Vehicle Trajectory Extraction in Traffic Scene Based on Monocular Vision

Wang Ying¹, Zhao Liang²

¹Zhonghuan Information College, Tianjin University of Technology, Tianjin, 300380, China
²Automotive Data of China (Tianjin) Co., Ltd., Tianjin, 300300, China

Abstract. Accurately extracting the trajectory information of multiple vehicles in urban traffic videos is of great significance in applications. It can help analyze and predict drivers’ behavior, count road flow, and restore accident scenarios. However, due to the complexity of the interaction between vehicles and the environment, it is still a challenging problem to accurately extract the trajectory of multi-target vehicles in urban traffic scenes, which often requires the assistance of a variety of different devices, which increases the practicality and difficulty of application. A method is proposed based on the combination of detection and tracking. This method only needs to use a video stream captured by a fixed panoramic camera and the coordinate information of several feature points in the scene to extract the speed and trajectory information of the vehicle in the scene. Different optimization methods are used in each link to reduce errors, and build a simulation environment for the test scenario, reconstruct and verify the extracted trajectory information in the simulation environment. Experiments have proved that the effect of the method is basically satisfactory. It has the advantages of fewer constraints, convenient deployment, and accurate restoration while ensuring a certain real-time performance, which has practical engineering significance.

1. Overview of research

With the popularity of the concept of smart city, deep learning is becoming more and more widely used in the analysis of urban traffic videos. By extracting vehicle location, speed, and trajectory information from large-scale videos, it’s of great importance for analyzing road flow, predicting driver behaviors and monitoring accidents. However, it is still a difficult task to obtain accurate vehicle trajectories in massive traffic videos at low cost. Firstly, the camera calibration data is usually not easy to obtain. Secondly, due to the complexity of the interaction between vehicles and the environments in urban traffic scenes, the loss of tracking and the switching of IDs are two thorny issues without good solutions.

Currently the multi-object tracking method based on deep learning has become the mainstream. This approach was first proposed in [1] (SORT). It uses neural networks to detect vehicles and bipartite matching [10] to associate objects in different frames. In addition, a simple Kalman filtering is used to optimize the tracking results in the image space. Reference [2] improves on the basis of [1]. It combines the feature vectors from the network and the state vectors from Kalman filtering to associate tracks. Reference [3] adds a branch in the detection network to extract the ID information of the object for use in the subsequent association phase. All these methods run by separating the detection and the association in two stages, which causes efficiency problems. Some follow-up studies have tried to combine the stages into one. For example, reference[4] proposes to use a heat map from the previous
frame combined with the current frame as network input, and obtains the relative displacement of the center of the detected object at the output. This method balances the speed and accuracy, but since only the information of two adjacent frames is used in each round, the problem of ID switching still occurs during long-range tracking. Hence it’s difficult to obtain a relatively complete trajectory when used to extract vehicle trajectories.

The purpose of this paper is to propose a more exact approach to extract vehicle trajectories. Our method runs in an offline manner. We reduce the error in detection and tracking stages by combining several optimization algorithms.

1. Firstly we calibrate the camera’s intrinsic parameters and extrinsic parameters based on GPS information at several feature points in the scene.
2. Then we use neural networks to extract vehicles in the video stream frame by frame. Here YOLOv3 [8] trained on the coco data set [9] is used.
3. We then use bipartite matching and optical flow to associate and stitch trajectory fragments.
4. Then convert the pixel locations of vehicle trajectories to geodetic coordinates, and use Kalman filtering and RTS smoothing to smooth the trajectory.
5. Use the data obtained in step 4 to reconstruct the scene in a virtual simulation environment to verify the results.

Compared with [1][2][3][4], our method sacrifices some real-time performance, but saves the step of camera calibration and the trajectory obtained is more complete. The following is a detailed explanation of our method.

2. Algorithm flow

2.1. Camera calibration

We assume the camera is located significantly higher than the ground to avoid occlusions. We also assume the GPS data of more than four landmark points are known.

Choose one of the points as the origin $O$, take the north direction as the positive axis $x$, the east direction as the positive axis $y$, the vertical ground downward as the positive axis $z$ and establish a right-handed cartesian system $Oxyz$. We specify the ground as plane $z = 0$, and convert all latitude and longitude coordinates to cartesian coordinates. The cartesian system is also called the world coordinate system.

In order to convert between world coordinates and pixel coordinates we need to know the intrinsic and extrinsic parameters of the camera. An approximating procedure is used here. Firstly we assume the camera is modeled by the ideal pinhole camera model [5], so its distortion coefficients are all zero and the image center is located on the main axis of the camera, so the camera matrix $K$ is like

$$
K = \begin{pmatrix}
    f & 0 & w/2 \\
    0 & f & h/2 \\
    0 & 0 & 1
\end{pmatrix}.
$$

Where $w, h$ are the width and the height of the image, $f$ is the focal length and is the only unknown. $K$ is a single parameter function of $f$. We denote $K$ by $K_f$ to indicate this relation. For any given $f$, we can use $K_f$ to get the rotation matrix and translation vector $R, T$.

According to the pinhole camera model [5] mapping relations between world coordinates $P = (X, Y, Z)$ and image coordinates is given as follows.

$$
\begin{pmatrix}
    x \\
    y \\
    1
\end{pmatrix} = K \begin{pmatrix}
    X \\
    Y \\
    Z + T
\end{pmatrix}.
$$

(1)
s is a nonzero real number. The conversion F from world coordinates to image coordinates is given by:

\[
\begin{pmatrix}
  x \\
  y \\
  \end{pmatrix} = F(p) = \frac{\left( a_{11}X + a_{12}Y + a_{13}Z + b_1 \right)}{\left( a_{31}X + a_{32}Y + a_{33}Z + b_3 \right)} \quad (2)
\]

Where \( A = KR = (a_j)_{i,j=1,3}, \quad B = KT = (b_1, b_2, b_3)^T \).

When calculating the world coordinates \( P \) from the pixel coordinates \( p \), we must specify the plane \( Z = a \) that \( P \) lies on because of dimension reduction. We have

\[
sR^T K^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + R^T T.
\]

Let \( C = R^T K^{-1} = (c_j)_{i,j=1,3}, \quad D = R^T T = (d_1, d_2, d_3)^T \), comparing the third component on both sides of the upper formula we get

\[
s = \frac{a + d_3}{c_{31}X + c_{32}Y + c_{33}} \quad (3)
\]

By comparing the first two components, we can get the expression of \( p \) as a function of \( P \)

\[
F_{inv}(P) = \begin{pmatrix} X \\ Y \\ \end{pmatrix} = \begin{pmatrix} sc_{11}x + sc_{12}y + sc_{13} - d_1 \\ sc_{21}x + sc_{22}y + sc_{23} - d_2 \\ \end{pmatrix} \quad (4)
\]

Based on the above analysis, let \( \{ p_i = (X_i, Y_i, 0), 1 \leq i \leq n \} \) be the \( n \) feature points. Their corresponding pixel coordinates in the video are \( \{ p_i = (x_i, y_i), 1 \leq i \leq n \} \) respectively. For any given \( K_f \) we use the solvePnP procedure and the \( n \) points to get the corresponding rotation matrix \( R_f \) and translation vector \( T_f \), so the corresponding mapping \( F_f \) is obtained. We define the loss function as the sum of the mean square error between the original pixels \( p_i \) and the reprojected pixel locations of \( P_i \):

\[
err(f) = \sum_{i=1}^n \left\| F_f(P_i) - p_i \right\|^2.
\]

The goal is to find \( f_{\min} \) that minimizes the lost under the constraint \( f > 0 \). We can use nonlinear optimization functions like sequential least squares programming [7] (SLSQ) to solve this problem, and use \( f_{\min} \) as the focal length. Then camera matrix \( K \) is obtained and we use solvePnP procedure again to get the corresponding rotation matrix \( R \) and translation vector \( T \).

2.2. Multi-target vehicle detection and tracking

The next step is to detect vehicles in the video and associate the detections from different frames to form a stable multi-object tracking. We use YOLOv3 as the convolutional net to detect the video frame by frame, the result of each frame is a list of objects, where each object contains the following attributes.

1. ID in this frame
2. Classification label.
3. Confidence score.
4. The bounding box \((x_1, y_1, x_2, y_2)\), where \((x_1, y_1)\) is the pixel coordinates of the upper left corner, \((x_2, y_2)\) is the pixel coordinates of the lower right corner.

To associate objects in different frames, we maintain a list of trajectories \( \Gamma = \{ T_1, \ldots, T_m \} \), each track \( T_j \in \Gamma \) holds the following information:
1. The list of classification labels from each detection.
2. A sequence of bounding boxes \( B_k, \ldots, B_{k+l} \), where \( k \) is the index of the first frame in which the vehicle is detected. \( k + l \) is the index of the last frame in which the vehicle is detected. We allow certain \( B_i \) to be null, which means that the vehicle is not detected in the \( i \)-th frame.
3. A bool value. It’s true when the trajectory is active (the vehicle is still in the field of view) and false when the trajectory is dead (the vehicle is out of the field of view). \( \Gamma_{\text{active}} \) is the subset of all active trajectories.

Let \( X = \{X_1, \ldots, X_n\} \) be the set of vehicles detected in the current frame, and their bounding boxes are \( \{B_1, \ldots, B_n\} \) respectively. We associate \( X \) and \( \Gamma_{\text{active}} \) after each frame and update \( \Gamma \) and \( \Gamma_{\text{active}} \) along the way:

1. Choose a threshold \( N \) and \( \alpha \) such as \( N = 30, \ \alpha = 0.3 \).
2. Let \( \Gamma_{\text{active}} = \{T_1, \ldots, T_M\} \) be the set of active tracks, and repeat step 3 for each \( k = 0, \ldots, N \).
3. Let \( \{C_1, \ldots, C_M\} \) be the bounding boxes of \( \{T_1, \ldots, T_M\} \) in frame \( t - k \) and define the bipartite graph \( G \) as follows. The vertices of \( G \) is \( \{C_1, \ldots, C_M\} \cup \{B_1, \ldots, B_n\} \), and its edges are like \( e = (C_p, B_q) \), the weight of \( e \) is the IOU score between \( C_p \) and \( B_q \):

\[
\text{weight}(e) = \text{iou}(C_p, B_q) = \frac{|C_p \cap B_q|}{|C_p \cup B_q|}
\]

Using the Hungarian algorithm, we can obtain a maximal bipart matching of \( G \). If \( (C_p, B_q) \) belongs to this matching and \( \text{iou}(C_p, B_q) > \alpha \), we then associate the object \( X_q \) with the trajectory \( T_p \).

Note that once \( X_q \) matches \( T_p \), both of them will not be considered in the next loop.

4. At the end of the cycle of step 2, if an object \( X_q \) does not match any active trajectories, it is considered a new vehicle and we assign it a new trajectory and add this trajectory to \( \Gamma_{\text{active}} \).
5. Finally check each track in \( \Gamma_{\text{active}} \). If a track has not been updated for more than \( N \) frames then we think it’s dead and remove it from \( \Gamma_{\text{active}} \).
6. After all tracks are obtained as a list \( \Gamma_{\text{final}} \). For each pair of tracks \( T_1, T_2 \in \Gamma_{\text{final}} \), we use the following rule to determine whether \( T_1, T_2 \) are created by the same vehicle and stitch them together.

   1. The start time of \( T_2 \) must be later than the end time of \( T_1 \), but not later than 3 seconds.
   2. The classification labels of \( T_1, T_2 \) are the same (the label of a track is determined by its label with the highest frequency).
   3. Start an optical flow tracking fromm the last detection of \( T_1 \), and check if its bounding box matches with \( T_2 \) at some time in the first 10 frames of \( T_2 \).

   If all the three restrictions above are satisfied, we merge \( T_1 \) and \( T_2 \) into the same track.

At this point, all vehicle tracks from the video are extracted. The next step is to convert them to the world coordinates on the real ground.

2.3. Vehicle world coordinate positioning and trajectory optimization

In the calibration step we have the mapping \( F_{\text{inv}} \) that converts pixels to points on plane \( Z = a \) in world sytem. For a given bounding box \( B_i \) from a track at time \( t \), we choose a pixel \( p \) in \( B_i \) and use \( F_{\text{inv}} \) to project it to a suitable plane \( Z = a \), and use the resulting \( (X, Y) \) components as the vehicle ground location. The usual choice is to take \( p \) as the center or the midpoint of the bottom edge,
but due to the perspective principle, the projected coordinates will be farther than the true values if \( p \) is taken as the center of \( B_r \). And will be closer if \( p \) is taken as the midpoint of the bottom edge.

For this reason we use some prior knowledge from reality, that is, a car has height roughly about 1.6 meters and buses and trucks have height roughly about 2 meters. We map the center \( p \) to the plane \( Z = -1.6 \) or \( Z = -2 \) according to the vehicle type. This gives us a smaller projection error. All the non-null boxes in a track \( T \) is projected in this way, and a list of trajectory points in the world system is obtained. Such a trajectory is usually not continuous in time, and due to the accumulation of errors in each step, the dispersion range is also relatively large. We then use the extended Kalman filtering to optimize the result.

We use the bicycle model from [11] for the EKF processing, the model state vector is \( x = (x, y, v, \varphi, \beta)^T \). Where \( x, y \) is the ground coordinates of the vehicle centroid, \( v \) is the absolute value of the vehicle speed, \( \varphi \) is the yaw angle and \( \beta \) is the slipping angle of the front wheel. They system model equation is

\[
\begin{align*}
\dot{x} &= v \cos(\varphi + \beta) \\
\dot{y} &= v \sin(\varphi + \beta) \\
\dot{v} &= u_t \\
\dot{\varphi} &= v \sin(\beta)/l_r \\
\dot{\beta} &= u_\varphi
\end{align*}
\]

Where \( l_r \) is the distance between the centroid and the rear axle, and \( u_t \) is the acceleration of the vehicle. \( u_\varphi \) is determined by driver’s control.

The observation model is the pinhole camera model obtained from the first two components of the \( x \) mapping \( F \) given by (2).

For every active track \( T \in \Gamma_{\text{active}} \) its initial state \( x \) is zero. By using the centers of the bounding boxes as observations and substitute them into the EKF, an optimized trajectory for each vehicle is obtained.

Note the Kalman filtering takes a number of iterations to converge and the estimation errors in initial stages are relatively large. Since most vehicles only appear for a few seconds, Kalman filtering is not guaranteed to convergence. In particular, the speed of the vehicles in their first frames cannot be given. We use the Rauch-Tung-Striebel [6] method to perform further smoothing. In fact RTS smoothing is the linear minimum variance estimation that including the information of the entire history. The following figure shows the difference between with and without RTS smoothing:
We can seen that RTS smoothing do improve the trajectory accuracy.

3. Simulation verification
We built a virtual scene in the game development engine UE, and used the extracted trajectory data as input to rebuild the scene in the video. The result animation runs smoothly, and the trajectories is consistent with the trajectory in the real video, thus proves the feasibility of our method.
4. Conclusion
A method is proposed to extract vehicle trajectory from videos, which requires less constraints and has enough accuracy as verified by our simulation. It is a practical method with potential applications.

Acknowledgment
Project funds: Young Teachers’ Research and Nursery Fund of Zhonghuan Information College Tianjin University of Technology (2020KYLG09)

References
[1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft. “Simple Online and Realtime Tracking”. arXiv:1602.00763.
[2] Nicolai Wojke, Alex Bewley, Dietrich Paulus.. “Simple Online and Realtime Tracking with a Deep Association Metric”. arXiv:1703.07402.
[3] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, Shengjin Wang. “Towards Real-Time Multi-Object Tracking”. arXiv:1909.12605.
[4] Xingyi Zhou, Vladlen Koltun, Philipp Krähenbühl. “Tracking Objects as Points”. arXiv:2004.01177.
[5] Hartley, R.-I. and Zisserman, A. "Multiple View Geometry in Computer Vision". Cambridge University Press. 2004.
[6] Rauch, H.E.; Tung, F.; Striebel, C. T. (August 1965). "Maximum likelihood estimates of linear dynamic systems". AIAA Journal. 3 (8): 1445–1450.
[7] Dieter Kraft, “A Software Package for Sequential Quadratic Programming”, DFVLR-FB 88-28, 1988.
[8] Joseph Redmon, Ali Farhadi. "YOLOv3: An Incremental Improvement". arXiv:1804.0276.
[9] https://cocodataset.org/.
[10] J. Munkres, "Algorithms for the Assignment and Transportation Problems", Journal of the Society for Industrial and Applied Mathematics, 5(1):32–38, 1957 March.
[11] P. Polack, F. Alth, B. d'Andra-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles". in 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 812–818, June 2017.