) prompts the user to first open a bitmap image of the scene. The user is then able to draw different lines and optionally assign lengths to those lines. The user may first draw lines that represent lane separation (green, Figure 1). They may then draw lines to designate the width of the lanes (red, Figure 1). The user may also designate known lengths in conjunction with the lane separation marks (). The blue line is used to mark special hot spots in the image, such as the location where we want to compute vehicles speeds.

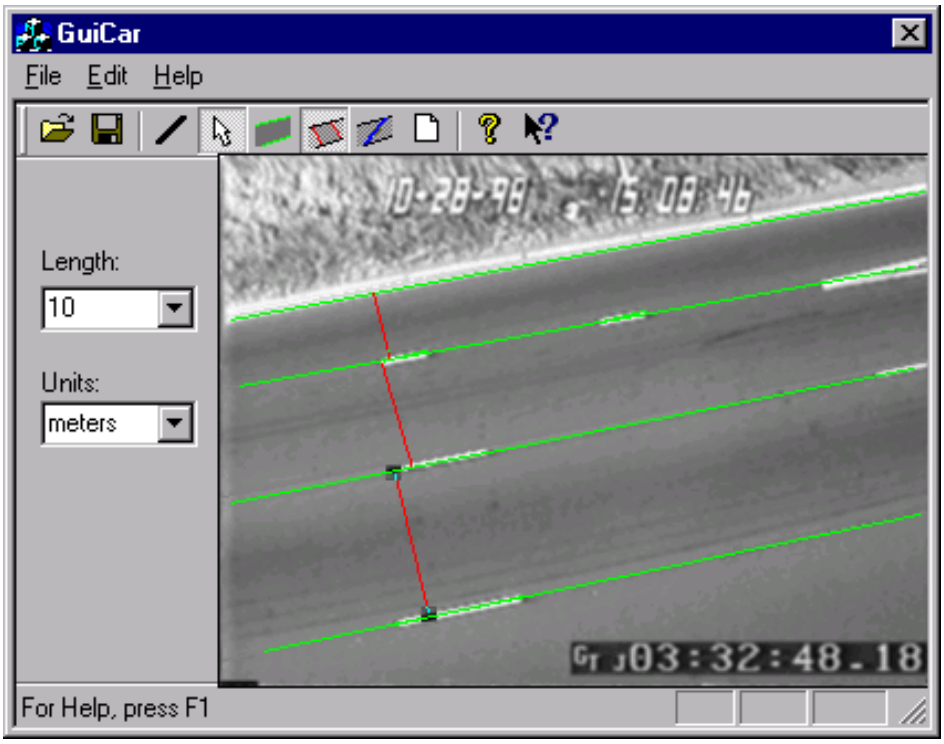


Figure 1. Definition of parallel lines and distances between them

From the orientation of the green lines, we can find the vanishing point of the road and all lines parallel to it. Once we obtain perpendicular lines and distances and various distances along the lane markings we can determine those parameters of the camera such as distance, pitch, roll, etc. to sufficiently calculate velocities and relative position within the scene. How these parameters are determined from the information given by the GUI is fully outlined in the following chapter. It is sufficient for now to know that these parameters can be obtained by simply marking what is already known in the scene using this GUI.

RESULTS

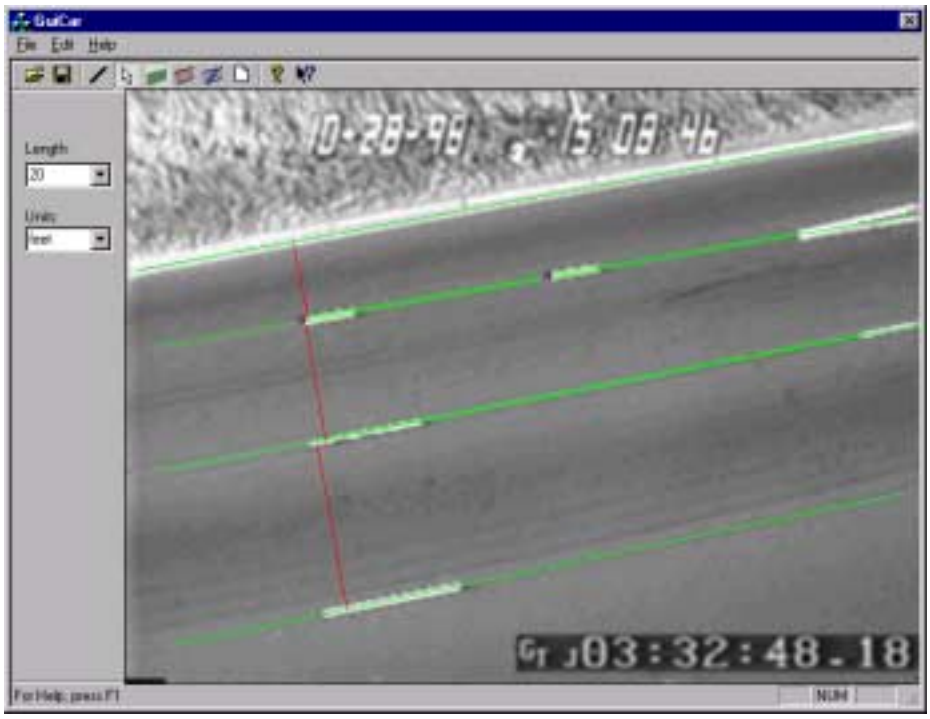


Figure 2. Definition of distances along the lane markings

The GUI interface has proven to be much more intuitive than the previous methods. The only real difficulty arose with respect to accuracy in determining distances in the direction of the road. Some of these inaccuracies arise because the markings on the road themselves are not precise. The user s ability to mark endpoints in the image also affects accuracy.

In general, however, in spite of the inaccuracies discovered, this method of calibration proved to be much quicker than those previous used, equally accurate, and more adaptable to generic scenes.

CHAPTER 2

A CAMERA CALIBRATION ALGORITHM FOR TRAFFIC SCENES

INTRODUCTION

Camera calibration for traffic scenes is necessary to obtain accurate information about position and velocity from a vision system. Unfortunately, calibration can be a tedious process when done outside a laboratory. Many calibration techniques that rely on taking images of a grid are impractical in this case. In this work, we propose to use cues from the scene itself to perform camera calibration. The algorithm we propose performs a minimization on camera parameters to fit the data given to it. This algorithm is the backend to the graphical user interface described in the previous chapter.

CAMERA CALIBRATION

In the most general case, camera calibration involves finding the camera s intrinsic and extrinsic parameters. Intrinsic parameters are usually given by the matrix

$$A = \begin{bmatrix} \alpha_u & -\alpha_u \cot\theta & u_0 \\ 0 & \frac{\alpha_v}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The parameter α_u corresponds to the focal length in pixels along the horizontal axis of the image. In fact, $\alpha_u = fk_u$, where f is the focal length and k_u is image sensor resolution given in pixels per unit length. The two terms are not separable and therefore, only their product (α_u) can be recovered. α_v is similar but corresponds to the vertical axis. It is equal to α_u when the sensor has square pixels. The horizontal and vertical axes may not be exactly perpendicular. The parameter θ is the angle between them. The optical axis may not intersect the image plane at the center of the image. The coordinates of this intersection are given by (u_0, v_0). In addition to these parameters, there are parameters that can be used to model radial lens distortion.

The extrinsic parameters specify the location and orientation of the camera and can be given by the matrix

$$T = [R|t], \tag{2}$$

where R is a rotation matrix and $\|\mathbf{P}_1 - \mathbf{P}_2\|$ is a translation vector. Therefore, the total number of extrinsic parameters is six. These describe the pose of the camera in some world-coordinate system.

There has been a considerable amount of research that deals with camera calibration (both intrinsic and extrinsic). The calibration process normally involves taking a few images and matching some features. To achieve a good level of accuracy, one can take images of a specially constructed grid and then match corners (either manually or automatically).

TRAFFIC SCENES

Classical calibration techniques are not suitable for traffic cameras simply because in most cases, it is not feasible to place calibration objects in the scene, for that may interfere with

traffic. It is also not feasible to take multiple images from different locations in the scene since that would require mounting the camera at different locations (simply panning the camera does not help in calibration). It is highly desirable to be able to perform calibration using the images captured by the camera.

Of all the intrinsic parameters, the focal length (and hence, α_u and α_v), is the only parameter that could change due to changing the level of magnification (zoom). The ratio between the two (the squareness of pixels) can be easily computed at the laboratory or even looked up in the camera specs. The rest of the intrinsic parameters usually remain constant and can be calibrated for at the laboratory. Moreover, making the assumption that $\theta = 90°$ and (u_0, v_0) to be at the center of the image works well enough since they are rarely different from that. This reduces to one the number intrinsic parameters that must be estimated.

As for extrinsic parameters, traffic scene geometry simplifies the situation. The motion in a traffic scene can be assumed to be constrained to a plane (the road). This is also known as the *ground-plane constraint* (GPC) [27]. The number of extrinsic parameters can therefore be reduced from six to four: camera roll, pitch, yaw, and elevation above the plane. Therefore, the total number of parameters (both intrinsic and extrinsic) that need to be estimated is five. One of these parameters (implicit in elevation) is the scale which is trivial to estimate and can be determined as a last step by taking one measurement in the real-world coordinate system. This leaves four nontrivial parameters.

Previous work in camera calibration for traffic scenes was done by Worral *et al.* [27]. They developed an interactive tool that allows the user to pick two parallel lines in the scene to find the vanishing point. This reduces the number of free variables from four to two. The two remaining parameters are camera roll about an axis through the vanishing point and the focal

length. The former is set by allowing the user to slide a bar and watch the effect on a grid overlaid on the road until the grid appears to be squared with respect to the road. The focal length can be set in a similar way or by providing a second measurement on the road (perpendicular to that used to determine the scale).

The main shortcoming of this approach is that what appears squared with respect to the road can be subjective and hard to judge correctly. This is especially true when the road lanes converge weakly (see Figure 3). After the completion of the calibration process, the grid may appear to be lying correctly on the road.

In our approach, we propose to use more information taken from the scene itself in the form of measurements. Besides the fact that this may be the only available information, there are two other motivations for this approach. First, lane width and lane marking lengths and separation comply with highway standards and therefore are more accurate than other cues in the scene. Secondly, the purpose of camera calibration is to obtain accurate data regarding velocities and distances and therefore, it makes sense to force the calibration to conform with as many measurements scattered in the region of interest as possible.

In some cases, when lane markings are not available or are known to be inaccurate, it may be possible to take some measurements manually. These measurements need to be in the scene but not necessarily on the road to avoid the need to interrupt traffic.

CALIBRATION ALGORITHM

Our calibration algorithm utilizes GPC as in [27]. However, the user does not need to make any decisions based on visual appearance. Instead, the user identifies features in the image and provides measurement information. The algorithm performs an optimization to minimize the error in the model conforming with these measurements. There are two main

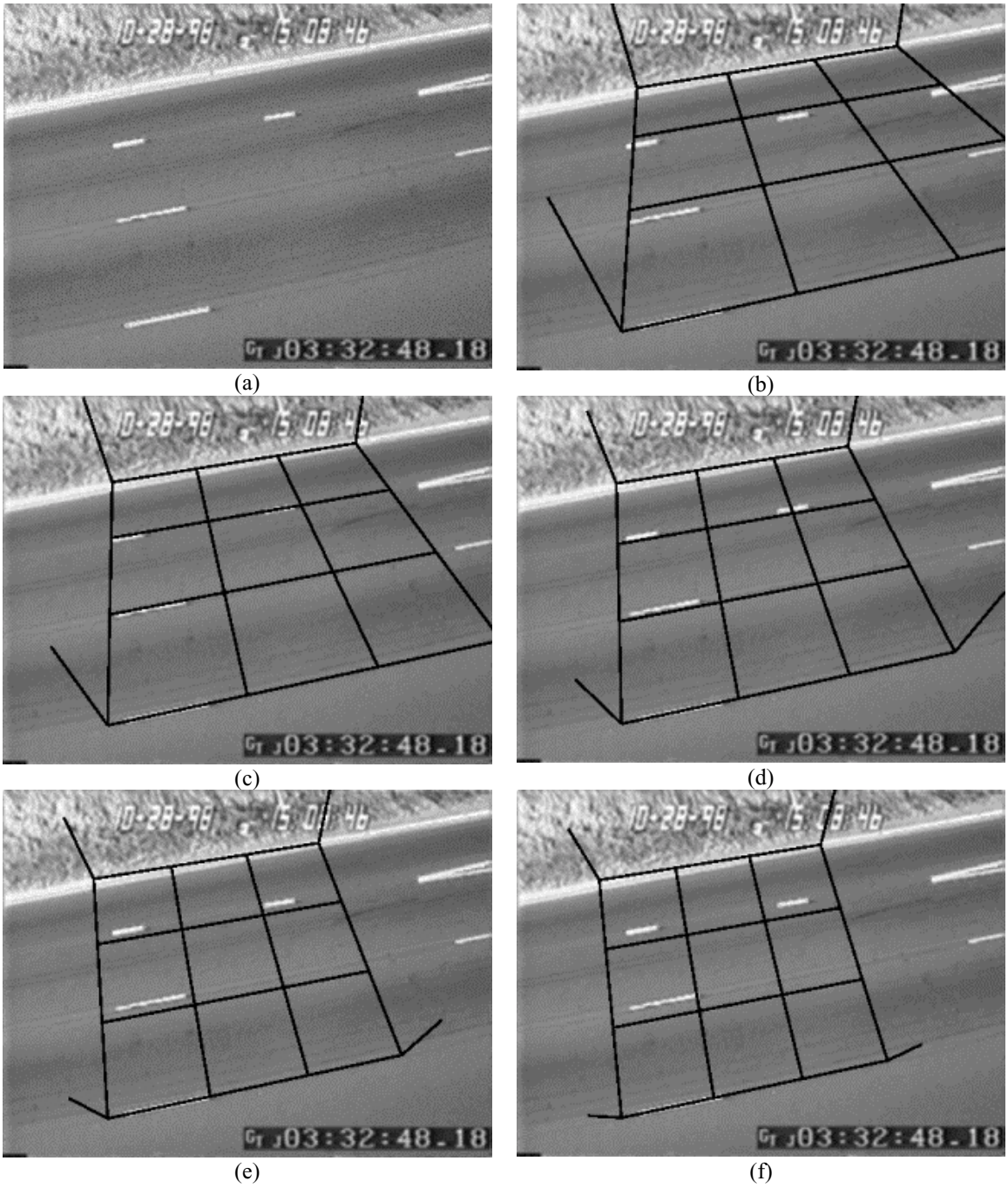
stages in the process: identifying parallel lines (used to find the vanishing point) and providing real-world distances.

Camera Coordinate System

We use a camera coordinate system similar to [27], where the X and Z axes are parallel to the image plane and the Y axis is along the optical axis. Without loss of generality, we assume that the optical axis intersects the ground plane at the point $\begin{bmatrix}0 & 1 & 0\end{bmatrix}$. The exact point of intersection depends on the scale, which can be computed using one real-world measurement.

Vanishing Point

A minimum of two parallel lines is necessary. In this case, the vanishing point is the apparent intersection of the two. For better accuracy, the user can specify more lines and the intersection point is then computed as the point whose sum of squared distances to all the lines is minimum. This point, $\mathbf{p}_i$, is easily computed by solving a system of linear equations of the form $\mathbf{Mx} = \mathbf{b}$. $\mathbf{M}$ and $\mathbf{b}$ have as many rows as there are lines. If $\mathbf{l}_i$ is a unit vector representing the direction of the ith line in the image and $\mathbf{p}_i$ is a point on that line, then $\mathbf{M}_i = \begin{bmatrix}-\mathbf{l}_{i2} & \mathbf{l}_{i1}\end{bmatrix}$ and $\mathbf{b}_i = \left(\mathbf{l}_i \times \mathbf{p}_i\right)^T \cdot \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$.



(a) (b) (c) (d) (e) (f)

Figure 3. Different camera roll angles (with respect to the vanishing direction) visualized with the aid of an overlaid grid with attached normals. (a) Original image. (b), (c), (d), (e), (f) show the grid at a roll angles of 30, 37.5, 45, 52.5, and 60 degrees, respectively. The correct angle is shown in (c). It is difficult to discern the correct angle based on appearance in this case due to the lack of other visual cues on the road. The fact that the grid lines do not coincide with the middle lane markings cannot be used as a cue since the focal length can be always chosen to make them coincide.

The vanishing direction, $\mathbf{v}$, is then computed as $\mathbf{A}^{-1}\mathbf{x}$, where $\mathbf{x}$ is the vanishing point and $\mathbf{A}$ is the camera intrinsic parameters matrix.

Ground Plane Coordinate System

The ground plane is described by three unit vectors: $\mathbf{G}_x$ which is perpendicular to the vanishing direction, $\mathbf{G}_y$ which coincides with the vanishing direction, and the plane normal $\mathbf{G}_z$ which completes the right-hand coordinate system. Let $\hat{\mathbf{v}} = [v_x v_y v_z]$ be the normalized vanishing direction. Therefore, $\mathbf{G}_y = \hat{\mathbf{v}}$. It can be shown [27] that

$$\mathbf{G}_x = \begin{bmatrix} (1-\beta v_x^2)\cos\varphi + \beta v_x v_z \sin\varphi \\ v_z \sin\varphi - v_x \cos\varphi \\ -(1-\beta v_z^2)\sin\varphi + \beta v_x v_z \cos\varphi \end{bmatrix}^T \text{ and} \tag{3}$$

$$\mathbf{G}_x = \begin{bmatrix} (1-\beta v_x^2)\cos\varphi + \beta v_x v_z \sin\varphi \\ -v_z \sin\varphi - v_x \cos\varphi \\ (1-\beta v_z^2)\sin\varphi + \beta v_x v_z \cos\varphi \end{bmatrix}^T \tag{4}$$

where $\beta = 1/(1+v_y)$ and φ is the roll angle about the vanishing direction. The formulas provide a complete parameterization of the plane in the two unknowns α_u and φ.

Distances

To compute the distance between the projections of two image points on the ground plane, we first compute the projections. Given an image point $\mathbf{x}, \mathbf{p} = \mathbf{A}^{-1}\mathbf{x}$ is a vector in the direction of the ray from the camera center through $\mathbf{x}$. We can solve for the intersection of this ray and the ground plane to obtain $\mathbf{P} = \frac{\mathbf{G}_{z2}}{\hat{\mathbf{p}} \cdot \mathbf{G}_z}\hat{\mathbf{p}}$, where $\mathbf{G}_{z2}$ is the second element of $\mathbf{G}_z$, and $\hat{\mathbf{p}} = \frac{\mathbf{p}}{\|\mathbf{p}\|}$.

Given two projections on the ground plane $\mathbf{P}_1$ and $\mathbf{P}_2$, the distance is simply $\|\mathbf{P}_1 - \mathbf{P}_2\|$. We can also compute the distance between $\mathbf{P}_1$ and $\mathbf{P}_2$ along the direction of $\mathbf{G}_x$ or $\mathbf{G}_y$ as $|(\mathbf{P}_1 - \mathbf{P}_2) \cdot \mathbf{G}_x|$ and $|(\mathbf{P}_1 - \mathbf{P}_2) \cdot \mathbf{G}_y|$. This is particularly useful in case the measurements provided are lane width measurements or lane marking measurements.

Parameter Estimation

To estimate α_u and φ, we use the Levenberg-Marquadt nonlinear least squares method. The residual we try to minimize is completely dependent on the relationship among the scene measurements. We use one measurement, m_0, as a reference and relate all other measurements to it. The residual is computed as

$$r = \sum_{i=1}^{n} \left(\frac{d_i m_0}{d_0 m_i} - 1 \right)^2 \tag{5}$$

where m_i are the measured distances and d_i are the computed distances.

Finally, the scale is computed as $\frac{m_0}{d_0}$.

RESULTS

Our algorithm was tested on a variety of real-time scenes and was always able to converge and closely match the given measurements. To test the behavior in a more systematic way, we constructed a sequence of scenes with different vanishing directions, camera locations and zoom levels. These are shown in Figure 4, Figure 5, and Figure 6. The data provided in each case was the four parallel lanes with distances separating them, and a total of 6 measurements along the middle two lanes. The algorithm converged in each case to a solution that minimized the cost function (5). Table 1 shows the result for each case as $\sqrt{r/n}$, where n

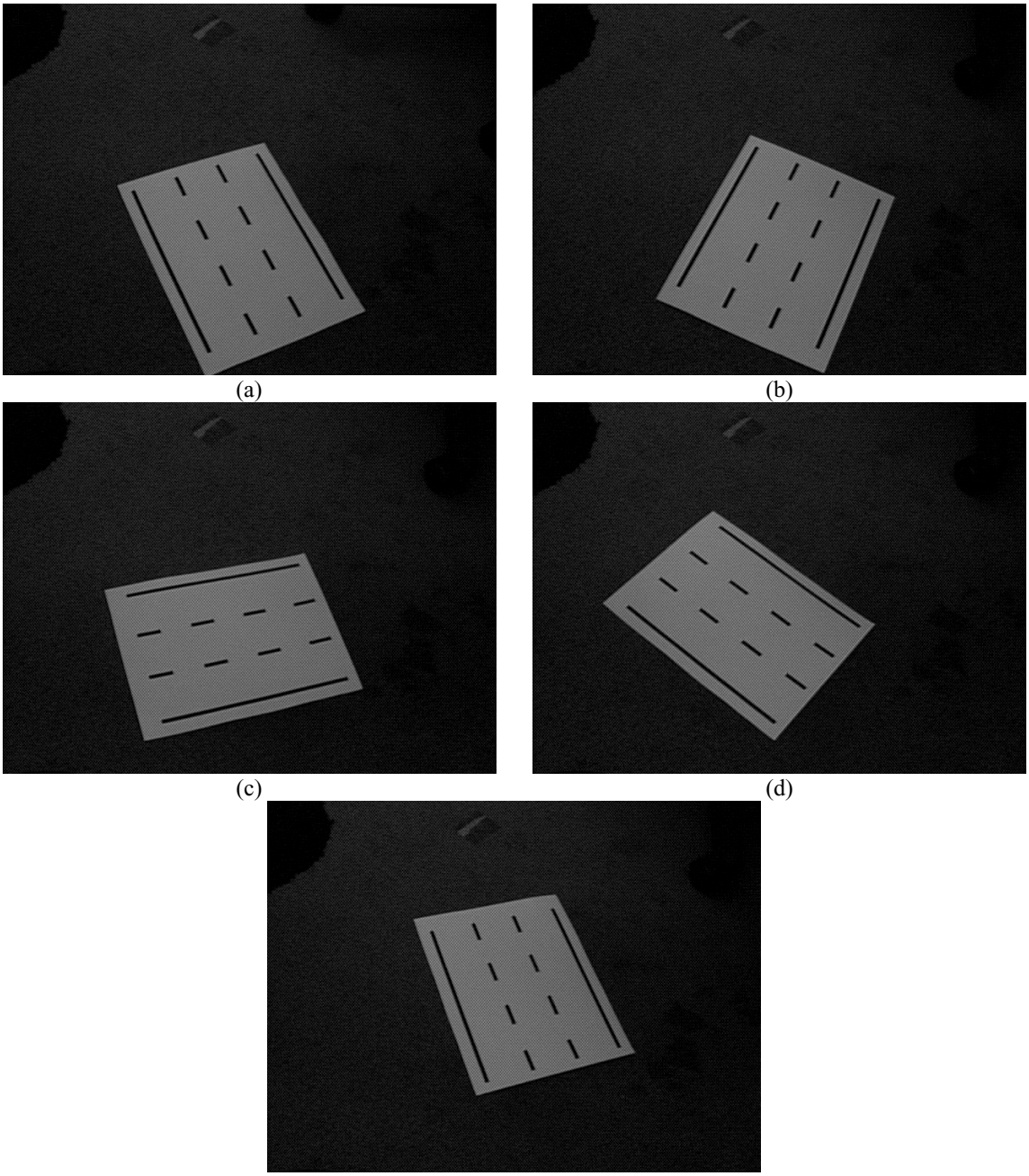
is the number of measurements used. The results show a very small discrepancy between the real and the estimated measurement ratios.

Scene	RMS error
1(a)	0.0306
1(b	0.0154
1(c)	0.0600
1(d)	0.0194
1(e)	0.0148
2(a)	0.0066
2(b)	0.0087
2(c)	0.0184
2(d)	0.0201
3(a)	0.0161
3(b)	0.0453
3(c)	0.0236

Table 1. RMS errors for each scene.

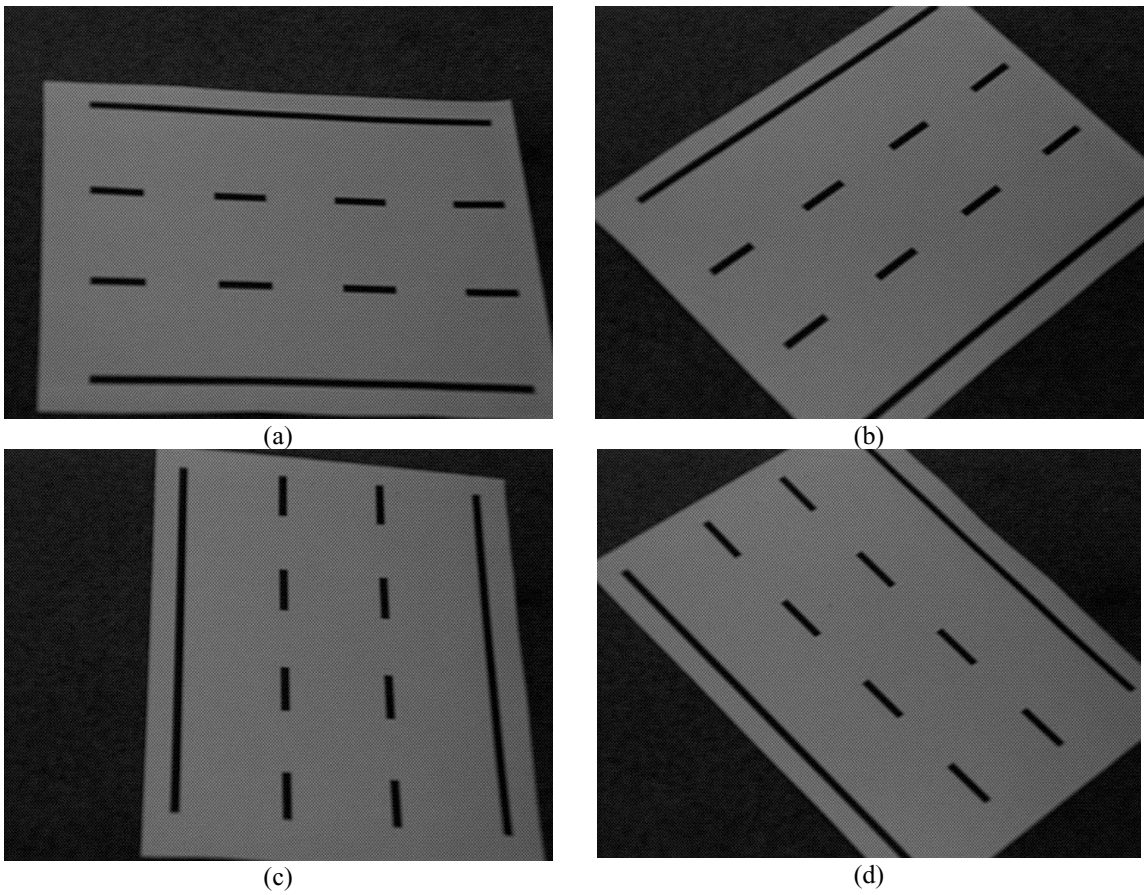
CONCLUSION

We presented an algorithm to perform camera calibration suited to traffic scenes. The algorithm uses measurements from the scene which can be provided by a graphical user interface. The results demonstrate the accuracy of this technique in many different camera/scene configurations.



(a) (b) (c) (d) (e)

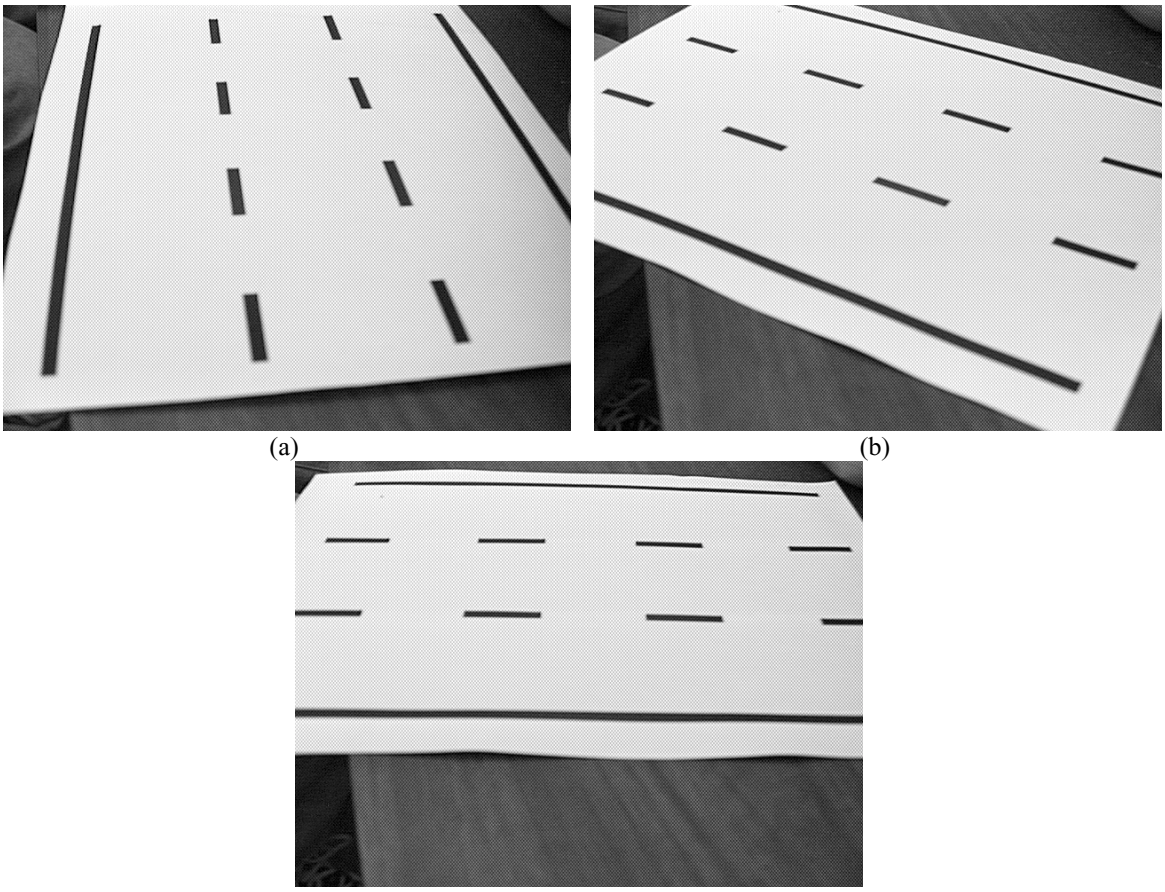
Figure 4. Near vertical pitch, far zoom



(a) (b)

(c) (d)

Figure 5. Near vertical pitch, close zoom



(a) (b)

(c)

Figure 6. Near horizontal pitch, close zoom

CHAPTER 3

MONITORING WEAVING SECTIONS

Traffic control in weaving sections is a challenging problem because vehicles are crossing paths, changing lanes, or merging with through traffic as they enter or exit a highway. There are two types of weaving sections depending on the number of points of entry and the number of points of exit: single weaving sections and multiple weaving sections. Sensors which are based on lane detection or tripline detection often fail to accurately monitor weaving sections since they cannot track vehicles which cross lanes. The fundamental problem that needs to be addressed is the establishment of correspondence between a traffic object A in lane x and the same object A in lane y at a later time. For example, vision systems which depend on multiple detection zones [18,25] simply cannot establish correspondences since they assume that the vehicles stay in the same lane.

The motivation behind this work is to compensate for inefficiencies in existing systems as well as to provide more comprehensive data about the weaving section being monitored. In this work, we use visual information as our sensory input. The rich information provided by vision sensors is essential for extracting information that may cover several lanes of traffic. The information we are considering includes:

1. Extraction of vehicles and thus a count of vehicles.
2. Velocity of each vehicle in the weaving section.

3. Direction of each vehicle. This is actually a trajectory versus time rather than a fixed direction (since vehicles may change direction while in a highway weaving section).

Our goal is to build a portable system which can gather data in various settings (something similar to the workzone variable message sign units). Such a system can have a wide range of uses including:

1. Data collection for weaving sections,
2. Optimizing traffic control at weaving sections,
3. Real-time traffic simulation, and,
4. Traffic prediction.

Our system uses a single fixed camera mounted in an arbitrary position. We use simple rectangular patches with a certain dynamic behavior to model vehicles. Overlaps among vehicles and occlusions are dealt with by allowing vehicle models to overlap in the image space and maintain their existence in spite of the disappearance of some features.

A large body of vision research has been targeted at vehicle tracking. One popular research direction deals with three-dimensional tracking. Three-dimensional tracking uses models for vehicles and aims to handle complex traffic situations and arbitrary configurations. A suitable application would be conceptual descriptions of traffic situations [8]. Robustness is more important than computational efficiency in such applications. Kollnig and Nagel [14] developed a model-based system. They proposed to use image gradients instead of edges for pose estimation. They fitted image gradients to synthetic model gradients projected onto the image. A Kalman filter was used to stabilize the tracking. Optical flow was used to generate object candidates at the initialization stage only. Selection of the model was done manually for each vehicle since the emphasis was on robust tracking as opposed to classification. In [13], the

same authors increased robustness by utilizing optical flow during the tracking process as well. Nagel *et al*. [19] and Leuck and Nagel [15] extended the previous approach to estimate the steering angle of vehicles. This was a necessary extension to handle trucks with trailers which were represented as multiple linked rigid polyhedra [19]. Experimental results in [15] compared the steering angle and velocity of a vehicle to ground truth showing good performance. They also provided qualitative results for other vehicles showing an average success rate of 77%. Tracking a single vehicle took 2-3 seconds per frame.

Three-dimensional tracking of vehicles has been extensively studied by the research group at the University of Reading. Baker and Sullivan [2] and Sullivan [21] utilized knowledge about the scene in their 3D model-based tracking. This includes knowledge that the vehicles move on a plane, the camera calibration, the vehicles, and their dynamics. The ground plane assumption reduces the tracking search parameters to a translation and a rotation. The matching is also done by comparing the projection of the model to features in the image. Sullivan *et al*. [23] extended this approach so that image features act as forces on the model. This reduced the number of iterations and improved performance. They also parameterized models as deformable templates and used principal component analysis to reduce the number of parameters. A filter that was used by Maybank *et al*. [16] to stabilize tracking was demonstrated to surpass the extended Kalman filter when vehicles undergo complex motion such as a three-point turn.

Three-dimensional methods are sensitive to initial pose estimate. Some interesting work on extracting initial pose from static images was done in [24]. In addition, pose estimation and tracking can be improved by using more refined models [14] which would require a large set of models and therefore greater computational resources.

In the case of applications that require monitoring of controlled traffic situations, it is usually possible to implement a much more efficient, yet still robust, system. Sullivan *et al*. [22] developed a simplified version of their model-based tracking approach to achieve real-time performance. The method was not designed to track vehicles across lanes, however.

Another tracking approach is feature-based. Features, such as corners and edges, are tracked and grouped based on their spatial and temporal characteristics. This approach is robust with respect to occlusions. However, the density of reliable features may not be high enough in certain circumstances. If the application requires isolation of vehicles (e.g., a classification application), there may be a need for an additional component to the tracking system. Feature tracking is considered a computationally expensive operation due to the use of correlation. Smith [20] used custom hardware to achieve tracking at 25Hz. Beymer *et al*. [3] used 13 C40 DSPs to achieve up to 7.5Hz tracking in uncongested traffic. This may not remain a big disadvantage in the long run, however, considering the constant increase in computing power of ordinary PCs. Active deformable contours were also used to track vehicles [12]. However, initialization can be problematic especially for partially occluded vehicles. Combining (or switching among) different approaches has been considered as a way of dealing with diverse weather and lighting conditions. Some algorithms have been developed to quantify scene illumination conditions (well lit, poorly lit, has shadows, etc.) [26] and road conditions (wet, dry, snow, etc.) [28]. A method for combining multiple algorithms was given in [7].

Finally, some researchers tracked connected regions in the image (blobs) [10,11,4,5]. Region-based tracking is perhaps the most computationally efficient but in general suffers from segmentation problems in cluttered scenes. This happens because blobs of several vehicles may merge into a single blob. Alternatively, a single vehicle may be composed of several blobs. In

[4], the authors use a rule-based system to refine the tracking and resolve low-level segmentation errors. The system we describe in this work is region-based. Our method differs from other region-based methods by the way it handles relating blobs to vehicles. This relation is allowed to be many-to-many, and is updated iteratively depending on the observed blobs behavior and predictions of vehicles behavior. Processing is done at three levels (this structure is based on our earlier work on pedestrian tracking [17]). Features are obtained at the lowest level by a simple recursive filtering technique. In the second level, which deals with blobs, feature images are segmented to obtain blobs which are subsequently tracked. Finally, tracked blobs are used in the vehicles level where relations between vehicles and blobs as well as information about vehicles is inferred.

The next two chapters describe the processing done at the three levels mentioned above.

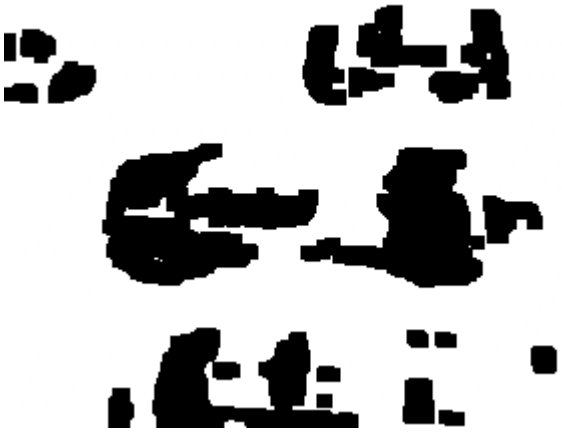


Figure 7. Feature image (corresponds to the image in Figure 11(c))

CHAPTER 4

BLOB ANALYSIS

FEATURES LEVEL

Features are extracted using a recursive filtering technique (a slightly modified version of Halevi and Weinshall s algorithm [9]). This technique is simple, time-efficient and therefore, suitable for real-time applications. A weighted average at time i, M_i, is computed as

$$M_i = \alpha \times I_{i-1} + (1-\alpha) \times M_{i-1} \tag{1}$$

where I_i, is the image at time i, and α is a fraction in the range 0 to 1 (typically 0.5). The feature image at time i, F_i, is computed as follows: $F = |M_i - I_i|$. The feature image captures temporal changes (features) in the sequence. Moving objects result in a fading trail behind them. The feature image is thresholded to remove insignificant changes. Figure 7 shows a typical thresholded feature image.

As it will be explained below, our algorithm makes use of bounding boxes in blob tracking. The bounding boxes track most effectively when the targeted blobs fill the box, which is achieved when they are aligned horizontally or vertically (i.e., diagonal blobs are not desirable). Since the camera pan may be large as shown in Figure 8, we warp the original image so that vehicles appear perpendicular to a virtual camera's optical axis (Figure 12 (b) is

the warped version of Figure 8), and then perform feature extraction. Apart from camera pan, rotated blobs can appear due to the diagonal motion of a weaving vehicle. However, this rotation is small and does not affect our algorithm since it occurs over a long distance compared to lane width. The warping process uses extrinsic camera parameters such as location and orientation to perform inverse perspective projection. These parameters are estimated interactively to a high precision at initialization by utilizing ground truth measurements in the scene (such as lane width, lane marking length, etc.). The tracking algorithm is not sensitive to errors in these parameters. However, since these parameters are also used at the vehicles level to provide information on vehicle speed and location, precise estimation of these parameters is desired. In [27], a simple interface has been developed which requires minimal operator interaction and leads to good precision estimates.

BLOBS LEVEL

At the blobs level, blob extraction is performed by finding connected regions in the feature image. A number of parameters are computed for each blob. These parameters include perimeter, area, bounding box, and density (area divided by bounding box area). We then use a novel approach to track blobs regardless of what they represent. Our approach allows blobs to merge, split, appear, and vanish. Robust blob tracking was necessary since the vehicles level relies solely on information passed from this level.

BLOB TRACKING

When a new set of blobs is computed for frame i, an association with the set of blobs in frame $(i-1)$ is sought. The relation between the two sets can be represented by an undirected bipartite graph, $G_i(V_i, E_i)$, where $V_i = B_i \cup B_{i-1}$. Here, B_i and B_{i-1} are the sets of vertices

associated with the blobs in frames i and $i-1$, respectively. We will refer to this graph as a *blob graph*. Figure 9 shows an example where blob 1 split into blobs 4 and 5, blob 2 and part of blob 1 merged to form blob 4, blob 3 disappeared, and blob 6 appeared.

The process of blob tracking is equivalent to computing G_i for $i = 1,2,\mathsf{K}\ ,n$, where n is the total number of frames. We do this by modeling the problem as a constrained graph optimization problem where we attempt to find the graph which minimizes a cost function.

Let $N_i(u)$ denote the set of neighbors of vertex $u \in V_i$, $N_i(u) = \{v | (u,v) \in E_i\}$. To simplify graph computation, we will restrict the generality of the graph to those graphs which do not have more than one vertex of degree more than one in every connected component of the graph. This is equivalent to saying that from one frame to the next, a blob may not participate in a split and a merge at the same time. We refer to this as the *parent structure constraint*. According to this constraint, the graph in Figure 9(c) is invalid. If, however, we eliminate the arc between 1 and 5 or the arc between 2 and 4, it will be a valid graph. This restriction is reasonable assuming a high frame rate where such simultaneous split and merge occurrences are rare.

To further reduce the number of possible graphs, we use another constraint which we call the *locality constraint*. With this constraint, vertices can be connected only if their corresponding blobs have a bounding box overlap area which is at least half the size of the bounding box of the smaller blob. This constraint, which significantly reduces possible graphs, relies on the assumption that a blob is not expected to be too far from where it was in the previous frame. This is also reasonable to assume if we have a relatively high frame rate. We refer to a graph which satisfies both the parent structure and locality constraints as a *valid* graph.

To find the optimum G_i, we need to define a cost function, $C(G_i)$, so that different graphs can be compared. A graph with no edges, i.e. $E_i = \phi$, is one extreme solution in which all blobs in V_{i-1} disappear and all blobs in V_i appear. This solution has no association among blobs and should therefore have a high cost. In order to proceed with our formulation of the cost function, we define two disjoint sets, which we call *parents*, P_i, and *descendents*, D_i, whose union is V_i such that $D_i = \bigcup_{u \in P_i} N_i(u)$. P_i can be easily constructed by selecting from V_i all vertices of degree more than one, all vertices of degree zero, and all vertices of degree one which are only in B_i. Furthermore, let $S_i(u) = \sum_{v \in P_i} A(v)$ be the total area occupied by the neighbors of u. The cost function that we use penalizes graphs in which blobs change significantly in size. A perfect match would be one in which blob sizes remain constant (e.g., the size of a blob that splits equals the sum of the sizes of blobs it splits into). We now write the formula for the cost function as

$$C(G_i) = \sum_{u \in P_i} \frac{|A(u) - S_i(u)|}{\max(A(u), S_i(u))}. \tag{2}$$

This function is a summation of ratios of size change over all parent blobs.

Using this cost function, we can proceed to compute the optimum graph. First, we notice that given a valid graph $G(V,E)$ and two vertices $u,v \in V$, such that $(u,v) \notin E$, the graph $G'\left(V, E \cup \{(u,v),(v,u)\}\right)$ has a lower cost than G provided that G' is a valid graph. If it is not possible to find such a G', we call G *dense*. Using this property, we can avoid some useless enumeration of graphs which are not dense. In fact, this observation is the basis of our algorithm to compute the optimum G.

Our algorithm to compute the optimum graph works as follows: A graph G is constructed such that the addition of any edge to G makes it violate the locality constraint. There can be only one such graph. Note that G may violate the parent structure constraints at this moment. The next step in our algorithm systematically eliminates just enough edges from G to make it satisfy the parent structure constraint. The resulting graph is valid and also dense. The process is repeated so that all possible dense graphs are generated. The optimum graph is the one with the minimum cost. The computational complexity of this step is highly dependent on the graph being considered. If the graph already satisfies the parent structure constraint, it is $O(1)$. On the other hand, if we have a fully connected graph, the complexity is exponential in the number of vertices (bounded by 2^{mn}). Fortunately, because of the locality constraint and the high frame rate, the majority of graphs considered already satisfy the parent structure constraint. Occasionally, a small cluster of the graph may not satisfy the parent structure constraint and the algorithm will need to enumerate a few graphs. In practice, the algorithm never took more than a few milliseconds to execute even in the most cluttered scenes. Other techniques to find the optimum (or near optimum) graph (e.g., stochastic relaxation using simulated annealing) can also be used. The main concern, however, would be their efficiency which may not be appropriate for this real-time application due to their iterative nature.

At the end of this stage, we use a simple method to calculate the velocity of each blob, v, based on the velocities of the blobs at the previous stage and the computed blob graph. The blob velocity will be used to initialize vehicle models as described later. If v is the outcome of a splitting operation, it will be assigned the same velocity as the parent blob. If v is the outcome of a merging operation, it will be assigned the velocity of the largest child blob. If v is

a new blob, it will be assigned zero velocity. Finally, if there is only one blob, u, related to v, the velocity is computed as

$$\mathbf{V}(v) = \beta \frac{(\mathbf{b}_v - \mathbf{b}_u)}{\delta t} + (1-\beta)\mathbf{V}(u) \tag{3}$$

where $\mathbf{b}_v$ and $\mathbf{b}_u$ are the centers of the bounding boxes of v and u, respectively, β is a weight factor set to 0.5 (found empirically), and δt is the sampling interval since the last stage.

Figure 8. Original image. Camera axis is not perpendicular to the street.

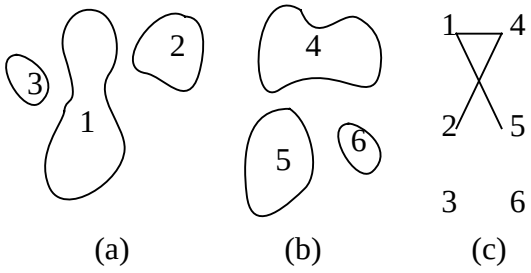


Figure 9. (a) Blobs in frame $(i-1)$. (b) Blobs in frame i. (c) Relationship among blobs

CHAPTER 5

MODELING VEHICLES

The input to this level is tracked blobs and the output is the spatio-temporal coordinates of each vehicle. The relationship between vehicles and blobs in the image is determined at this point. Each vehicle is associated with a list of blobs. Moreover, a blob can be shared by more than one vehicle. Vehicles are modeled as rectangular patches with a certain dynamic behavior. We found that for the purpose of tracking, this simple model adequately resembles the vehicle shape and motion dynamics. In our system we use the Extended Kalman Filter (EKF) in the tracking process. We now present this model in more detail and then describe how tracking is performed.

VEHICLE MODEL

Our vehicle model is based on the assumption that the scene has a flat ground. A vehicle is modeled as a rectangular patch whose dimensions depend on its location in the image. The dimensions are equal to the projection of the dimensions of an average size vehicle at the corresponding location in the scene. The patch is assumed to move with a constant velocity in the scene coordinate system. The patch acceleration is modeled as zero-mean, Gaussian noise to accommodate for changes in velocity. The discrete-time dynamic system for the vehicle model can be described by the following equation:

$$\mathbf{x}_{t+1} = \mathbf{F}\mathbf{x}_t + \mathbf{v}_t \tag{4}$$

where the subscripts indicate time, $\mathbf{x} = \begin{bmatrix} x & \dot{x} & y & \dot{y} \end{bmatrix}^T$ is the state vector consisting of the vehicle location, (x, y) and velocity, $(\dot{x}, \dot{y})$, $\mathbf{F}$ is the transition matrix of the system given by $\begin{bmatrix} 1 & \delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and $\mathbf{v}_t$ is a sequence of zero-mean, white, Gaussian process noise with covariance matrix $\mathbf{Q}$. We compute $\mathbf{Q}$ as in [1] (page 84) to become $\begin{bmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{A} \end{bmatrix} q$ where $\mathbf{A} = \begin{bmatrix} \frac{(\delta t)^3}{3} & \frac{(\delta t)^2}{2} \\ \frac{(\delta t)^2}{2} & \delta t \end{bmatrix}$ and represents the variance of the acceleration.

VEHICLE TRACKING

The next five subsections describe one tracking cycle.

Relating vehicles to blobs

We represent the relationship between vehicles and blobs as a directed bipartite graph, $GP_i(VP_i, EP_i)$, where $VP_i = B_i \cup P$. B_i is the set of blobs computed from the i th image as in the previous chapter. P is the set of vehicles. An edge (p,u), $p \in P$ and $u \in B_i$ denotes that blob u participates in vehicle p. We call GP_i a vehicle graph. Given a blob graph, $G_i(V_i, E_i)$, and a vehicle graph, GP_{i-1}, EP_i is computed as follows:

$$EP_i = \{(p,u) \mid (u,v) \in E_i \wedge (p,v) \in EP_{i-1}\}. \tag{5}$$

In other words, if a vehicle was related to a blob in frame $(i-1)$ and that blob is related to another blob in frame i (through a split, merge, etc.), then the vehicle is also related to the latter blob.

Prediction

Given the system equation as in the previous section, the prediction phase of the Kalman filter is given by the following equations:

$$\begin{aligned} \hat{\mathbf{x}}_{t+1} &= \mathbf{F}\mathbf{x}_t, \\ \hat{\mathbf{P}}_{t+1} &= \mathbf{F}\mathbf{P}_t\mathbf{F}^T + \mathbf{Q}. \end{aligned} \tag{6}$$

Here, $\hat{\mathbf{x}}$ and $\hat{\mathbf{P}}$ are the predicted state vector and state error covariance matrix, respectively. $\mathbf{x}$ and $\mathbf{P}$ are the previously estimated state vector and state error covariance matrix, respectively.

Calculating vehicle positions

In this step, we use the predicted vehicle locations as starting positions and we apply the following rule to update the locations of vehicle: Move each vehicle, p, as little as possible so that it covers as much as possible of its blobs, $\{u \mid (p,u) \in EP_i\}$; and if a number of vehicles share some blobs, they should all participate in covering all these blobs. The amount by which a vehicle covers a blob implies a measure of overlap area. We have already used the bounding box as a shape representation of blobs. However, since the blob bounding box area may be quite different from the actual blob area, we will include the blob density as computed in the previous chapter in the computation of the vehicle-blob overlap area. Let $BB(p)$ be the bounding box of a vehicle p, and $BB(b)$ be the bounding box of a blob, b. The intersection of $BB(p)$ and $BB(b)$ is a rectangle whose area is denoted by $X(BB(p), BB(b))$. The overlap area between p and b is computed as $X(BB(p), BB(b)) \times D(b)$. When multiple vehicles share a

blob, the overlap area is computed this way for each vehicle only if the other vehicles do not also overlap the intersection area. If they did, that particular overlap area is divided by the number of vehicles whose boxes overlap the area. Figure 10 illustrates this situation. The overlap area for p_1 is computed as $a \times D(b_1) + b \times D(b_2) + \frac{c \times D(b_2)}{2}$. For p_2, the overlap area is $d \times D(b_2) + \frac{c \times D(b_2)}{2}$.

The problem of finding the optimum locations of vehicles can be stated in terms of the overlap area measure that we just defined. We would like to place vehicles such that the total overlap area of each vehicle is maximized. We restate the optimization problem as the problem of finding the minimum total overlap area arrangement of vehicles which has the smallest distances between old and new locations of the vehicle. We do not attempt to solve the problem optimally because of its complexity. Instead, we resort to a heuristic solution using relaxation. First, a large step size is chosen. Then, each vehicle is moved in all possible directions by the step size, and the location which minimizes the overlap area is recorded. Vehicle locations are then updated according to the recorded locations. This completes one iteration. In each following iteration, the step size is decreased. In our implementation, we start with a step of 64 pixels and halve the step size in each iteration until a step size of 1 pixel is reached.

The resulting locations form the measurements that will be fed back into the EKF to produce the new state estimates. Moreover, we use the overlap area to provide feedback about the measurement confidence by setting the measurement error standard deviation, which is described below, to be inversely proportional to the ratio of the overlap area to the vehicle area. That is, the smaller the overlap area, the less reliable the measurement is considered to be.

Estimation

A measurement is a location in the image coordinate system as computed in the previous subsection, $\mathbf{z}$. Measurements are related to the state vector by

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \mathbf{w}_t \tag{7}$$

where $\mathbf{h}$ is a non-linear measurement function (the inverse perspective projection function) and $\mathbf{w}_t$ is a sequence of zero-mean, white, Gaussian measurement noise with covariance $\mathbf{R}_t$ given by $\begin{bmatrix} \sigma_t^2 & 0 \\ 0 & \sigma_t^2 \end{bmatrix}$. The measurement error standard deviation, σ_t, depends on the overlap area computed in the previous section. We let $\mathbf{H}$ be the Jacobian of $\mathbf{h}$. The EKF state estimation equations become

$$\begin{aligned} \mathbf{K}_{t+1} &= \hat{\mathbf{P}}_{t+1}\mathbf{H}^T\left(\mathbf{H}\hat{\mathbf{P}}_{t+1}\mathbf{H}^T + \mathbf{R}_t\right)^{-1}, \\ \mathbf{x}_{t+1} &= \hat{\mathbf{x}}_{t+1} + \mathbf{K}_{t+1}\left(\mathbf{z}_{t+1} - \mathbf{h}(\hat{\mathbf{x}}_{t+1})\right), \\ \mathbf{P}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})\hat{\mathbf{P}}_{t+1}. \end{aligned} \tag{8}$$

The estimated state vector $\mathbf{x}_{t+1}$ is the outcome of the vehicles level.

Refinement

At the end of this stage, we perform some checks to refine the vehicle-blob relationships since vehicles have been relocated. These can be summarized as follows:

a. If the overlap area between a vehicle and one of its blobs becomes less than 10% of the size of both, it will no longer be considered as belonging to this vehicle. This serves as a splitting procedure when one vehicle passes another.

b. If the overlap area between a vehicle and a blob that does not belong to any vehicles becomes more than 10% of the size of either one, the blob will be added to the vehicle

blobs. This makes the vehicle re-acquire some blobs that may have disappeared due to occlusion.

c. If a cluster of blobs is found which are not related to any vehicles and whose age is larger than a threshold value (i.e., they have been successfully tracked for a certain number of frames), we do the following: A new vehicle may be initialized only if it will be more than 30% covered by the blobs cluster. The vehicle is given an initial velocity equal to the average of the blobs velocities. This serves as the initialization step. The age requirement helps reduce the chances of unstable blobs being used to initialize vehicles.

d. Select one of the blobs which is already assigned one or more vehicles but can accommodate more vehicle patches. Create a new vehicle for this blob as in c. This handles cases in which a more than one vehicle appear in the scene while forming one big blob which does not split. If we do not do this step, only one vehicle would be assigned to this blob.

e. If a vehicle leaves the view of the camera or has not been covered by any blobs for an extended period of time, the vehicle is deleted.

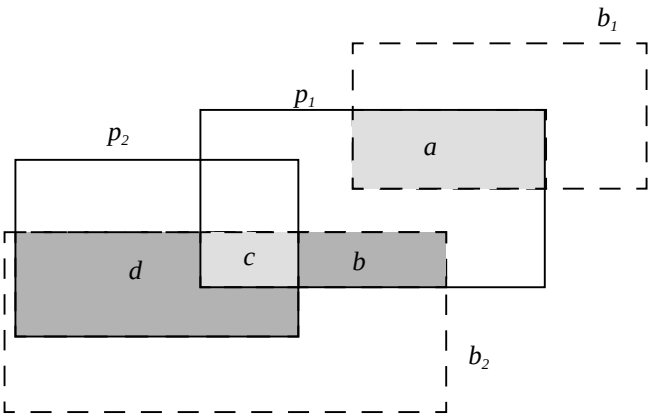


Figure 10. Overlap area. Vehicles p_1 and p_2 share blob b_2 while b_1 is only part of p_1

CHAPTER 6

RESULTS AND CONCLUSIONS

EXPERIMENTAL RESULTS

The system was implemented on a dual-Pentium 200MHz PC equipped with a C80 Matrox Genesis vision board. Tracking was achieved at 15 frames/second even for the most cluttered scenes. Several weaving sequences were used to test the system. Variations included different lighting conditions, camera view angles, and vehicle speeds (e.g., rush hour). The system was able to track most vehicles successfully (average accuracy 85%) and provided accurate trajectory information. Ground truth was established by using manual counters. The system dealt well with partial occlusions. The success is partly due to Kalman filtering since the measurement error standard deviation is set to reflect confidence in the found features. However, there were some failures especially with large trucks which, depending on the camera view, may completely occlude other vehicles. Figure 11 and Figure 12 show some snapshots of the tracking process performed on scenes with two different camera view angles and during different times of the day. One drawback of the current implementation is the use of a fixed size vehicle model for all vehicles. Even though this model works for most vehicles, it does not correctly track very large vehicles. Figure 13 shows how a large vehicle is tracked as two separate vehicles. Another drawback is apparent in the feature extraction component of the

system: large shadows are extracted as features and can confuse the tracking process. We are currently working on ways to deal with these two problems. An interesting probabilistic shadow handling approach is given in [6]. Wixson *et al*. [26] proposed using different parameters (or in some cases different algorithms) for different scene illumination conditions and provided an elegant condition assessment method.

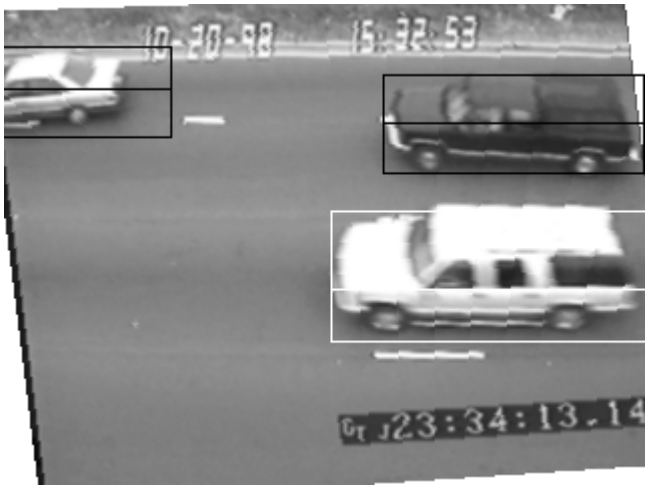
By providing the location of lane boundaries, our system was used to perform data collection for the Minnesota Department of Transportation. The data included the count and average speed of vehicles in each lane as well as vehicles that move between the two lanes farthest from the camera (weaving vehicles).

The data was provided cumulatively every 10, 30, 60, and 300 seconds. Sample data for the two lanes farthest from the camera are shown in Table 2. In the table, periods are measured in seconds and speed is measured in miles per hour. A visualization of vehicle trajectories accumulated over an hour of heavy traffic is shown in Figure 14. Notice the crisp vehicle paths for the two lanes closer to the camera. Also notice the gray area between the other two lanes which shows that a great deal of weaving took place. The trail in lane 4 appears shorter because the system starts the tracking a bit later than other lanes, due to the fact that vehicles in this lane move faster in the image.

CONCLUSIONS

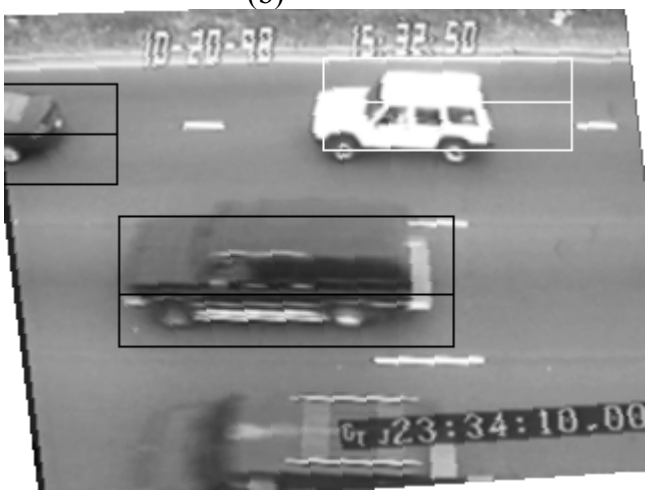
We presented a real-time model-based vehicle tracking system capable of working robustly under many difficult circumstances. The main goal of the system is to provide data on weaving sections. For each vehicle in the view of the camera, the system produces location and

velocity information as long as the vehicle is visible. There are some issues that still need to be addressed. Dealing with large shadows and large vehicles are two such issues.



(a)

(b)



(c)

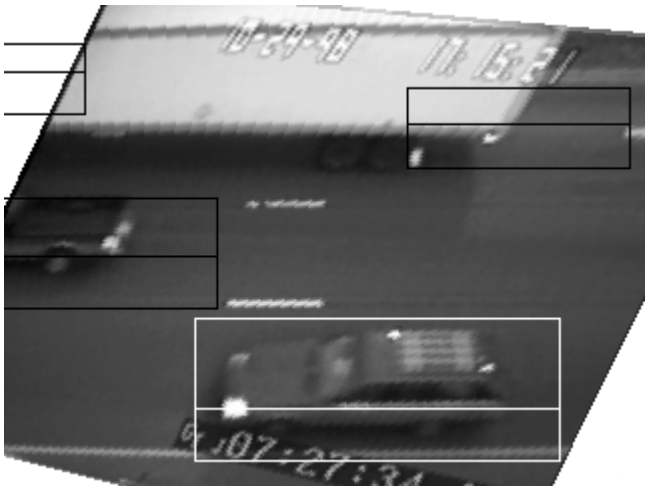
Figure 11. Three snapshots from a day sequence

(a)

(b)

Figure 12. Two snapshots from a night sequence

(a)



(b)

Figure 13. A large vehicle tracked as two vehicles

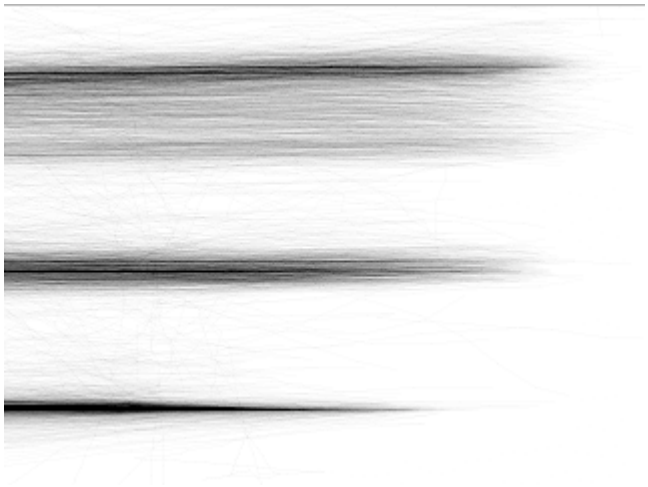


Figure 14. A visualization of vehicle trajectories on four lanes over a period of one hour

Time/ period	Vehicles no. (no. in period)	Lane 1->2 Spd (no.)	Lane 2->1 Spd (no.)	Lane 2->2 Spd (no.)
16:30:40 /10	1243 (15)	34.5 (3)	37.6 (2)	42.1 (3)
16:30:50 /10	1265 (22)	32.6 (1)	25.2 (3)	42.3 (4)
16:31:00 /10	1289 (23)	28.4 (4)	0 (0)	38.4 (5)
16:31:00 /30	1289 (60)	31.2 (8)	30.2 (5)	40.6 (12)
16:31:00 /60	1289 (110)	28.8 (18)	29.2 (7)	38.3 (23)

Table 2. Sample output

REFERENCES

[1] Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association*, Academic Press, 1988.

[2] K.D. Baker and G.D. Sullivan, Performance assessment of model-based tracking, in *Proc. of the IEEE Workshop on Applications of Computer Vision*, pp. 28—35, Palm Springs, CA, 1992.

[3] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik, A real-time computer vision system for measuring traffic parameters, in *Proc. of the IEEE Conf. Computer Vision and Pattern Recognition*, pp. 496—501, June 1997, Puerto Rico.

[4] R. Cucchiara, M. Piccardi, and P. Mello, Image analysis and rule-based reasoning for a traffic monitoring system, in *Proc. of the IEEE Conference on Intelligent Transportation Systems*, pp. 758—763, Tokyo, Japan, October 1999.

[5] D. Dailey and L. Li, Algorithm to estimate vehicle speed using un-calibrated cameras, in *Proc. of the IEEE Conference on Intelligent Transportation Systems*, pp. 441—446, Tokyo, Japan, October 1999.

[6] N. Friedman, S. Russell, Image segmentation in video sequences, in *Proc. of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, Providence, Rhode Island, 1997.

[7] S. Gil, R. Milanese, and T. Pun, Combining multiple motion estimates for vehicle tracking, in *Proc. of the Fourth European Conference on Computer Vision*, vol. 2, pp. 307—320, Cambridge, UK, April 1996.

[8] M. Haag and H.-H. Nagel, Incremental recognition of traffic situations from video image sequences, *Image and Vision Computing*, vol. 18, pp. 137—153, 2000.

[9] G. Halevi and D. Weinshall, Motion of disturbances: detection and tracking of multi-body non-rigid motion, in *Proc. of the IEEE Conf. Computer Vision and Pattern Recognition*, pp. 897—902, June 1997, Puerto Rico.

[10] K.P. Karmann and A. Brandt, Moving object recognition using an adaptive background memory, V. Cappellini, editor, *Time-Varying Image Processing and Moving Object Recognition*, Amsterdam, Netherlands, 1990.

[11] M. Kilger, A shadow handler in a video-based real-time traffic monitoring system, in *Proc. of the IEEE Workshop on Applications of Computer Vision*, pp. 1060—1066, Palm Springs, CA, 1992.

[12] D. Koller, J. Weber, and J. Malik, Robust multiple car tracking with occlusion reasoning, in ECCV, Stockholm, Sweden, 1994.

[13] H. Kollnig and H.-H. Nagel, Matching object models to segments from an optical flow field, in *Proc. of the Fourth European Conference on Computer Vision*, vol. 2, pp. 15—18, Cambridge, UK, April 1996.

[14] H. Kollnig and H.-H. Nagel, 3D pose estimation by directly matching polyhedral models to gray value gradients, *International Journal of Computer Vision*, vol. 23, no. 3, pp. 283—302, 1997.

[15] H. Leuck, and H.-H. Nagel, Automatic differentiation facilitates OF-integration into steering-angle-based road vehicle tracking, in *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 360—365, Fort Collins, CO, June 1999.

[16] S.J. Maybank, A.D. Worrall, and G.D. Sullivan, Filter for car tracking based on acceleration and steering angle, in *Proc. of the Seventh British Machine Vision Conference* (BMVC '96), vol. 2, pp. 615—624, Edinburgh, England, September 1996.

[17] O. Masoud and N. P. Papanikolopoulos, A robust real-time multi-level model-based pedestrian tracking system, in *Proc. of the ITS America Seventh Annual Meeting*, Washington, DC, June 1997.

[18] P.G. Michalopoulos, Vehicle detection video through image processing: The autoscope system, *IEEE Transactions on Vehicular Technology*, vol. 40, pp. 21—29, 1991.

[19] H.-H. Nagel, T. Schwarz, H. Leuck, and M. Haag, T3wT: tracking turning trucks with trailers, in *Proc. of the IEEE Workshop on Visual Surveillance*, pp. 65—72, Bombay, India, January 1998.

[20] S.M. Smith and J.M. Brady, ASSET-2: real-time motion segmentation and shape tracking, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(8):814—820, 1995.

[21] G.D. Sullivan, Model-based vision for traffic scenes using the ground-plane constraint, *Phil. Trans. Roy. Soc. (B)*, 337, pp. 361—370, 1992.

[22] G.D. Sullivan, K.D. Baker, A.D. Worrall, C.I. Attwood, and P.M. Remagnino, Model-based vehicle detection and classification using orthographic approximations, *Image & Vision Computing*, vol. 15, no. 8, pp. 649—654, Aug. 1997.

[23] G.D. Sullivan, A.D. Worrall, and J.M. Ferryman, Visual Object Recognition Using Deformable Models of Vehicles, in *Proc. Workshop on Context-Based Vision*, pp. 75—86, Cambridge Massachusetts, June 1995.

[24] T.N. Tan, G.D. Sullivan, and K.D. Baker, Model-based localization and recognition of road vehicles, *International Journal of Computer Vision*, vol. 27, no. 1, pp. 5—25, 1998.

[25] J. Versavel, Road safety through video detection, in *Proc. of the IEEE Conference on Intelligent Transportation Systems*, pp. 753—757, Tokyo, Japan, October 1999.

[26] L. Wixson, K. Hanna, and D. Mishra, Improved illumination assessment for vision-based traffic monitoring, in *Proc. of the IEEE Workshop of Visual Surveillance*, pp. 34—41, Bombay, India, January 1998.

[27] A.D. Worrall, G.D. Sullivan, and K.D. Baker, A simple, intuitive camera calibration tool for natural images, in *Proc. of the 5th British Machine Vision Conference*, pp. 781—790, 1994.

[28] M. Yamada, K. Ueda, I. Horiba, and N. Sugie, Discrimination of the road condition towards understanding of vehicle driving environments, in *Proc. of the IEEE Conference on Intelligent Transportation Systems*, pp. 20—24, Tokyo, Japan, October 1999.