Tracking and Reconstruction of Vehicles for Accurate Position Estimation

Källén, Hanna; Ardö, Håkan; Enqvist, Olof

Published in:
Applications in Computer Vision (WACV), 2011 Workshop on

DOI:
10.1109/WACV.2011.5711491

2011

Tracking and Reconstruction of Vehicles for Accurate Position Estimation

Hanna Källén Håkan Ardö Olof Enqvist
Centre for Mathematical Sciences, Lund University, Sweden
www.maths.lth.se/vision

Abstract

To improve traffic safety it is important to evaluate the safety of roads and intersections. Today this requires a large amount of manual labor so an automated system using cameras would be very beneficial. We focus on the geometric part of the problem, that is, how to get accurate three-dimensional data from images of a road or an intersection. This is essential in order to correctly identify different events and incidents, for example to estimate when two cars gets dangerously close to each other.

The proposed method uses a standard tracker to find corresponding points between frames. Then a RANSAC-type algorithm detects points that are likely to belong to the same vehicle. To fully exploit the fact that vehicles rotate and translate only in the ground plane, the structure from motion is estimated using an optimization approach based on the L_∞-norm. The same approach also allows for easy setup of the system by estimating the camera orientation relative to the ground plane. Promising results for real-world data are presented.

1. Introduction

To reduce the number of road traffic injuries it is important to know how safe certain roads and intersections are. There are different ways to evaluate this. The classic method is to count the number of accidents that occurs. Since accidents are rare it can take years to get a good assessment of safety this way. A faster approach is to predict the numbers of accidents that will happen by manually observing certain events, conflicts, during a much shorter time period [5]. This is done by letting trained personnel study video of the intersection. Naturally, this is very expensive and time-consuming and an automated method would be very beneficial.

In [8] a method for automated surveillance was proposed. To calculate the position of a vehicle, the 2D image of that vehicle was projected onto the road plane. If the camera can be placed right above the intersection, this will work fairly well. In most cases though, this is not possible. The projection of the vehicle gets stretched out and the estimated position incorrect. A better way to estimate the position would be to make a three-dimensional representation of the object and use this to calculate the position of the vehicle. This is the approach considered in this paper.

There is much work in the vision literature regarding traffic scenes. In [9] a system for tracking pedestrians and cars was presented which is based on object detectors. A system for making 3D shape reconstruction for traffic surveillance with multiple cameras is presented in [10]. To do this predefined 3D models of cars are used. In [2] vehicles was tracked by tracking feature points on it. After the features exit the tracking region, they are grouped into discrete vehicles using a motion constraint. In [13] a system for automatic calibration of a camera from traffic scenes was proposed. If the height of the camera is known, both intrinsic and extrinsic parameters can be found. A method to rectify images is given in [1]. By tracking the motion of two vehicles moving in constant speed an estimation of the ground plane can be done.

The approach described in this paper differs from most methods in at least one important aspect. Inspired by recent research in optimal methods for computer vision, the reprojection errors are minimized with respect to the L_∞-norm rather than the more common L_2-norm. This makes it possible to find the global optimum and it also makes it easier to impose extra constraints, such as the fact that vehicles are only moving in the ground plane.

1.1. Overview

We start with captured video from the intersection. For the reconstruction we need to find corresponding points between different frames. This is achieved in the following way. Every few frames we pick an image to use as a starting image. In this image a corner detector is used to find strong corner points. These points are then tracked a few frames using a KLT tracker [12], to get corresponding image points in a later frame.

To reduce the amount of outliers in the 3D reconstruction, we need to detect which points belong to the same vehicle. For this purpose we perform a motion segmenta-

tion algorithm that will be described in detail in Section 3.1. This algorithm finds all vechicles that are visible in the starting frame. The vehicles are then tracked both forwards and backwards in time until the vehicles leaves the camera field of view or until the tracking fails.

This procedure yields long tracks of corresponding image points likely to belong to the same vehicle. From these tracks we use a few views to perform our 3D reconstruction, which is described thoroughly in Section 3.2. The scale is fixed by assuming that some reconstructed points lie in the ground plane. Having a 3D model we use this to estimate the pose of the vehicle in all views, see Section 3.3. This gives us accurate information on the movement of the vehicle throughout the sequence. Section 4 described how the same framework can be used to estimate the camera rotation with respect to the ground plane. The idea is to do this once when the system is set up.

System Overview

1. *Track points between frames.*
2. *Cluster points from the same vehicle.*
3. *Reconstruct the vehicle using a few views.*
4. *Compute the vehicle pose in all views.*

2. Preliminaries

We assume that the ground is planar and that the vehicles are rigid bodies that only moves in that plane. We choose coordinate system, such that the z-axis is perpendicular to the ground. We also assume that the camera has known internal parameters. In reality we have one stationary camera and moving vehicles, but from the vehicles point of view it look like the camera is moving. Choosing this point of view we can assume stationary vehicles and multiple cameras.

2.1. Structure from Motion with Known Rotations

If the rotation between the different cameras are known we can solve the structure from motion problem with multiple cameras optimally with respect to the L_∞-norm, using second-order cone programming [6]. This gives us the translation of the cameras and the position of the 3D points. An advantage of this approach is that several views are used to estimate the 3D points, unlike the standard approach that gets initial estimates from only two views.

Due to noise there is no exact solution to the reconstruction problem. Thus we allow the reprojected points to deviate somewhat from the measured image points. To get linear constraints, we allow the reprojected point within a small square around the measured image point. This differs from the approach in [6]. The simplification allows us to solve the optimization using linear programming rather than second-order cone programming.

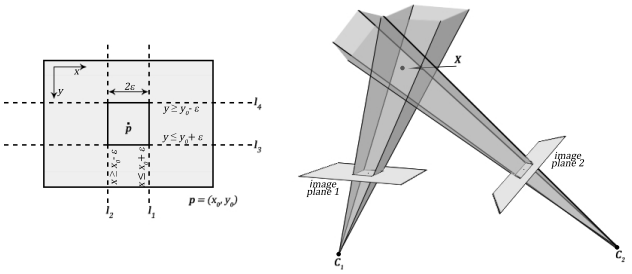

Figure 1. Linear constraints in the image plane and in space. The left figure shows the lines that define the box in which the reprojected point must lie inside. The right figure shows constraints in space from two cameras, the 3D point must lie inside both cones.

The square has side length 2ϵ, where ϵ is the tolerated error, and is defined by four lines in the image plane, see Figure 1. These lines are the intersections between the image plane and planes going through both the camera center and the image plane. The planes through the camera center form a generalized cone in which the 3D point has to be. A bigger value on ϵ results in a wider cone. For every camera, we get another cone constraint on the position of the 3D point. The constraints from two views can be seen to the right in Figure 1.

If we express the 3D point in cartesian coordinates we can write the camera equation on the form

$$\lambda \boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}(\boldsymbol{X} - \boldsymbol{t}), \tag{1}$$

where λ is the depth of the point, $\boldsymbol{x}$ the image point in homogeneous coordinates, $\boldsymbol{K}$ the camera calibration matrix, $\boldsymbol{R}$ the rotation of the camera, $\boldsymbol{X}$ the 3D point in cartesian coordinates and $\boldsymbol{t}$ is the camera center. For calibrated cameras we can exclude the calibration matrix and get

$$\lambda \boldsymbol{x} = \boldsymbol{R}(\boldsymbol{X} - \boldsymbol{t}). \tag{2}$$

From each one of the lines to the left in Figure 1 we get constraints on the form, $\boldsymbol{l}^T\boldsymbol{x} \leq 0$, where $\boldsymbol{l}$ is the equation of the line and $\boldsymbol{x}$ is the image point in homogeneous coordinates. Using the camera equation in (2) we can rewrite the constraints to

$$\boldsymbol{l}^T \frac{1}{\lambda}\boldsymbol{R}(\boldsymbol{X} - \boldsymbol{t}) \leq 0. \tag{3}$$

The depths has to be positive if the point is visible in the camera, therefore we can eliminate the depth from the equation. If we set $\boldsymbol{a} = \boldsymbol{l}^T\boldsymbol{R}$ we get

$$\boldsymbol{a}(\boldsymbol{X} - \boldsymbol{t}) \leq 0, \tag{4}$$

defining a plane through the camera center and the image plane as shown on the right in Figure 1.

For every 3D point there are four inequalities per camera, one for each plane. We collect all these inequalities into a matrix

$$\boldsymbol{A}\boldsymbol{u} \leq \boldsymbol{0}, \tag{5}$$

where $\boldsymbol{A}$ holds the coefficients and $\boldsymbol{u}$ holds the unknowns, consisting of the 3D points and the camera centers. Checking if there exists a solution that fulfills all constraints for a fixed error tolerance ϵ is a linear programming feasibility problem. To find the minimal ϵ we can use bisection.

A weakness of this approach is that it is sensitive to outliers. One way to get around this is to use auxiliary variables, as described in [11]. To each row of (5) we add a variable $s_i \geq 0$, yielding

$$\boldsymbol{a}_k^T \boldsymbol{x} \leq s_i. \tag{6}$$

There is one s_i for each measured image point. For example, if we have five views with ten image points in each view we get 50 auxiliary variables. Ideally we would like as many s_i's as possible to be zero, but since this is a very hard optimization problem we try to minimize the sum instead. This turns the convex feasibility problem in (5) into a convex optimization problem,

$$\begin{aligned} &\min \sum s_i \\ &\boldsymbol{A}\boldsymbol{u} \leq \boldsymbol{P}\boldsymbol{s}, \end{aligned} \tag{7}$$

where $\boldsymbol{P}$ is a matrix picking the relevant s_i.

2.2. Minimal Solver

This section presents a method to calculate the rotation and translation between two frames. The method is minimal in the sense that it requires a minimal number of correspondences. Assuming planar motion and known camera rotation with respect of the ground, we need two pairs of corresponding points.

For all cameras we have same rotation, $\boldsymbol{R}_0$, with respect to the ground and we can rewrite the camera equation (1)

$$\lambda \boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}_0\boldsymbol{R}_z\left(\boldsymbol{X} - \boldsymbol{t}\right), \tag{8}$$

where $\boldsymbol{R}_z$ is the rotation around the z-axis.

Using normalized image points $\boldsymbol{u} = \boldsymbol{R}_0^T\boldsymbol{K}^{-1}\boldsymbol{x}$ we get

$$\lambda \boldsymbol{u} = \boldsymbol{R}_z\left(\boldsymbol{X} - \boldsymbol{t}\right). \tag{9}$$

For the first camera we set both the rotation around the z-axis and the translation to zero, getting the simpler relations,

$$\lambda_k \boldsymbol{u}_k = \boldsymbol{X}_k, \quad \text{for } k = 1, 2 \tag{10}$$

where $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are the two image points.

For the second camera we let γ_1 and γ_2 denote the depths and use $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ for the image points. We get

$$\gamma_k \boldsymbol{R}_z^T \boldsymbol{v}_k = \left(\boldsymbol{X}_k - \boldsymbol{t}\right), \quad \text{for } k = 1, 2. \tag{11}$$

Since we are dealing with planar motion the translation is zero in the z-direction, so $\boldsymbol{t} = (t_x, t_y, 0)^T$. The rotation, $\boldsymbol{R}_z$, is given by a single rotation angle, θ.

If we know all four depths, λ_1, λ_2, γ_1 and γ_2, we can calculate the 3D points from (10) and the rotation and translation from (11). Hence, one way to solve the problem is to determine the depths. First note that the depth γ_k can be expressed in λ_k using the z-component of equations (10) and (11). We get

$$X_{kz} = \lambda_1 u_{kz} = \gamma_1 v_{kz} \text{ and } \gamma_k = \frac{u_{kz}}{v_{kz}}\lambda_k \tag{12}$$

To calculate the ratio between λ_1 and λ_2 we use the geometry of the scene. The scene seen from above is showed in Figure 2. For each camera we get one triangle with corners in the camera center and the two 3D points.

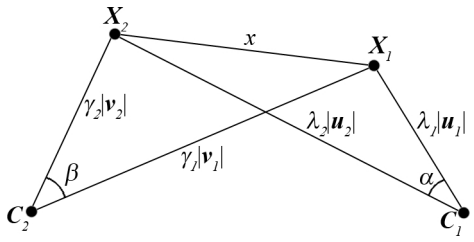

Figure 2. The scene seened from above

The two triangles share the side x and the length of this side can be calcluated from either of the triangles using the law of cosine. That gives us the equation

$$\begin{aligned} &\lambda_1^2\left|\boldsymbol{u}_1\right|^2 + \lambda_2^2\left|\boldsymbol{u}_2\right|^2 - 2\lambda_1\lambda_2\left|\boldsymbol{u}_1\right|\left|\boldsymbol{u}_2\right|\cos\alpha = \\ &\quad = \gamma_1^2\left|\boldsymbol{v}_1\right|^2 + \gamma_2^2\left|\boldsymbol{v}_2\right|^2 - 2\gamma_1\gamma_2\left|\boldsymbol{v}_1\right|\left|\boldsymbol{v}_2\right|\cos\beta, \end{aligned} \tag{13}$$

where α is the angle between the two vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, being the image points from the first camera. In the same way β is the angle between vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of the second camera.

By setting λ_1 to 1 we fixate the scale. Then we have three unknown depths and three equations. Inserting (12) into (13) yields an equation of second degree that gives us up to two real solutions for the depths.

For each of these solutions we calculate the rotation and translation. This is done by solving the linear equation system below. From (11) we have four unused equations,

$$\begin{cases} \gamma_1\left(av_{1x} + bv_{1y}\right) = X_{1x} - t_x \\ \gamma_1\left(-bv_{1x} + av_{1y}\right) = X_{1y} - t_y \\ \gamma_2\left(av_{2x} + bv_{2y}\right) = X_{2x} - t_x \\ \gamma_2\left(-bv_{2x} + av_{2y}\right) = X_{2y} - t_y, \end{cases} \tag{14}$$

where $a = \cos\theta$ and $b = \sin\theta$.

3. Tracking and Reconstruction

To do the reconstruction we have to find corresponding image points in the different frames in the video. To get that

we find interest points in one frame and track those points to the following frames. Then we cluster the points so that points in one cluster belongs to the same vehicle. When we have corresponding image points for a vehicle we use a few frames to do a 3D reconstruction. The reconstruction is then used to calculate the pose for all frames.

3.1. Tracking

To detect vehicles in the video, we start by finding interest points in one frame somewhere in the sequence. Then we track these points a number of frames using a KLT-tracker [12]. To separate points on stationary objects from points on moving objects we remove all points that has not moved between the first and the last frame. For the rest of the points we perform a motion segmentation algorithm to find out which points belongs to the same vehicle. For all the vehicles found in the scene the feature points are tracked until the vehicle leaves the scene. When all vehicles in the scene are found we choose another frame later in the sequence. This is repeated until we come to the end of the video.

The motion segmentation algorithm works by choosing two image points from two images, the first and the last image. From these two points the rotation and relative translation between the frames are calculated using the minimal solver from Section 2.2. The minimal solver requires that the rotation of the camera with respect to the ground is known. Depending on the number of real solutions for the equation of second degree in 13 the solver returns up to two solutions for the rotation and translation. For these solutions the rotation and translation are used to create two camera matrices per solution, one for each view. The camera matrices are calculated by

$$\boldsymbol{P}_1 = \boldsymbol{K}\boldsymbol{R}_0 \left(\boldsymbol{I}|\boldsymbol{0}\right) \text{ and } \boldsymbol{P}_2 = \boldsymbol{K}\boldsymbol{R}_0\boldsymbol{R}_z \left(\boldsymbol{I}|-\boldsymbol{t}\right), \quad (15)$$

where $\boldsymbol{K}$ is the camera calibration matrix, $\boldsymbol{R}_0$ is the camera rotation with respect to the ground, $\boldsymbol{R}_z$ is the rotation around the z-axis between the frames and $\boldsymbol{t}$ is the translation.

The camera matrices are then used to triangulate all image points to get the positions of the 3D points in space, the triangulation is done with the method proposed in [4]. The 3D points are then reprojected with both cameras

$$\lambda\hat{\boldsymbol{x}}_i = \boldsymbol{P}_i\boldsymbol{X}, \quad (16)$$

where $\hat{\boldsymbol{x}}_i$ is the reprojected image points for view i and $\boldsymbol{X}$ are the 3D points. The reprojection errors, $\epsilon_{ij} = \|\boldsymbol{x}_{ij} - \hat{\boldsymbol{x}}_{ij}\|_2$, are measured for all points in both views. Points, j, that have a small reprojection error in both views, $\epsilon_{1j} < \epsilon_t$ and $\epsilon_{2j} < \epsilon_t$, are classified as inliers and are likely to belong to the same vehicle. The number of inliers is counted for the different rotations and translations and then two new image points are chosen. The algorithm is then repeated for all pair of image points and the number of inliers for all possible solutions are counted. The solution that gives the highest number of inliers is chosen.

The points classified as inliers are then tracked with the KLT-tracker until the vehicle leaves the scene. Points that fails to be tracked are removed. First we track the points from one frame to the next and then back again, points that do not return to the original position are removed. We also remove points which comes to close to the edge of the image. We stop the tracking when there are no points left. To get longer tracks we start by following the vehicle forward in time, from the starting frame to the following frames until the vehicle drives away. Then we go back to the starting frame and follows the vehicle backwards in time, from the starting frame to previous frames until the vehicle disappears.

The points classified as inliers in this estimation are removed, and the motion segmentation algorithm is restarted to search for more vehicles. The motion segmentation algorithm is repeated until there are too few points left.

Since we choose new starting frames at short intervals we typically get several tracks of the same vehicle. To avoid reconstructing the same vehicle more than once, we try to detect this by comparing the position of the image points in the different tracks. Tracks where the positions for some image points coincide with image points in other tracks for corresponding frames are likely to represent the same vehicle.

For a frame in the first track the corresponding frames are found in the following tracks. The image points are compared between the different tracks, and if any image points coincide the tracks are merged together.

Tracking

The first frame in the video sequence is chosen as starting frame
1. Detect interest points.
2. Track the points a number of frames and remove points that does not move.
3. Find points belonging to the same vehicle by performing the motion segmentation algorithm.
4. Track the vehicle, until it leaves.
5. Remove points classified as inliers,.
6. Repeat step 3-5 until there are too few points left.
7. Choose a new starting frame, a few frames after the previous starting frame.
8. Repeat step 1-7 until the end of the video is reached.

3.2. Reconstruction

After the tracking algorithm we have tracks of corresponding image points from that the vehicles enter the camera field of view until the vehicles leave. We will use a few

of these views, typically 10, to make a 3D reconstruction of the vehicles.

First we have to know the rotation between the frames we have chosen. Still the vehicles only rotate around the z-axis and we can use the minimal solver from Section 2.2 to calculate the rotations for consecutive views. The rotations are calculated in the same way as in the motion segmentation. We calculate the rotation for two points at a time, then all points are triangulated and reprojected. This is done for all pairs of points and we choose the rotation that gives the highest number of inliers.

Knowing the rotation we can use the method from Section 2.1 to compute the reconstruction. To avoid translation ambiguity we fixate the position of the first camera. We also know that the vehicles do not translate in the z-direction. Then we can fixate the z-coordinate for all cameras, we also assume that the camera is placed above the vehicles and we set a upper limit of the z-coordinate for all 3D points. This limit we set somewhat lower than the height of the camera. Then we solve the optimization problem in (7). For the points that are inliers the value of the corresponding s_i will be very close to zero while outliers have much higher value of the corresponding s_i. To remove outliers we remove 3D points where the corresponding s_i is higher than some small threshold for all views.

Now we have a reconstruction of the vehicle, but the scale is still unknown. For surveillance purposes it is important that the scale is consistent with respect to the other reconstructed vehicles. To achieve this we consider the point in the vehicle model that has the lowest z-coordinate. Assuming that this point is close to the ground, we choose the scale such that this point gets z-coordinate equal to zero.

3.3. Pose

When we have both the correspondences between frames for the vehicle and a 3D reconstruction of it, we can calculate the position of the vehicle by calculating the camera pose for all views. If the rotation of the camera is known we can calculate the pose with the method presented in Section 2.1, though now we just have one camera and we know the position of the 3D points. Hence we just have to calculate the position of a camera, significantly reducing the number of unknown variables.

To find the rotation of the camera we perform a branch and bound search through rotation space. The branch and bound algorithm is described in detail in Section 4, where it is used for a larger search space. In pose estimation, we only have to perform the rotation search in one dimension since we already know the rotation of the camera with respect to the ground and only want to find the rotation around the z-axis.

To handle outliers, we use the following scheme. We choose some of the points on the vehicle at random, typically half of the points, and calculate the pose using the chosen points. All 3D points are then projected with the estimated camera and the number of inliers is counted. This procedure is repeated a number of times with different points and the rotation that gives the highest number of inliers is chosen. Then the pose is calculated again using all points that were classified as inliers.

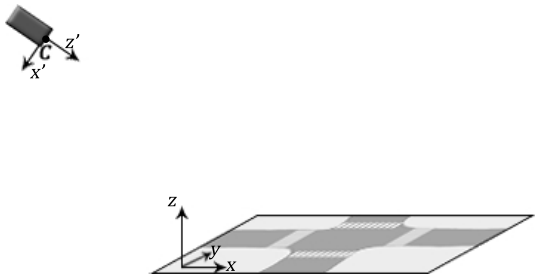

Figure 3. Rotation of the camera, the camera is first rotated around the z-axis, then around the x′-axis and at last around the z′-axis.

4. System Calibration

In all previous sections we assumed that the camera rotation with respect to the ground was known. To calibrate the system we need to find this rotation. This is done by choosing a few views of some vehicle, preferably a big one, with corresponding image points, without outliers. For this vehicle a 3D reconstruction is calculated, at the same time we get the rotations and translations of the cameras.

To find the rotations between the different views we perform a branch-and-bound search, similar to that in [3]. The parameterization of the rotation is shown in Figure 3. The camera is first rotated around the z-axis, then around the x′-axis and at last around the z′-axis and the total rotation can be written as $\boldsymbol{R} = \boldsymbol{R}_{z'}\boldsymbol{R}_{x'}\boldsymbol{R}_{z},$, where $\boldsymbol{R}$ is the total rotation and $\boldsymbol{R}_{z'}$, $\boldsymbol{R}_{x'}$ and $\boldsymbol{R}_{z}$ are the rotations around the z′-, x′- and z-axes respectively.

We choose coordinate system, such that the z-axis is perpendicular to the ground plane. This means that vehicles rotate only around the z-axis and thus $\boldsymbol{R}_{z'}$, $\boldsymbol{R}_{x'}$ are equal for all views. They specify the orientation of the ground plane relative to the camera. The rotation around the z-axis for the first camera can be chosen to be zero. Each of the angles can be chosen between -π and π. Thus our search space can be identified with the product space $[-\pi, \pi]^{(N+1)}$, where N is the number of cameras.

The branch and bound algorithm is initiated with a list containing one block, $[-\pi, \pi]^{(N+1)}$ as well as an initial error threshold ϵ. At each iteration, we pick a block from the list and try to determine if this block can contain any solution with reprojection errors $< \epsilon$. This is determined by solving an LP feasibility problem as described in Section 2.1 but with error tolerance $\epsilon + \Delta$ instead of ϵ. The

Branch-and-bound Algorithm

Iterate until desired precision is reached.

1. Pick the first block from the list.

2. Calculate the constraints and set up the LP problem.

3. Determine if there is a solution to the LP problem.

4. If there is a solution.

- Divide the block into smaller blocks and add them to the list.

- Try to update the error threshold by performing the bisection algorithm.

5. Remove the current block from the list.

Δ is an extra uncertainty that accounts for the size of the block. Details can be found in [7].

If this test falls out positive, then the block is divided and the new blocks are added to the list. Otherwise it is simply deleted. This continues until the remaining set of rotations is small enough.

4.1. Bisection

In Section 2.1 we showed how to check feasibility for a fixed error tolerance ϵ. To find the L_∞ optimal solution we also need a method to update this tolerance. This is done with a bisection algorithm. For blocks that pass the feasibility test we try to find a solution having a smaller reprojection error. We fix the rotations to the middle of the block and perform a feasibility test with error tolerance ϵ. If this passes we know that we have found a better solution. To know how good, we use bisection.

We start with the interval $[0, \epsilon]$. Let $\gamma = \epsilon/2$ and check feasibility with error tolerance γ. If this is feasible we know that the best reprojection error is somewhere between 0 and γ and we set the upper bound to γ. If there is not a solution, we set the lower bound to γ instead. The interval is now half the length of the original interval. Again we try to find a solution in the middle of the interval and change either the upper or the lower bound. This is repeated until the interval is as short as desired. Finally we update the error threshold.

5. Experiments

The presented methods were evaluated on real-world data. The captured video has a resolution of 320 $\times$ 240 pixels and is around 10 minutes. The following sections describe the different parts of the evaluation.

5.1. Motion segmentation

To illustrate how the motion segmentation algorithm works, one frame from the video was selected. In this frame interest points were detected and tracked for a couple of frames. Figure 4 a) shows the first and last image with the image points marked with yellow dots. After removing stationary points, the points in Figure 4 b) remained. The result from the motion segmentation is shown in Figure 4 c), the green dots are points belonging to the first vehicle and the red dots are points belonging to the second vehicle.

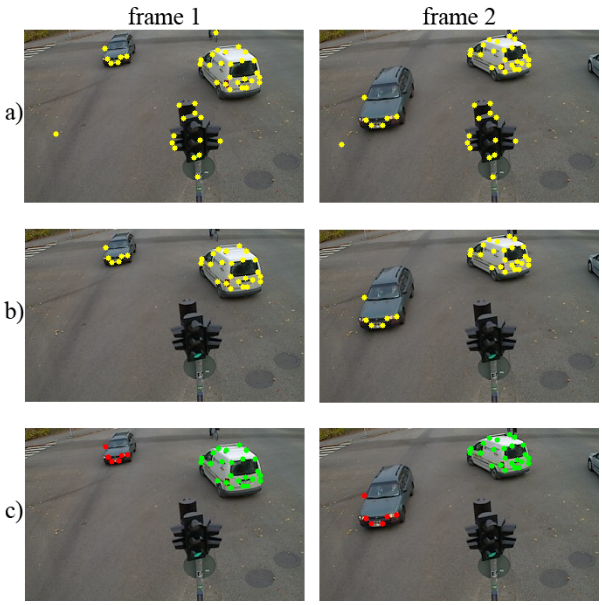

Figure 4. Illustration of the motion segmentation algorithm.

5.2. Tracking

To evaluate the tracking algorithm, we manually counted the number of vehicles in the entire video sequence to get the ground truth. We also noted in which direction the vehicles are driving. There are 16 ways to pass through the intersection. Next we calculated the number of vehicles that the tracking algorithm found and compared this number with the ground truth. The results are shown in Table 1. For some of the directions the tracking algorithm finds most of the vehicles. For other, the tracking algorithm fails more often. In the cases where the tracking algorithm works well, the vehicles drive closer to the camera, which gives fairly well resolution of the vehicles. Vehicles that drive far away is harder to track since they are very small. The KLT-tracker has to be able to to track points on the vehicle for a number of frames for it to be detected, and when it fails, they will not be detected.

5.3. Reconstruction and Pose

The reconstructions of the vehicles are made according to Section 3.2 and the pose for the vehicles in all frames where the vehicles are visible is calculated as in Section 3.3. For all frames, the L_∞-norm of the reprojection errors is calculated, that is the largest of the reprojection errors. Ta-

turn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	tot
ground truth	34	120	0	0	25	17	8	0	32	102	27	1	12	13	15	0	406
tracking	32	30	2	0	17	17	12	0	21	63	6	0	6	7	0	0	213

Table 1. Comparison between the real number of vehicles and the number the tracking algorithm finds. Each column represent one of the 16 different ways to pass through the intersection.

ble 2 shows a summary for 16 vehicles. The reprojection errors are coordinate-wise and measured in pixels, the resolution of the images is 320 $\times$ 240 pixels. The number of points used to do the reconstruction varies between vehicles and the number of points used to calculate the pose varies between frames.

Figure 5 shows three frames for which the pose of the vehicles has been calculated. The dots representing image points on the different vehicles have different colors.

Figure 5. Three frames from the video. The pose for the three vehicles has been calculated for the frames where the vehicles are visible.

The 3D reconstruction of the vehicles and their relative position is shown in Figure 6. The points numbered with number 1 represent the frame to the left in Figure 5, the points numbered 2 represent the middle frame and points with number 3 represent the frame to the right.

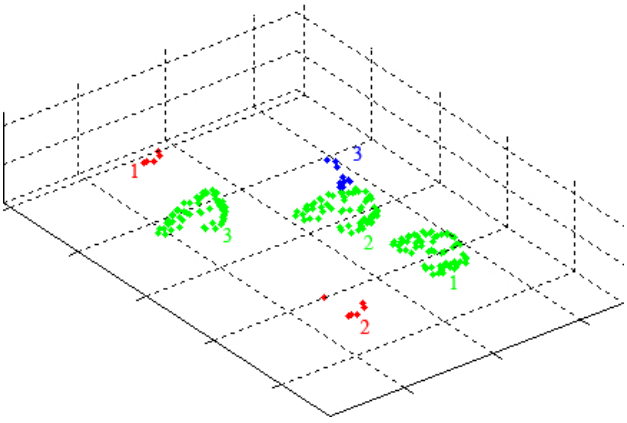

Figure 6. 3D reconstruction and pose for three frames in the video. Different colors represent different vehicles.

Next, the 3D points were projected onto the ground plane to estimate the positions of the vehicles in the intersection, the result can be seen Figure 7. The positions can be compared with the images in Figure 5.

Finally the pose of the vehicles has been calculated for all frames where they are visible. Figure 8 shows the projection of the vehicles for all frames.

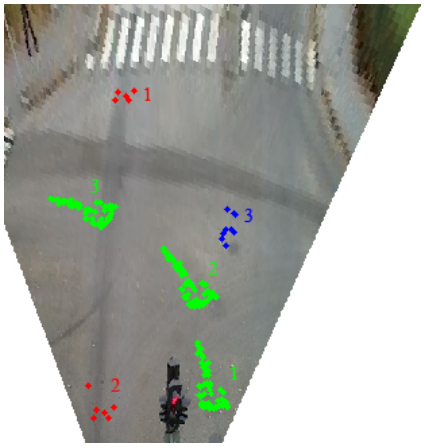

Figure 7. Projection of the 3D points into the ground plane to show the position of the vehicles. The positions for three different time points are shown. Like before different colors represent different vehicles.

Figure 8. Same as Figure 7 but the positions for all time points are shown. Points above the crosswalk are removed during the tracking algorithm.

5.4. Calibration

To estimate the camera rotation with respect to the ground plane, three images of a bus driving through the intersection were used. Figure 9 shows the images used, while the result of the reconstruction can be seen in Figure 10.

Now it is possible to create a map of the intersection by rectifying an image using the estimated ground plane. This map was used to produce the images in Figures 7 and 8. The accuracy of this estimation should give us a rough quality measure.

vehicle no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
worst frame	4.10	3.28	4.33	3.66	3.44	4.69	2.56	3.81	3.63	3.08	4.69	2.32	4.04	3.85	4.02	2.86
best frame	1.54	1.03	1.99	0.73	1.28	0.96	0.21	1.03	0.83	0.68	0.67	0.74	0.32	0.43	1.46	0.19
mean	2.87	1.92	2.82	2.22	2.29	1.86	1.40	1.53	2.08	1.57	1.82	1.64	2.45	2.11	3.00	0.82

Table 2. L_∞ errors for 16 of the vehicles in the intersection, the table displays the largest, smallest and mean value of the L_∞ norm of the reprojection errors. The errors are measured in pixels and the resolution of the images is 320 $\times$ 240 pixels. The number of points varies between vehicles and frames.

Figure 9. Three images of a bus driving through the intersection. The yellow dots mark the image points which were used for the reconstruction and the yellow lines show the boundaries of two sides of the bus.

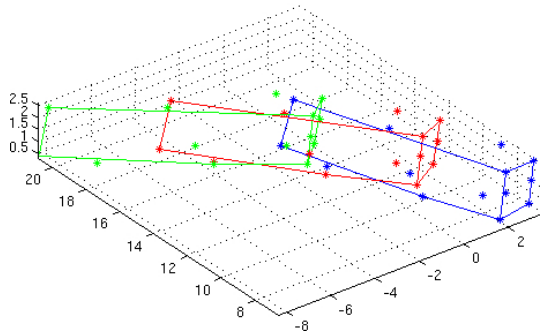

Figure 10. Reconstruction of the bus driving through the intersection at three time points. The tracked points used for the reconstruction are marked as well as some lines used for visualization. The blue bus corresponds to the first time point, the red one to the second and the green to the third.

By comparing distance in the rectified image and a real map of the intersection, we can estimate the size of the bus. The height was estimated to 2.65 m, the width to 2.15 m and the length to 12.7 m. The bus is 2.8 m high, 2.3 m wide and 12 m long, giving us an average error of 0.3 m. That is significantly better than the approach in [8] of simply projecting a segmented object onto the ground which in this case would give an error of several meters.

6. Conclusions

We presented some ideas on how to achieve accurate 3D reconstructions for traffic surveillance. The approach builds on recent research in optimal methods for computer vision. This makes it possible to fully exploit restrictions such as the ground being planar. The approach requires only one camera, which means that no synchronization between cameras is needed. Moreover, the camera position with respect to the ground can be estimated automatically as described in Section 4. Altogether the system is accurate, yet easy to set up and these are characteristics that should be attractive to many traffic scientists.

References

[1] B. Bose and E. Grimson. Ground plane rectification by tracking moving objects. In *IEEE International Workshop on Visual Surveillance and PETS*, 2004. 1

[2] B. Coifman, B. C. Corresponding, P. Mclauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance, 1998. 1

[3] R. Hartley and F. Kahl. Global optimization through rotation space search. *Int. Journal Computer Vision*, 2009. 5

[4] R. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997. 4

[5] C. Hydén. *The development of a method for traffic safety evaluation: The Swedish Traffic Conflicts Technique*. PhD thesis, Department of Traffic Planning and Engineering, Lund University, 1987. 1

[6] F. Kahl and R. Hartley. Multiple view geometry under the L_∞-norm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(9):1603–1617, 2008. 2

[7] H. Källén. 3d reconstruction for traffic surveillance. Master's thesis, Dept. of Mathematics, Lund University, Sweden, 2009. 6

[8] A. Laureshyn and H. Ardö. Automated video analysis as a tool for analysing road user behaviour. In *ITS World Congress*, London, UK, 2006. 1, 8

[9] B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool. Coupled object detection and tracking from static cameras and moving vehicles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1683–1698, Oct. 2008. 1

[10] K. Müller, A. Smolic, M. Drose, P. Voigt, and T. Wiegand. 3-d reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. *IEEE Trans. Circuits Syst. Video Techn.*, 15(4):538–549, 2005. 1

[11] C. Olsson, A. Eriksson, and R. Hartley. Outlier removal using duality. In *CVPR 2010*, 2010. 3

[12] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Computer Science Department, Pittsburgh, PA, April 1991. 1, 4

[13] Z. Zhang, M. Li, K. Huang, and T. Tan. Practical camera auto-calibration based on object appearance and motion for traffic scene visual surveillance. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008. 1