Pseudo-realtime Activity Detection for Railroad Grade Crossing Safety

ZuWhan Kim
California PATH/Computer Science Division
University of California, Berkeley, CA, USA
zuwhan@cs.berkeley.edu

Theodore E. Cohn
School of Optometry and Dept. of Bioengineering
University of California, Berkeley, CA, USA
tecohn@spectacle.berkeley.edu

Abstract

It is important to understand the factors underlying grade crossing crashes, and to examine potential solutions. We have installed a camera in front of a locomotive to examine grade crossing accidents (or near accidents). We present a computer vision system that automatically extracts possible near accident scenes by detecting activity of vehicles crossing in front of the train after signals are ignited. We present a fast algorithm to detecting moving objects recorded by a moving camera with minimal computation. The moving object is detected by 1) estimating ego-motion of the camera, and 2) detecting and tracking feature points whose motion is inconsistent with the camera motion. We introduce a pseudo-realtime ego-motion (camera motion) estimation method with a robust optimization algorithm. We present experiments on ego-motion estimation and moving object detection. Our algorithm works in pseudo-realtime and we expect that our algorithm can be applied to realtime applications such as collision warning in the near future with the development of hardware technology.

1. Introduction

According to U.S. Department of Transportation, in year 2000, a total of 2,895 accidents occurred at railroad crossings causing 306 deaths, [1]. The property damage in such events is large, as is personal injury. Therefore, it is important to understand the factors underlying grade crossing crashes, and examine potential solutions. While there are databases that record collisions and the prevailing conditions, little systematic attention has been paid to the activity that is antecedent to a collision, persons or vehicles crossing the tracks in front an oncoming train.

Video data is an important source of examining the grade crossing accidents (or near accidents). We have worked with the California Department of Transportation to install three video cameras (one facing forward and the others facing sides) in front of a locomotive which is operated along the *San Joaquin Rail Corridor* (about 280 miles from Bakersfield to Emeryville). There are about 700 crossings on this route. The video data is recorded realtime in an MPEG format (320×240 resolution, 30 frames per sec) in the *data collection device* installed on the locomotive. The data collection machine is also equipped with the global positioning system (GPS). GPS information is collected in every second. When the train arrives at the Oakland AMTRAK facility, the recorded video data (about $4\sim8$ hours) and GPS information are transferred to the *server computer* via wireless communication.

While more than 6 hours of video data are obtained daily, the number of the useful video scenes is small. First, the average number of crossing accidents/incidents is only about 5 per million train miles. The number of activities that we are interested in may be larger, but the ratio of the useful information is still too small for manual examination. Next, much of the gathered data contain images obtained between images. One easy way of removing useless scenes is to use the GPS coordinates. We have a database which contains the GPS coordinates of most of the crossings, and we can thus retain only those scenes near the crossings. About 75% of the data can be removed in this way, but the volume of the data remains too large to manually examine. In this paper, we present research on the automated detection of the activity near the crossings.

An example video frame is shown in Figure 1. Our purpose is to develop a moving object detection algorithm where the camera is also in motion (mostly moving forward). The algorithm needs be fast enough to process a large volume of data daily (does not need to be realtime) and robust enough to detect most of the (near) accident events with a small number of false alarms. There has been no fast enough (pseudo-realtime) algorithm of detecting moving objects from a moving camera because of the following difficulties:

- it is hard to estimate the camera motion (ego-motion) robustly but to use an accurate optical flow which requires a large amount of computation, and

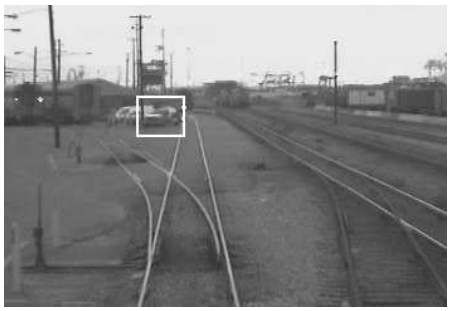


Figure 1. An example video frame. The vehicle inside of the rectangle is moving to the left.

- the 3-D structure of the background is also unknown. To robustly estimate both camera motion and the background structure, a large amount of computation involving a large number of frames is required (otherwise, it will results in noisy camera parameter estimation for some of the frames).

We present a pseudo-realtime algorithm which addresses the above difficulties. Our algorithm uses a relatively small number (100∼200) of corner features and their matches. The ego-motion is estimated from the corner matches from a small number of frames (two, in our example), and the corner points which moves inconsistently to the camera motion are detected. Both the corner matching and the camera motion estimation (from a small number of frames) may introduce noise. To handle such noise we use a *robust technique* to estimate the camera motion, and use temporal information to find only corners which moves *consistently inconsistent* to the camera motion. It removes the corners detected by false matches or wrong camera motion estimation.

In Section 2, an introduction on the ego-motion estimation is presented. We present our algorithm in Section 3 and show experiments in Section 4. Finally, the conclusion and the future work is shown in Section 5.

2. Background

The goal of the ego-motion estimation is to recover the motion of the camera using image measurements of fixed points in the scene. When the 3-D coordinates (or constraints among them such as planarity) of the feature points are known (for example, [9] or camera calibration problems), another coordinate system (world coordinate) is often introduced and the rotations and the translations of each and every frames are estimated according to the world coordinate. However, in this application, we do not have any information on the 3-D coordinates of the feature points. Therefore, we only estimate the relative configuration (rotation and translation) of the camera with respect to the previous one. For each frame, we recover the rotation and the translation of the camera with respect to the previous frame.

When the number of frames is small enough, it is tractable to estimate camera configurations of all the frames as well as the 3-D coordinates of the feature points at the same time, [8], [4]. However, such a calculation deals with a matrix of substantial dimension, including all the camera parameters and the image and world coordinates of the feature points. Therefore, it is not efficient to process hours of video data with this approach. Hence, we limit the scope of this paper to the two frame case described above.

In this section, we introduce a brief survey on the ego-motion estimation from two image frames. Most of the contents in this section are also found in the photogrammetry literature (e.g. [5]).

2.1. Projective Geometry

In the basic pinhole camera model, a 3-D point $\mathbf{X} = (X\ Y\ Z)^\top$ on a *camera coordinate* is mapped to the point on the image plane (*image coordinate*) $\mathbf{u} = (u\ v)^\top$, where $u = fX/Z$, $v = fY/Z$, and f is the focal length. In a matrix form,

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \mathbf{K} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

For simplicity, we redefine $\mathbf{u} \equiv (wu\ wv\ w)^\top$. For a finite projective camera, the internal camera parameters,

$$\mathbf{K} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix},$$

where α_u and α_v represent the focal length of the camera in terms of pixel dimensions in the u and v direction respectively, and $(u_0\ v_0)^\top$ is the coordinate of the image center.

Usually, a separate coordinate system, *world coordinate*, which is different from the camera coordinate is introduced. In this case, the Euclidean transformation (rotation, $\mathbf{R}$, and translation, $\mathbf{T}$) between the world and camera coordinate is introduced: $\mathbf{u} = \mathbf{K}(\mathbf{R}\mathbf{X} + \mathbf{T})$.

Using the above equation, $\mathbf{K}$, $\mathbf{R}$, and $\mathbf{T}$ can be estimated when we know the correspondences between points in the world and image coordinates (see [5] for details).

2.2. Two View Camera Geometry

Consider two cameras (or two camera configurations) $\mathbf{C}$ and $\mathbf{C}'$. The *baseline* is defined as the line segment joining two camera centers. For each camera, the *epipole* is defined as the point of intersection of the baseline and the image plane. Consider a 3-D point $\mathbf{X}$ and its projection, $\mathbf{u}$, on $\mathbf{C}$.

$\mathbf{X}$, $\mathbf{u}$, and the baseline lies on the same plane, which is the *epipolar plane*. The *epipolar line* is defined as the intersection of the epipolar plane with the image plane. For any image point $\mathbf{u}$ of $\mathbf{C}$, its correspondence of $\mathbf{C}'$, $\mathbf{u}'$, also lies on the same epipolar plane (and the corresponding epipolar line) regardless the 3-D position of $\mathbf{X}$. Furthermore, for any corresponding points $\mathbf{u}$ and $\mathbf{u}'$,

$$\mathbf{u'}^{\top}\mathbf{F}\mathbf{u} = 0, \tag{1}$$

where $\mathbf{F}$ is the *fundamental matrix*. With the fundamental matrix, we can get epipolar lines. For any point $\mathbf{u}$, the corresponding epipolar line is

$$\mathbf{l}' = \mathbf{F}\mathbf{u}. \tag{2}$$

In our application, $\mathbf{C}$ and $\mathbf{C}'$ are the same camera with different position and orientation. Regard $\mathbf{C}$ be the camera configuration of the previous frame and $\mathbf{C}'$ of the current frame. For simplicity, we let the the camera coordinate of $\mathbf{C}$ be the world coordinate. Then the configuration of $\mathbf{C}'$ is represented by a rotation, $\mathbf{R}$, and a translation, $\mathbf{T} = (T_X\ T_Y\ T_Z)^{\top}$.

We introduce a new coordinate system called *normalized coordinates*. A normalized coordinate is obtained from the corresponding image coordinate given the internal camera parameters, $\mathbf{K}$: $\mathbf{x} = (x\ y\ 1)^{\top} = \mathbf{K}^{-1}(u\ v\ 1)^{\top}$.

For $\mathbf{C}$, a point $\mathbf{X} = (X\ Y\ Z)^{\top}$ on the world coordinate can be easily transformed to a point $\mathbf{x} = (x\ y\ 1)^{\top}$ on the normalized coordinate:

$$x = X/Z, \text{ and } y = Y/Z. \tag{3}$$

For $\mathbf{C}'$, the transformed point

$$\mathbf{x}' = (x'\ y'\ 1)^{\top} = (X'/Z'\ Y'/Z'1)^{\top}, \tag{4}$$

where $\mathbf{X}' = (X'\ Y'\ Z')^{\top} = \mathbf{R}\mathbf{X} + \mathbf{T}$.

As the counterpart of Equation 1, for any two corresponding points $\mathbf{x}$ and $\mathbf{x}'$,

$$\mathbf{x'}^{\top}\mathbf{E}\mathbf{x} = 0, \tag{5}$$

where $\mathbf{E}$ is the *essential matrix*. In fact,

$$\mathbf{E} = [\mathbf{T}]_{\times}\mathbf{R}, \tag{6}$$

where $[\mathbf{T}]_{\times}$ is the *skew-symmetric matrix* of $\mathbf{T}$. See [5] for details.

2.3. Ego-Motion Estimation

Assume the internal parameters, $\mathbf{K}$, are known (they can be obtained from a separate calibration procedure). The ego-motion estimation is to recover $\mathbf{R}$ and $\mathbf{T}$ from a set of point correspondences, where their relationship is non-linear.

One approach is to first estimate the essential matrix, $\mathbf{E}$, and calculate $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ (the estimates of $\mathbf{R}$ and $\mathbf{T}$) from it. From a point correspondence, we get two linear equations (with respect to the parameters of $\mathbf{E}$) by applying it to Equation 5. Given a number of point correspondences, we can easily get $\hat{\mathbf{E}}$ by, for example, using the least square estimation. Once $\hat{\mathbf{E}}$ is obtained, we get $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ by applying the singular value decomposition. See [5] for details.

However, this method does not provide accurate estimation, [7], because the parameters of $\mathbf{E}$ are not directly related to the physical properties, and minimizing their errors assuming for example, Gaussian error distributions (the least square estimation) is not meaningful. We want to minimize meaningful errors, for example, the reprojection error:

$$Err_{\text{rprj}}(\mathbf{x}, \mathbf{x}', \hat{\mathbf{X}}) = \sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2, \tag{7}$$

where ($\mathbf{x}_i$, $\mathbf{x}'_i$) is the measured position of the i-th correspondence ($\mathbf{x} = \mathbf{x}_{1,\ldots,n}$ and $\mathbf{x}' = \mathbf{x}_{1,\ldots,n}$, where n is the number of points), $\hat{\mathbf{X}}$ is the estimate of the "true" 3-D positions of the points, $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$ are the projection of $\hat{\mathbf{X}}_i$ on $\mathbf{C}$ and $\mathbf{C}'$, and $d()$ is the Euclidean distance. We can estimate $\mathbf{R}$ and $\mathbf{T}$ by solving non-linear equations of Equation 3, Equation 4, and Equation 7, on the parameters of $\hat{\mathbf{R}}$, $\hat{\mathbf{T}}$, and $\hat{\mathbf{X}}_i$. We can use any iterative method, and the result of the first approach (which is using the least square estimate of the essential matrix) is usually used as a initial guess. The initial values of $\hat{\mathbf{X}}_i$ can be obtained from the initial values of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$. From Equation 4, we get

$$\begin{pmatrix} x'_i\hat{Z}'_i \\ y'_i\hat{Z}'_i \\ \hat{Z}'_i \end{pmatrix} = \hat{\mathbf{R}} \begin{pmatrix} x_i\hat{Z}_i \\ y_i\hat{Z}_i \\ \hat{Z}_i \end{pmatrix} + \hat{\mathbf{T}}.$$

We get two linear equations on $\hat{Z}'_i$ from the above equation, and can estimate $\mathbf{X}_i$ from it.

Note that $\mathbf{R}$ is a 3×3 matrix, but has only 3 DOF. Therefore, we have 6 more constraints to make $\mathbf{R}$ a valid rotation matrix. [7] uses a unit quaternion to represent the rotation to effectively enforce these constraints.

3. Moving Object Detection

The flowchart of the moving object detection algorithm is shown in Figure 2. For each frame, we apply an eigenvalue-based corner detector [3]. When a corner is detected, a small 9×9 template of the grey level image is extracted. Then we search for the matches of the extracted templates in the next frame. The match score is based on the correlation and the search is performed on a 9×9 search window.

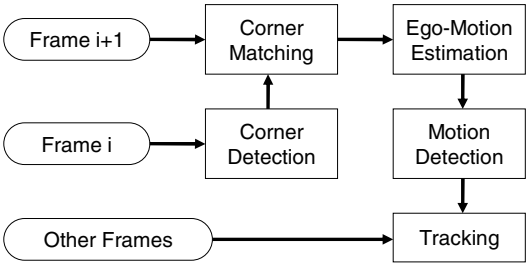


Figure 2. The flowchart of the moving object detection algorithm.

Figure 3. Detected corner features (square boxes) and their optical flows. Note that the line features can cause false matches (aperture problem). The optical flows were exaggerated by 3 times for the visibility.

Figure 3 shows detected corner features and their matches (shown as optical flows). Note that there are many line features. The matching process can suffer from the *aperture problem* and we may have some false matches. Now, our goal is to find the corners of which the movements are not consistent with others. This is done by estimating the ego-motion of the camera and checking the consistency of the movement. Finally, we track detected moving corner features for several more frames to remove the noise (falsely detected motion).

An effort on a similar problem is found in [10]. It introduces the *multibody fundamental matrix* (a set of fundamental matrices) to segment motions into several groups. In fact, many of the moving objects (corner features) can be detected without fully recovering the ego-motion but only by recovering the fundamental matrix (or the essential matrix). When we know the fundamental matrix, we can calculate the corresponding epipolar line of a point (Equation 2), and we can detect a moving object as long as its motion does not accidentally lie on the corresponding epipolar line. However, this "accidental alignment" of the motion occurs often in our application. An example is shown in Figure 4. Most of the object motions that we are interested in occur in

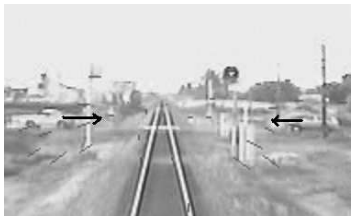


Figure 4. We need to recover the ego-motion fully because the "accidental alignment" of the object motion and the corresponding epipolar line occurs frequently.

a horizontal direction in the middle of the image, while the epipolar lines in the middle of the images are also mostly horizontal.

In fact, this is an intrinsic ambiguity which cannot be removed unless we know the 3-D coordinate (or the depth) of the point. However, this ambiguity can be addressed if we constrain the 3-D coordinates of the objects. For example, we no visible object can have negative Z-coordinate, nor (usually) stays in front of a moving train closer than a certain distance (say 10 meter). To apply these constraints, we need to recover the 3-D coordinates of the points. Therefore, we recover the full camera parameters, $\mathbf{R}$ and $\mathbf{T}$.

3.1. Ego-Motion Estimation for Railroad Crossing Imagery

In our application, the ego-motion is mostly a forward translation (T_Z) with a very small amount of other translation (T_X, T_Y), and rotation. However, the ego-motion along the Z axis is difficult to recover robustly, [2]. For example, the pitch (rotation about the Y axis) and T_X causes similar optical flows (*bas-relief ambiguity*). Therefore, standard ego-motion estimation methods (Section 2.3) do not give good estimation, and they result in many false alarms in motion detection. In this section, we introduce an augmentation for the present application.

Error Model

The reprojection error (Equation 7) was introduced based on the assumption of a Gaussian error distribution of the reprojected points. This assumption is valid when the scene is static and all the matches (correspondences) are correct. However, in our application, we also need to consider the error caused by false matches and by object motion. If we consider these outliers, the error distribution is no longer Gaussian but close to a binomial or multinomial distribution. This error can be handled in the optimization procedure by applying *robust estimation* techniques, such as least-trimmed-square. To simulate the robust estimation techniques efficiently, we change the objective function

such that it minimizes the number of matches which do not fit to the current camera model:

$$Err_{\text{sigmoid}} = \sum_i e(\mathbf{x}_i, \mathbf{x}'_i, \hat{\mathbf{X}}_i), \tag{8}$$

where

$$e(\mathbf{x}_i, \mathbf{x}'_i, \hat{\mathbf{X}}_i) = \begin{cases} 1, \text{if } \hat{Z} < 0 \text{ or } \hat{\mathbf{X}} \text{ is right in front of the camera, or} \\ \text{sigmoid}(d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2; \mu), \text{otherwise,} \end{cases}$$

and μ is the mean of the sigmoid function. We allow maximum 1.0 reprojection error. Therefore $\mu = 1.0$.

In many cases, optimizing only with Err_{sigmoid} does not give good result because numerically $\text{sigmoid}(x) = \text{sigmoid}(y) = 1.0$ for any large numbers of x and y. For many point correspondences, the reprojection errors of the initial estimate are large enough to cause this numerical error. Therefore, we propose two step optimization. For the first, say, 15, iterations, we apply the reprojection error, then apply Err_{sigmoid} for the rest of the iterations.

Optimization

We use an iterative optimization method (e.g. the gradient descent method) to minimize the objective function. Usually, the initial estimates of the camera parameters are obtained from the least-square-estimate of the essential matrix, $\hat{\mathbf{E}}$ (see Section 2.3). However, in our case, involving forward ego-motion and outliers, the estimates of the rotation parameters, $\hat{\mathbf{R}}$, are highly unstable when obtained in that way. Therefore, we assume pure translation at the initial estimation. When there is no rotation $\mathbf{E} = [\mathbf{T}]_\times$ from Equation 6. In this instance, we can use the initial estimates of $\hat{T}_X = (\hat{E}_{3,2} - \hat{E}_{2,3})/2$, $\hat{T}_Y = (\hat{E}_{1,3} - \hat{E}_{3,1})/2$, and $\hat{T}_Z = (\hat{E}_{2,1} - \hat{E}_{1,2})/2$.

The next step is to apply the iterative optimization on $\hat{\mathbf{R}}$, $\hat{\mathbf{T}}$, and $\hat{\mathbf{X}}_i$. We apply a gradient-based optimization algorithm. However, the optimization process can be extremely time consuming because the number of parameters to optimize is $3n + 6$, where n is the number of the point matches. Also the initial estimation error of $\hat{\mathbf{X}}_i$ can be very large because its estimation is very sensitive to $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$. For example, if we do not have a correct estimate on the *point of expansion*, we will get many $\hat{Z}_i$'s with negative values.

Therefore, we optimize $\hat{\mathbf{X}}_i$ and the camera parameters ($\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$) separately. For each iteration step, we can first optimize (update) $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ given the point matches ($\mathbf{x}_i$ and $\mathbf{x}'_i$) and $\hat{\mathbf{X}}_i$'s, then optimize $\hat{\mathbf{X}}_i$'s given $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ (triangulation). In fact, the parameters of $\hat{\mathbf{X}}_i$ ($\hat{X}_i$, $\hat{Y}_i$, and $\hat{Z}_i$) are highly correlated among each other since $\hat{X}_i = \hat{x}_i \hat{Z}_i$, and $\hat{Y}_i = \hat{y}_i \hat{Z}_i$ (note that we need to estimate $\hat{X}_i$, $\hat{Y}_i$, and $\hat{Z}_i$ separately, mainly because x_i and y_i are not the real values but just observations which contain the sampling errors). However, when we assume that $\hat{Z}_i$ is the only independent variable and $\hat{X}_i = x_i \hat{Z}_i$ and $\hat{Y}_i = y_i \hat{Z}_i$ (*i.e.* when we assume $\mathbf{x}_i$ is correct), we get a closed-form solution of $\hat{Z}_i$ optimizing the square sum of the distance error, given $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$:

$$\arg_{\hat{Z}_i} \min\{(\hat{x}_i - x'_i)^2 + (\hat{y}_i - y'_i)^2\} = \frac{(r_{x_i}\hat{T}_Z - \hat{T}_X)(x'_i\hat{T}_Z - \hat{T}_X) + (r_{y_i}\hat{T}_Z - \hat{T}_Y)(y'_i\hat{T}_Z - \hat{T}_Y)}{(r_{x_i}\hat{T}_Z - \hat{T}_X)(r_{x_i} - x'_i) + (r_{y_i}\hat{T}_Z - \hat{T}_Y)(r_{y_i} - y'_i)}, \tag{9}$$

where $r_{x_i} = \hat{R}_{1,1}x_i + \hat{R}_{1,2}y_i + \hat{R}_{1,3}$ and $r_{y_i} = \hat{R}_{2,1}x_i + \hat{R}_{2,2}y_i + \hat{R}_{2,3}$. See Appendix for the proof. In fact, the objective function we minimize for $\hat{\mathbf{X}}$ (Equation 9) is different from the objective function (Equation 8). However, this approximation is good enough since our goal is not to accurately estimate $\mathbf{X}$ but just to find inconsistent motion. In [6] Hartley and Sturm introduced optimal triangulation method minimizing the reprojection error. However, we did not apply this method because the optimization process is complex and time-consuming.

In this application, we separate the iterative update procedure of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ because the convergence characteristics for $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ are very different from each other. Assume $Err()$ be the objective function to be minimized. For each iteration, we apply the following optimization steps:

Step 1: estimate $\frac{\partial Err}{\partial \hat{\mathbf{R}}}$, $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \alpha \frac{\partial Err}{\partial \hat{\mathbf{R}}}$,

Step 2: optimize $\hat{\mathbf{X}}_i$'s given $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$,

Step 3: estimate $\frac{\partial Err}{\partial \hat{\mathbf{T}}}$, $\hat{\mathbf{T}} \leftarrow \hat{\mathbf{T}} - \beta \frac{\partial Err}{\partial \hat{\mathbf{T}}}$, and

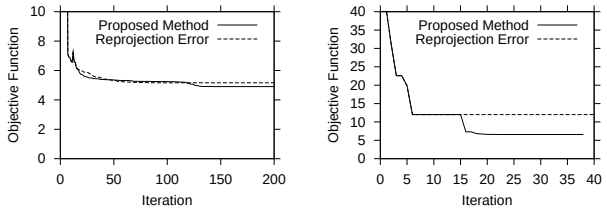
Step 4: optimize $\hat{\mathbf{X}}_i$'s given $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$,

where α and β controls the convergence speeds as in Levenberg-Marquardt iteration. **Step 2** is necessary because, when we update $\hat{\mathbf{T}}$ in **Step 3**, we use the newly updated $\hat{\mathbf{R}}$ of **Step 1**. However, when we skip **Step 2**, $\hat{\mathbf{R}}$ and $\hat{\mathbf{X}}_i$ will be inconsistent in **Step 3**. See Section 4, for the comparison between with and without separating the update of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$.

3.2. Motion Detection

Once the ego-motion is estimated, we detect object motion by finding inconsistent corner motions. We use $\hat{\mathbf{X}}_i$ (obtained from the ego-motion estimation) to detect inconsistent motion. The i-th point match is inconsistent with the estimated ego-motion when $e(\mathbf{x}_i, \mathbf{x}'_i, \hat{\mathbf{X}}_i) > \text{sigmoid}(1.0; \mu)$ (cf. Equation 8).

The inconsistency of a point motion can come from 3 possible reasons: 1) object motion, 2) matching error, or 3) ego-motion estimation error. Note that the inconsistency caused by the object motion is distinguished from the others because it is consistent over the frames. Therefore we applied a simple tracking algorithm to check this "consistent



(a) (b)

Figure 5. Comparison between the proposed optimization method and the reprojection error method, when (a) there is no visible false match; and (b) several false matches were found.

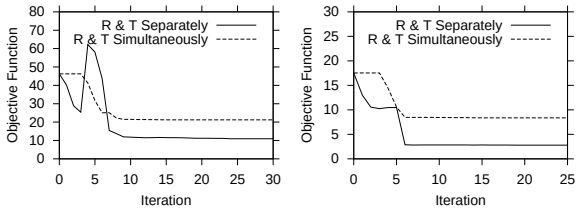


Figure 6. We get better estimation when we update $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ separately (see Section 3.1).

inconsistency." Once a corner is determined to be inconsistent, its matches are found for the next 7 or more frames, and we report it as a moving object when its motion is consistently inconsistent to the ego-motion.

4. Experiments

Currently, only a small volume of video data is available for a preliminary test. We focus our experiment on the ego-motion estimation.

We first compare the proposed optimization method and the reprojection error method. Figure 5 shows the convergence properties of two example pairs of frames. When there was a small number of false matches (for example, no visible false match was found in the frame pair for Figure 5a), the difference between two methods was small. However, when there were false matches (for example, at least 3 visible false matches were found in the frame pair of Figure 5b), the proposed method showed much better performance (6.58 *vs* 11.98).

The comparison is based on the objective function of Equation 8. However, even when we compared the reprojection error (Equation 7), the proposed methods showed superior performance when there were false matches (for example, 5.90 *vs* 21.27 for Figure 5b, both resulted in two outliers). At the same time, the differences were not significant when there was no false match (for example, 7.34 *vs* 7.31 for Figure 5a). In fact, the proposed method showed

Figure 7. A detected moving object with the trajectories of its corner features.

even better performances on many examples with no false matches. From this result, we find that applying a robust estimation method reduces the risk of falling into a local minimum caused by false matches.

We also performed an experiment to justify the separate update of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$. The comparison result is shown in Figure 6. We find that the simultaneous update of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$ caused poor performance because of the different convergence characteristics of $\hat{\mathbf{R}}$ and $\hat{\mathbf{T}}$.

Finally, the moving object detection result is shown in Figure 7. Five corners of the moving object were detected without any false alarm. The amount of computation depended on the number of the corners processed and the convergence characteristics on the motion. The experiment was performed on Pentium III (1GHz), and for most cases, about 50$\sim$150ms was spent for the corner detection and matching and 20$\sim$500ms (but mostly under 150ms) was spent for the optimization.

5. Conclusion and Future Work

We presented a moving object detection algorithm for the railroad crossing safety. We proposed a fast triangulation method and a robust optimization algorithm to meet our application's need. We presented preliminary experiments which showed that the proposed approach is promising. More thorough evaluation will be performed when a large volume of video data is collected. We will generate a learning data set by showing video clips to human, and compare the detection result with our algorithm.

Our algorithm works at pseudo-realtime (about 3$\sim$4 frames/sec on Pentium III). We expect that in near future (with the development of computer hardware) it can be applied to realtime applications such as an in-vehicle collision warning system.

Appendix: Proof of Equation 9

We define the object function,

$$Err(x', y', x, y, \hat{Z}) = \{(\hat{x'}(x, y, \hat{Z}) - x')^2 + (\hat{y'}(x, y, \hat{Z}) - y')^2\},$$

to be minimized w.r.t. $\hat{Z}$. From Equations 3 and 4,

$$\hat{x'}(x, y, \hat{Z}) = \frac{\hat{X}'(x, y, \hat{Z})}{\hat{Z}'(x, y, \hat{Z})} = \frac{\hat{Z}r_x + \hat{T}_X}{\hat{Z}r_z + \hat{T}_Z},$$

and

$$\hat{y'}(x, y, \hat{Z}) = \frac{\hat{Y}'(x, y, \hat{Z})}{\hat{Z}'(x, y, \hat{Z})} = \frac{\hat{Z}r_y + \hat{T}_Y}{\hat{Z}r_z + \hat{T}_Z},$$

where $r_x = \hat{R}_{1,1}x + \hat{R}_{1,2}y + \hat{R}_{1,3}$, $r_y = \hat{R}_{2,1}x + \hat{R}_{2,2}y + \hat{R}_{2,3}$, and $r_z = \hat{R}_{3,1}x + \hat{R}_{3,2}y + \hat{R}_{3,3}$.

We get $\hat{Z}$ that minimizes $Err()$ by taking its partial derivative w.r.t. $\hat{Z}$:

$$\frac{\partial Err}{\partial \hat{Z}} = \frac{-2}{(\hat{Z}r_z + \hat{T}_Z)^3}[(r_x\hat{T}_Z - \hat{T}_X)\{(x' - r_x)\hat{Z} + (x'\hat{T}_Z - \hat{T}_X)\}$$
$$+ (r_y\hat{T}_Z - \hat{T}_Y)\{(y' - r_y)\hat{Z} + (y'\hat{T}_Z - \hat{T}_Y)\}] = 0.$$
$$\hat{Z} = \frac{(r_x\hat{T}_Z - \hat{T}_X)(x'\hat{T}_Z - \hat{T}_X) + (r_y\hat{T}_Z - \hat{T}_Y)(y'\hat{T}_Z - \hat{T}_Y)}{(r_x\hat{T}_Z - \hat{T}_X)(r_x - x') + (r_y\hat{T}_Z - \hat{T}_Y)(r_y - y')}.$$

This is the only extremum of $Err()$ w.r.t $\hat{Z}$, which is the global minimum.

Acknowledgment

We would like to acknowledge René Vidal for his helpful comments on the optimization methods. This work was supported by CalTrans/PATH (RTA 65A0089).

References

[1] Railroad safety statistics annual report. *Office of Safety Analysis, Federal Railroad Administration, U.S. Department of Transportation*. Available at http://safetydata.fra.dot.gov/offi ceofsafety/.

[2] K. Daniilidis and H.-H. Nagel. Analytical results on error sensitivity of motion estimation from two views. *Image and Vision Computing*, 8:297–303, 1990.

[3] W. Forstner and E. Gulch. A fast operator for detection and precise location of distinct points, corners, and centers of circular features. In *Proc. Intercommision Conf. on Fast Processing of Photogrammetric Data*, pages 281–305, 1987.

[4] M. Han and T. Kanade. Creating 3d models with uncalibrated cameras. In *Proc. Fifth IEEE Workshop on Applications of Computer Vision*, pages 178–185, 2000.

[5] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[6] R. I. Hartley and P. Sturm. Triangulation. In *Proc. Conference on Computer Analysis of Images and Patterns*, 1995.

[7] B. K. Horn. Projective geometry considered harmful. 1999. Available at http://www.ai.mit.edu/people/bkph/.

[8] J. Oliensis. A multi-frame structure-from-motion algorithm under perspective projection. *International Journal of Computer Vision*, 34(2):163–192, 1999.

[9] O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry. Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control. *Asian Journal of Control*, 1(3):128–145, 1999.

[10] R. Vidal, S. Soatto, Y. Ma, and S. Sastry. Segmentation of dynamic scenes from the multibody fundamental matrix. In *Proc. ECCV Workshop on Vision and Modeling of Dynamic Scenes*, 2002.