AutoCalib: Automatic Traffic Camera Calibration at Scale

ROMIL BHARDWAJ, Microsoft Research

GOPI KRISHNA TUMMALA, The Ohio State University

GANESAN RAMALINGAM and RAMACHANDRAN RAMJEE, Microsoft Research

PRASUN SINHA, The Ohio State University

Emerging smart cities are typically equipped with thousands of outdoor cameras. However, these cameras are usually not calibrated, i.e., information such as their precise mounting height and orientation is not available. Calibrating these cameras allows measurement of real-world distances from the video, thereby enabling a wide range of novel applications such as *identifying speeding vehicles* and *city road planning*. Unfortunately, robust camera calibration is a manual process today and is not scalable. In this article, we propose AutoCalib, a system for scalable, automatic calibration of traffic cameras. AutoCalib exploits deep learning to extract selected key-point features from car images in the video and uses a novel filtering and aggregation algorithm to automatically produce a robust estimate of the camera calibration parameters from just hundreds of samples. We have implemented AutoCalib as a service on Azure that takes in a video segment and computes the camera calibration parameters. Using video from real-world traffic cameras, we show that AutoCalib is able to estimate real-world distances with an error of less than 12%.

CCS Concepts: • **Computer systems organization** → **Sensor networks**;

Additional Key Words and Phrases: Traffic camera calibration, key-point detection, vehicle detection

ACM Reference format:

Romil Bhardwaj, Gopi Krishna Tummalala, Ganesan Ramalingam, Ramachandran Ramjee, and Prasun Sinha. 2018. AutoCalib: Automatic Traffic Camera Calibration at Scale. *ACM Trans. Sen. Netw.* 14, 3–4, Article 19 (November 2018), 27 pages.

<https://doi.org/10.1145/3199667>

1 INTRODUCTION

Capitalizing on the dramatic reduction in cost of sensing devices, many smart cities are deploying public cameras at large scale [1–4]. For example, according to one estimate, there are as many as 100 million public cameras in China [2]. In this article, we focus on public cameras that are installed to observe road traffic, called *traffic cameras*.

Video feeds from many such public traffic cameras are freely available today [7]. To facilitate building interesting applications from these video feeds, these traffic cameras need to be *calibrated*. Calibration allows automatic estimation of real-world distances in traffic videos, thereby enabling development of several novel smart city applications:

R. Bhardwaj and G. K. Tummalala are co-primary authors.

Authors' addresses: R. Bhardwaj, G. Ramalingam, and R. Ramjee, Microsoft Research, Bangalore, KA, India, 560001; emails: romilb@eecs.berkeley.edu, {grama, ramjee}@microsoft.com; G. K. Tummalala and P. Sinha, The Ohio State University, Columbus, OH, USA 43210; emails: tummalala.10@osu.edu, prasun@cse.ohio-state.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

1550-4859/2018/11-ART19 \$15.00

<https://doi.org/10.1145/3199667>

- (1) *Road safety*: Nearly 1.3 million people die due to road accidents every year [6]. Calibrated traffic cameras can help in automatic estimation of vehicle speed, enabling always-on enforcement of road speed limits. Calibrated cameras can also estimate inter-vehicular and human-vehicle separations, which can help accident prediction and prevention applications [21].
- (2) *City planning*: Calibrated cameras can help generate automatic traffic reports such as count of various vehicles (car, bus, bicycle) and their flows (direction, turns, etc.) [21]. This can help city officials in planning, for example, the introduction of special bike lanes. In addition, calibrated cameras can enable automatic toll and parking fee payments [28, 29].
- (3) *Multi-camera fusion*: Calibration brings different cameras viewing a scene to the same frame of reference, which is essential for multi-camera fusion applications. Different cameras can collaborate to construct a three-dimensional (3D) view [22] by fusing the observed images. Finally, automatic calibration equips cameras to steer themselves and provide various services [16].

In this article, we design and implement a system called AutoCalib, which takes in a video snippet from a given traffic camera and automatically computes its calibration parameters. AutoCalib uses a custom-trained deep neural network (Section 4) to automatically determine the location of several carefully selected key-points from the image of a car (e.g., taillights, mirrors, etc.). AutoCalib then uses a novel filtering and aggregation algorithm that matches these feature coordinates against known dimensions of the same features of the most popular car models, filters out outliers (e.g., car model mis-matches or errors in feature identification), and aggregates the rest to produce a low-error calibration.

Camera calibration involves estimating two types of camera parameters, viz., the intrinsic parameters, such as focal length and distortion matrix of the camera, and the extrinsic parameters that are orientation (represented by rotation matrix R) and position of the camera in real-world coordinates (T). In this article, we focus on automatic estimation of the extrinsic parameters of traffic cameras and assume that the intrinsic parameters, which are based on the camera's make/model, are known.

Estimation of camera extrinsic parameters is challenging today as it requires manual effort. When the cameras are installed, the primary objective is visibility of the scene. Physically measuring orientation of the installed camera is challenging due to the poor accuracy of commodity compasses that are affected by electromagnetic properties of the environment. Thus, camera calibration today is done virtually by visually identifying four or more landmarks in the scene of the camera, estimating their real-world coordinates using an application such as Google Earth [5] and utilizing these real-world coordinates to calculate the extrinsic parameters using a standard vision-based solver (Section 2). This process is error prone and manual, requiring enormous human effort for calibrating millions of already-installed cameras. Further, for advanced cameras that have pan, tilt, and zoom (PTZ) capabilities, the calibration has to be redone whenever the PTZ parameters are changed by the authorities (which we see often in our dataset [7]).

While the problem of automatic traffic camera calibration has been studied (Section 3), most prior work make strong assumptions (e.g., straight-line motion of vehicles) that are often violated, resulting in high error calibrations. In contrast, AutoCalib only assumes that known popular car models will occur with sufficient frequency in the video and utilizes the cars' known geometric properties (e.g., distance between the two taillights of a Honda Civic).

Advances in deep learning has resulted in high accuracy for image classification tasks [18]. Further, there exist pre-trained models for several image classification tasks. For example, CompCars [33] is a pre-trained deep neural network (DNN) that can classify a car model from an image with high accuracy. Unfortunately, we were unable to directly use this pre-trained model for our

needs, because the image of a single car forms a small part of the typical traffic camera view as seen in the leftmost image in Figure 1.¹ The resolution of the car image is thus of not sufficiently high quality for the pre-trained model to accurately identify the model of the car (visually, the authors were also unable to identify the models).

However, the features of the car, such as the mirrors and taillights, are still clearly visible in the image. Thus, we built an annotation tool that would automatically label the location of selected features of a car from these video images. While deep learning typically requires thousands of labeled images for training, we use transfer learning [18] to build a custom DNN from the pre-trained CompCars DNN using only about 500 manually annotated images containing labels of car features. In Section 6, we show that our annotation tool has a median error of only 6% of the car width, when compared to the manually annotated labels.

The output of the annotation tool produces the locations of various car features in the image. From this, we can calculate the distance between the feature points in the image coordinate system. We now need their corresponding distances in the real world. Since we are unable to automatically identify the car model, we only have a set of candidate distances (e.g., one each for the top 10 popular car models). The challenge then is to estimate the calibration parameters that result in the best match with one of these car models. Errors are possible due to a variety of reasons, including the car in the image not belonging to any of the top 10 models, errors in the annotation tool, and so on. We develop a novel filtering and aggregation algorithm that computes a high-accuracy camera calibration by carefully filtering out outliers and then averaging the results of estimated good matches. In contrast to AutoCalib [10, 11], we study different properties of accurate calibrations to design calibration filters (Section 4.7). A comparison study of these filters with statistical-based filtering algorithms are presented. Calibration techniques when there are vehicle-to-infrastructure communication links (V2I) are presented in Section 4.8.

We evaluate AutoCalib on 10 traffic camera video feeds that vary in camera location, orientation, and lighting conditions. We show that AutoCalib is able to produce a calibration with distance estimation errors of 12% or lower. This is in comparison to distance estimation errors of about 5% that are inevitable even using the manual calibration approach described earlier and prior techniques that result in errors as high as 56% (Section 6). Finally, a live demonstration of AutoCalib is available in Reference [8].

In summary, we make the following contributions:

- The first robust automatic calibration system for traffic cameras.
- A first-of-its-kind custom-trained DNN-based annotation tool that automatically annotates several key features of a car from low-resolution vehicular images.
- A novel filtering and aggregation algorithm that carefully refines and aggregates a large set of values obtained from the feature points identified by the annotation tool to produce an accurate calibration value with about 12% error or less.

2 BACKGROUND

In this section, we describe the pinhole camera model, provide some background on the camera calibration problem, and describe how calibration can be used in applications.

2.1 Camera Model

The *pinhole camera model* [27] describes the geometric relationship between the 2D image-plane (i.e., pixel positions in an image captured by a camera) and the 3D Ground Coordinate System (GCS), which is some fixed coordinate system in which real-world coordinates are measured.

¹The cameras are typically mounted to provide coverage over as large an area as possible.

Let the image plane be represented by the UV plane and the GCS be represented by the (XYZ) space. The following equation relates the pixel position (u, v) to its respective GCS coordinates (x, y, z) as

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad (1)$$

where f_x and f_y are the focal lengths of camera along x and y axes of the camera, respectively, and (c_x, c_y) represents the image center of the camera. $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ and $T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$ are the rotation matrix and the translation matrix of the camera, respectively.

2.2 Calibration Problem

Equation (1) can be written compactly in matrix-notation as follows:

$$sp_c = C[R|T]p_w. \quad (2)$$

The *camera calibration problem* is the problem of estimating C , R , and T . The camera matrix C depends only on the camera (and not on its position or orientation), while R and T represent the camera's position and orientation. Hence, C represents the intrinsic camera parameters, while R and T are referred to as the camera's extrinsic parameters. In our work, we focus only on the problem of estimating the camera's extrinsic parameters and assume that its intrinsic parameters are known.

Given n ($n \geq 4$) point-correspondences, consisting of both the real-world (GCS) coordinates as well as the corresponding image coordinates of n different points, as well as the camera's intrinsic parameters, we can solve for the camera's extrinsic parameters using Equation (1). This is known as the *perspective- n -point* problem. Several solutions exist for this problem [19] and an efficient $O(n)$ implementation is available in OpenCV (the open source computer vision library). We refer to this solution as *SolvePnP* and our auto-calibration makes use of this.

In this article, we automatically identify n point-correspondences from the image of a car in a traffic video and then use *SolvePnP* to compute the calibration parameters. Similarly, we compute calibrations for different car images and then use a novel filtering and aggregation algorithm that carefully combines these calibrations to produce a final calibration that is robust to errors such as in identifying the point-correspondences.

2.3 Using the Calibration

Once the camera parameters (C , R , and T) are known, Equation (2) can be used to transform between real-world (GCS) coordinates and image coordinates. However, the mapping from the GCS coordinates to the image coordinates is not a one-to-one function, as we map from three-dimensional coordinates to two-dimensional coordinates.

Given a point $P(u, v)$ in the image (the camera frame), Equation (2) allows us to map the point P to an infinite ray in the real world. Thus, to obtain the exact location of point P in GCS, we must have some extra information, such as one of its GCS coordinates.

Fortunately, this is not an issue for vehicular applications if we assume that the road in the region of interest is approximately flat. Then the feature-points in vehicles are at known heights from the ground plane. Using the knowledge of P 's height h_p , its 3D coordinates in GCS can be derived from Equation (1) by substituting z with h_p . Here, (x, y, h_p) denote the P 's coordinates in GCS. There are three unknowns, s , x , and y , which can be solved using three equations.

Now, consider the problem of estimating the speed of a car. We identify one of its feature-points, such as its tail-light, in two different frames at two different time instances t_1 and t_2 . We map the image of the tail-light to its GCS coordinate at t_1 and t_2 , allowing us to compute the car's speed between the two time instances. This can enable off-the-shelf traffic cameras to measure traffic speed without any dedicated sensors. A demo of vehicle speed measurement using the AutoCalib framework is available in Reference [8].

3 RELATED WORK

The problem of camera calibration has previously been studied in the literature. Two broad approaches to compute a static camera's calibration parameters are (i) using known geometric fixtures and (ii) using vanishing points.

Calibration using known geometric fixtures: Camera calibration can be performed from different static features present on the road such as road markings, width of the road, and electric poles. These features must be identified in the camera frame, and their coordinates in GCS need to be obtained. To compute the GCS coordinates, dimensions of fixed markers can be known from road marking standards or measured using Google Earth [5].

In traffic camera views, static features such as markers on the road and traffic signs are often occluded or unavailable. Moreover, the dimensions of static features may vary across roads depending on the city, mean traffic speed, and need for traffic delineation [9]. Thus, it is desirable to have minimal dependence on static scene features. Since vehicles are generally visible in any traffic camera, AutoCalib leverages vehicle key-points for camera calibration.

Rotation matrix using vanishing points: The set of parallel lines in the GCS when projected to the camera frame intersect at a unique point that is referred to as the *vanishing point*. The location of the vanishing point from parallel lines along each axis (X-, Y-, and Z-axes, respectively) derives the respective column values of the rotation matrix. In fact, vanishing points along two axes only are needed as two columns of the rotation matrix automatically determines the third column (Euler angles). Let the vanishing point observed along the X-axis be located at u_x, v_x . The first column of the rotation matrix can be derived by substituting $[1\ 0\ 0\ 0]^T$, u_x , and v_x in place of $[X\ Y\ Z\ 1]^T$, u , and v , respectively, in Equation (1), and so on. The translation matrix is then typically obtained using knowledge of real-world length of some geometric fixture in the image.

The closest work to ours is Reference [15], where authors assume a straight line of vehicle motion for computing the vanishing points and use known average sizes of vehicles (width, height, and length) to automatically calibrate the camera. References [14, 35] assume knowledge of the height of the camera. Other work, such as References [13, 32], assumes lane marking with known lane width, while [12] uses known mean vehicle size and known average vehicle speed, and Reference [24] uses knowledge of road lines.

While the vanishing points-based approaches reduce the manual labor of identifying points in the real world and their relative distances, they make several assumptions that make them less robust when employed in diverse settings. First, identifying lane markings or other geometric features automatically can be error prone. Second, assumptions such as straight-line motion of vehicles to derive vanishing points such as in References [14, 15] may be violated. For example, we observe that the derived vanishing points from References [14, 15] are not stable across different segments of a video, as vehicles may be changing lanes or at intersections where they may turn. Finally, none of these articles provide an automatic algorithm for filtering or aggregating calibrations.

4 DESIGN

Similarly to prior work, AutoCalib looks for feature points with known geometric shape. However, it does not rely on vanishing points. Further, in contrast to prior work, AutoCalib deals with errors

in the calibration due to vehicle detection and classification by filtering outliers and carefully fusing the remaining calibrations.

4.1 Challenges

We face the following key challenges in designing techniques for automatic traffic camera calibration:

- **Low-resolution images:** Infrastructure cameras are typically a mixture of low- and high-resolution cameras, depending on many factors such as when they were installed, budget and bandwidth considerations, and so on. Even with high-resolution cameras, the vehicle image forms only a small portion of the overall image, resulting in poor vehicle resolution. Low-resolution images lead to errors in *vehicle detection and classification*.
- **Detecting and annotating key-points:** We use points of interest such as taillights, side mirrors, and so on, which we refer to as key-points, to extract known geometric shapes for calibration. Automatically detecting and locating these points of interest from an image is a challenging task, since these points of interest are not standardized across different vehicles. The low resolution of images makes the detection process further challenging.
- **Consolidating a set of calibrations:** AutoCalib calibrates the infrastructure camera by utilizing the geometric shape of key-points in the image of a vehicle. The video stream from a traffic camera may contain a large number of vehicles that can result in a large set of calibration values. Some of the calibrations might be erroneous due to errors in classification of vehicles, errors in extracted geometry, and so on. Automatically filtering outliers and consolidating the remaining calibration values for deriving an accurate camera calibration value is a challenge.

4.2 Overview

Our pipeline for automatic calibration of traffic cameras consists of the following steps, as illustrated in Figure 1.

- (1) The input video frames are analyzed to detect instances of vehicles. AutoCalib combines traditional background-subtraction-based techniques and DNN based techniques to detect instances of vehicles, as explained in Section 4.3.
- (2) For each vehicle instance identified, AutoCalib crops the detected vehicle and identifies a set of key-points such as the centers of taillights and license plates. This vehicle key-point detection module is also DNN based and is described further in Section 4.4.
- (3) For each vehicle instance, the extracted key-points are used along with their respective dimensions from 10 popular car models to obtain a set of 10 calibrations of the camera using *SolvePnP* in Section 4.5.
- (4) The preceding steps produce a large set of calibrations. The calibrations suffer from several sources of errors present in the multiple steps of the pipeline. In the final stage, we apply a set of filtering techniques to eliminate outliers and apply averaging techniques to compute the final calibration in Section 4.6.
- (5) An accurate calibration must reproject feature point tracks belonging to the same vehicle to parallel lines segments of same length. This property is leveraged to design feature tracking-based filters in Section 4.7.
- (6) Vehicle to infrastructure (V2I) communication can be leveraged to derive the distance traversed by the vehicle across multiple frames. This is leveraged to design camera calibration techniques that are described in Section 4.8.

Fig. 1. AutoCalib pipeline for automatic calibration of traffic cameras.

- (7) AutoCalib can be extended for dashbord camera installations to support Advanced Driver Assistance Systems (ADAS) applications that are presented in Section 4.8.

4.3 Vehicle Detection

The goal of this step is to identify instances of vehicles (specifically, cars) in the frames of the input video. Previously proposed techniques for vehicle identification include (a) Background-subtraction-based techniques, (b) Haar-based classifiers, and (c) DNN-based techniques. Given the need for robustness across a wide range of camera resolutions, we combine Faster R-CNN [23], an efficient DNN-based approach for identifying objects, with a background-subtraction-based technique.

The background-subtraction-based approach is used to eliminate static bounding boxes (that do not change the position across successive frames). We then apply the DNN-based vehicle detection algorithm to detect vehicles. Whenever a successful match is found, we proceed to the next step in calibration computation. Further, we skip 5s of the video to avoid processing multiple images of the same vehicle and start looking for new vehicles.

4.4 Automatic Key-point Annotation

Once a vehicle is identified, the next step involves identifying and locating key-points of the vehicle in the image. We built a custom-trained DNN for this task by leveraging *transfer learning* [18] in order to reduce the training data and DNN training time.

Designing a DNN for vehicular key-point detection from scratch requires a large volume of annotated data. The idea behind transfer learning is to reuse a DNN that has been trained on a large generic dataset (e.g., ImageNet [17], that contains 1.2 million images with 1,000 categories). The pre-trained DNN is used either as initialization or as a fixed feature extractor for a more specific task of interest. This is motivated by the observation that the earlier layers of a DNN identify generic features of images (such as edges) that should be useful to many tasks. The later layers of the DNN become progressively more specific to the task of interest.

In our work, we reuse the CompCars DNN [33] for transfer learning. The CompCars DNN was built for classifying car models and was itself transfer learned from the imagenet dataset, with further training on CompCars' own dataset of 136,727 car images.

Dataset for transfer learning: For our task of identifying the key-points, we change the final layer of CompCars DNN from 431 Softmax outputs to 12 regression outputs, ranging from 0 to 1. These 12 outputs are 6 pairs of (x, y) coordinates for our 6 points: left and right side mirrors, left and right taillights, center tail lamp, and license plate. We selected these six points as the key-points as they are easily distinguishable in low-resolution images and provide points spread out in three dimensions (which is critical for *SolvePnP*). Each regression output has a range from 0 to 1 representing the x and y coordinates normalized with respect to the width and height of the bounding box. For example, $x = 0.3$, $y = 0.5$ for a 300×500 car image indicates that the key-point is located at (90,250). For training, we manually annotated 486 rear images of cars. From each annotated image, we produced 24 images using the following transformations: (0) The image;

Fig. 2. Different image augmentations performed on an image to enlarge the training sample set.

Fig. 3. Sample annotations from the DNN.

(1) four random crops in random aspect ratios by scaling the image; (2) two random rotations by any angle between 0 and 360; and (3) five random moves (placing the car image at random locations in a new image). Similarly, the same operations can be performed on a horizontal mirror image to create a total of 24 images per manually annotated image. This gave us a total of 10,344 images for the transfer learning process. Figure 2 shows different augmentations performed by AutoCalib to create the training image set. Figure 3 shows example annotations performed by our annotation tool. Detailed evaluation of the annotation tool is in Section 6.

4.5 Vehicle Model-based Calibration

Once we locate the vehicle key-points, we need their real-world 3D coordinates (relative to each other and ground level) so that we can use *SolvePnP* to calibrate the camera.

One way to get this information is to identify the vehicle model (e.g., Honda Civic) from the image. We studied the state-of-the-art vehicle classifier (CompCars [33]) for identifying vehicle models from different traffic cameras. We observed that CompCars classifies the vehicles accurately when high-resolution images are provided. However, the classification accuracy was quite poor for the vehicle images we obtained from the traffic cameras, possibly because of the image resolution. This is not surprising, since identifying the car models from these images was futile even for the authors.

Given the challenges in identifying the car model, we use a simple heuristic of using the 10 most popular sedans as the likely candidate car models of the vehicle. Thus, each identified car image produces 10 different calibrations from these 10 sedan models: Chevrolet Cruze, Toyota Corolla, Toyota Camry, Toyota Prius, Honda Accord, Honda Civic, Volkswagen Jetta, BMW 320i, Audi A4,

Table 1. Table Listing Real-world Coordinates (in meters) of Different Key-points

Model	Left taillight (LL)	Right taillight (RL)	License plate center (LIC)
Chevrolet Cruze	(0, 0, 0.89)	(1.44, 0, 1.89)	(0.72, 0, 0.77)
Toyota Corolla	(0, 0, 0.86)	(1.45, 0, 0.86)	(0.72, 0, 0.80)
Toyota Camry	(0, 0, 0.88)	(1.35, 0, 0.88)	(0.68, 0, 0.76)
Toyota Prius	(0, 0, 0.90)	(1.29, 0, 0.90)	(0.64, 0, 0.81)
Honda Accord	(0, 0, 0.88)	(1.52, 0, 0.88)	(0.76, 0, 0.81)
Honda Civic	(0, 0, 0.87)	(1.38, 0, 0.87)	(0.69, 0, 0.79)
Volkswagen Jetta	(0, 0, 0.85)	(1.29, 0, 0.85)	(0.64, 0, 0.77)
BMW 320i	(0, 0, 0.83)	(1.33, 0, 0.83)	(0.67, 0, 0.79)
Audi A4	(0, 0, 0.87)	(1.37, 0, 0.87)	(0.68, 0, 0.82)
Nissan Altima	(0, 0, 0.88)	(1.52, 0, 0.88)	(0.76, 0, 0.81)
Model	Center Lamp (CL)	Left Side Mirror (SWL)	Right Side Mirror (SWR)
Chevrolet Cruze	(0.72, 0.43, 1.12)	(−0.15, 2.79, 1.02)	(1.59, 2.79, 1.02)
Toyota Corolla	(0.72, 0.36, 1.10)	(−0.17, 2.83, 1.03)	(1.61, 2.83, 1.03)
Toyota Camry	(0.68, 0.46, 1.07)	(−0.24, 3.02, 1.02)	(1.60, 3.02, 1.02)
Toyota Prius	(0.64, 0.21, 1.16)	(−0.26, 2.66, 0.99)	(1.55, 2.66, 0.99)
Honda Accord	(0.76, 0.38, 1.1)	(−0.10, 2.88, 1.06)	(1.63, 2.88, 1.06)
Honda Civic	(0.69, 0.35, 1.07)	(−0.11, 2.64, 1.04)	(1.49, 2.64, 1.04)
Volkswagen Jetta	(0.64, 0.37, 1.06)	(−0.25, 2.84, 1.10)	(1.54, 2.84, 2.10)
BMW 320i	(0.67, 0.38, 1.07)	(−0.24, 2.72, 1.08)	(1.57, 2.72, 1.08)
Audi A4	(0.68, 0.42, 1.11)	(−0.26, 2.78, 1.01)	(1.63, 2.78, 1.01)
Nissan Altima	(0.76, 0.38, 1.1)	(−0.10, 2.88, 1.06)	(1.63, 2.88, 1.06)

The origin is located on the ground plane (road) underneath the left-tail light. The X-axis is along the width of the vehicle from left to right, Y-axis along the length of the vehicle, Z-axis is perpendicular to the ground plane from bottom to top.

and Nissan Altima. Table 1 lists down the real-world coordinates of different feature points for popular sedan models. Though vehicle distributions may vary across countries, AutoCalib can be easily tailored to different countries and their most popular vehicle models.

4.6 Statistical Calibration Filters

The preceding steps produce a large number of calibrations, one for each vehicle instance identified and candidate car-model. We observed a wide variance between these calibrations. The following factors contribute to errors in the calibration:

- *Annotation errors*: The DNN-based annotation tool that identifies key-points in a vehicle image is quite precise but can still introduce an error of a few pixels. This is not surprising, since even human annotation of these key-points can vary by a few pixels. Unfortunately, even one or two pixel errors in key-point annotation is magnified by the subsequent calibration steps and result in significant errors in estimations made using the calibration.
- *Vehicle identification errors*: AutoCalib uses the dimensions of 10 popular vehicle models to calibrate the camera. However, the vehicle observed by the camera may not be from this set. Even if the observed vehicle belongs to these models, only 1 calibration of 10 produced by AutoCalib is accurate.

Fig. 4. Using geometries extracted from a vehicle, AutoCalib calibrates the camera w.r.t. the car specific GCS.

The Problem. We exploit various statistical filters to eliminate a large fraction of the calibrations (including outliers) and compute an average of the remaining calibrations. One of the key challenges here is the fact that the calibrations produced from different vehicle instances all use different ground coordinate systems (GCS)! This makes it hard to compute an “average” of multiple calibrations. Identifying the right attribute(s) to filter outliers is also a challenge.

The Intuition. The GCS of the calibration produced from a vehicle instance depend on the vehicle’s position and orientation. Specifically, the three axes of this GCS correspond to the three axes of the 3D bounding box of the vehicle, with the origin at the bottom left corner. We exploit the following observations to deal with the set of calibrations with differing GCS: (a) For our application, only the ground (X-Y) plane of each calibration matters, and (b) barring errors, the ground plane of all generated calibrations must agree with each other (that is, the X-Y plane must be the same even though the X and Y axes may not be the same). These observations hold true provided the road lies in a single plane.

Observation (a) follows from our explanation in Section 2.3: The calibration can be used to map a point p_i in the image to a real-world point p_r only if we know some real-world coordinate of p_r ; for vehicular applications, we use the height of point p_r above the road (ground plane) as a known coordinate to determine its other coordinates. As Figure 4 illustrates, any two calibrations with the same ground plane (XY plane) will be equivalent for mapping image-points with known z-values.

Hence, we use the X-Y plane of the GCS of the different calibrations to do filtering as well as averaging, as explained below.

Details. The pseudo-code is depicted in Algorithm 1. Each calibration is represented by the pair (R, T) of a rotation and translation matrix. The camera matrix is the same for all calibrations and can be ignored here. The rotation matrix R is a 3×3 matrix and T is a 3×1 vector. The third column of the rotation-matrix R represents the unit vector along the Z-axis (of its GCS) and, hence, determines the orientation of the X-Y plane. We will refer to this unit vector as the *orientation* of the calibration.

Two calibrations with the same orientation have parallel X-Y planes but not necessarily the same X-Y plane. We compute a metric that is a measure of the distance between such (parallel) X-Y planes as follows. We identify a region of the image as the “focus region” (for our purpose, the

ALGORITHM 1: AutoCalib Filtering and Averaging

input: Set of all calibrations $C = [(R^1, T^1) .. (R^n, T^n)]$
output: Calibration Estimate c_{est}

```

1 Function Main( $C$ ):
2    $p = \text{ComputeFocusRegionMidPoint}()$ 
3    $C_\theta = \text{OrientationFilter}(C, 75)$ 
4    $C_{\theta,D} = \text{DisplacementFilter}(C_\theta, 50, p)$ 
5    $C_{\theta,D,\theta} = \text{OrientationFilter}(C_{\theta,D}, 75)$ 
6    $\vec{z}_{avg} = \frac{\sum_{i=1}^n R_{*,3}^i}{n}, R \in C_{\theta,D,\theta}$ 
7    $R_{avg} = \text{ComputeAverageRotationMatrix}(\vec{z}_{avg})$ 
8   foreach  $(R^i, T^i) \in C_{\theta,D,\theta}$  do
9      $R^i = R_{avg}$ 
10   $D = \text{ComputeDisplacement}(C_{\theta,D,\theta}, p)$ 
11   $c_{est} = \text{Sort } C_{\theta,D,\theta} \text{ by } D \text{ and pick the median element}$ 
12  return  $c_{est}$ 

13 Function OrientationFilter( $C, n$ ):
14   $\vec{z}_{avg} = \frac{\sum_{i=1}^n R_{*,3}^i}{n}$ 
15  foreach  $(R^i, T^i) \in C$  do
16     $\vec{z}_i = R_{*,3}^i$ 
17     $\theta_i = \arccos\left(\frac{\vec{z}_i \cdot \vec{z}_{avg}}{\|\vec{z}_{avg}\| \|\vec{z}_i\|}\right)$ 
18   $C_\theta = \text{Sort } C \text{ by } \theta \text{ and pick the lowest } n\% \text{ values}$ 
19  return  $C_\theta$ 

20 Function ComputeDisplacement( $C, p$ ):
21  foreach  $(R^i, T^i) \in C$  do
22     $p_i = \text{ReprojectToGround}(p, R^i, T^i)$ 
23     $d_i = \text{DistanceToCamera}(p_i, R^i, T^i)$ 
24  return  $D$ 

25 Function DisplacementFilter( $C, n, p$ ):
26   $D = \text{ComputeDisplacement}(C, p)$ 
27   $C_D = \text{Sort } C \text{ by } D \text{ and pick the middle } n\% \text{ values}$ 
28  return  $C_D$ 

```

road, is defined as the region where cars are detected). Let p denote the center of this region. We use each calibration c_i to map the point p to the corresponding real-world point p_i in the ground plane and compute the distance d_i between the camera and point p_i . We define d_i as the *displacement* of the calibration c_i (line 1).

For two calibrations with the same orientation, the difference in their displacements is a measure of the distance between their (parallel) X-Y planes. In particular, they have the same X-Y plane if and only if their displacements are the same.

Filtering. Our filtering heuristic for eliminating outliers uses both orientation and displacement. Let $\vec{z}_i$ denote the orientation of a calibration c_i . Let $\vec{z}_{avg}$ denote the average of all $\vec{z}_i$. The angle between $\vec{z}_i$ and $\vec{z}_{avg}$ is a measure of how much the orientation of c_i deviates from the average

orientation. We retain only the top 75% of calibrations with the smallest deviation from average (line 2). Second, we compute the displacement of the remaining calibrations, and retain only the middle 50% of calibrations (line 3). Finally, we apply the orientation-based filtering again (to benefit from the effects of displacement-based filtering) (line 4).

Averaging. Finally, we compute the “average” of the remaining calibrations. We average the Z-axis unit vector across all filtered calibrations and compute two mutually orthogonal X and Y-axis unit vectors. We then compute the Rotation Matrix for these three unit vectors, which forms our (averaged) final Rotation Matrix for the calibration estimate (lines 7–9). Using this average Rotation Matrix, we re-compute the displacements for all filtered calibrations, and the median value provides us the Translation Vector for our calibration estimate (lines 10 and 11). We use the median rather than the mean because of the non-linear behavior of the *SolvePnP* procedure.

4.7 Feature Tracking-based Calibration Filters

In this section, we describe different properties of accurate calibrations and present additional filters for calibrations.

The following are the properties of the accurate calibrations:

- The parallel lines projections in the image plane, when reprojected to the real-world coordinates using the calibration, must have the same slope.
- The line segment (of same length) projections in the image plane, when reprojected to the real-world coordinates using the calibration must result in line segments of same lengths.

We exploit these two properties to design additional filters for the calibrations. Inaccuracies in derived relative orientation of camera coordinates w.r.t. the real-world coordinates will result in reprojected lines with different slopes. To exploit this two properties, we derive feature tracks belonging to a vehicle. These points move along the parallel tracks by a same distance. Reprojections of these points are analyzed to identify the accurate calibrations. Algorithm 2 presents these calibrations filters exploiting these properties.

For the sake of simplicity, let us assume the vehicles observed by the camera are moving in straight lines. The paths traced by different points belonging to a vehicle must be parallel w.r.t. each other. This property can be leveraged to identify accurate calibrations. AutoCalib employs feature tracking techniques to identify the good features to track across the frames (line 2). These features are tracked across multiple frames to derive feature tracks in the image plane (line 3). Stationary feature points are filtered out by observing their velocity in the image plane (line 6). The bounding boxes derived from the vehicle detection module (line 4) are used to group the feature points belonging to the same vehicle (line 5). Figure 5 shows an example of feature tracks grouped by exploiting the vehicle detection bounding box. Due to the feature tracking errors, we observed that the obtained feature tracks have variable lengths as some of the feature points are tracked throughout the motion of the vehicle while the others tracks are terminated at the intermediate frames. We filter such feature points based on their length of tracks (line 7). Often the vehicle might change lanes or take a turn at an intersection. In such scenarios, the path traced by the vehicle is not a straight line. We employ linear fitting techniques to identify feature points moving in straight lines and filter the rest (line 10). These feature point tracks are projected to the ground plane using the calibrations from the abovementioned step and their real-world tracks are derived (line 18).

The real-world feature tracks are used in two ways to identify the accurate calibrations. First, the slope of the feature tracks must be same as these lines must be parallel. Second, the distance traveled by the feature points belonging to the same vehicle must be same across the X-axis and the Y-axis.

ALGORITHM 2: Feature Tracking-based Calibration filters**Input:** Video Frames *frames***Output:** Tracks of feature points *L*

```

1 Function ComputeImageTracks(frames):
2   Features = ComputeFeaturesToTrack(frames)
3   Tracks = DeriveFeatureTracks(Features)
4   Cars = DetectVehicles(frames)
5   DynamicTracks = DynamicFeatures(Tracks)
6   GroupedTracks = GroupFeatures(DynamicTracks, Cars)
7   FilteredTracks = LengthFilters(GroupedTracks)
8   ImageLines =  $\phi$ 
9   for k  $\in$  FilteredTracks do
10    | m, c, err = LinearFit(k)
11    | if err <  $\epsilon$  then
12    |   add(m, c, start, end, CarID) to ImageLines
13   return ImageLines;

14 Function ComputeRealWorldTracks(R, T, ImageLines):
15   foreach line  $\in$  ImageLines do
16   |   RealWorldLinesi = ReprojectToGround(line, R, T);
17   return RealWorldLines;

18 Function SlopeFilter(ImageLines, C, Anglethreshold):
19   foreach (Ri, Ti)  $\in$  C do
20   |   RealWorldLines = ComputeRealWorldTracks(Ri, Ti, ImageLines);
21   |   Angles = GetAngle(RealWorldLines);
22   |   Deviation = (max(Angles) - min(Angles));
23   |   if Deviation < Anglethreshold then
24   |   |   add(Ri, Ci) to FilteredCalibrations;
25   return FilteredCalibrations;

26 Function DistanceXFilter(ImageLines, C, threshold):
27   FilteredCalibrations = C
28   foreach (Ri, Ti)  $\in$  C do
29   |   RealWorldLines = ComputeRealWorldTracks(Ri, Ti, ImageLines)
30   |   foreach CarID  $\in$  ImageLines do
31   |   |   VehicleLines = GetVehicleLines(CarID)
32   |   |   DistanceX = GetFeatureMovementsX(VehicleLines)
33   |   |   Deviation =  $\frac{(\max(\text{DistanceX}) - \min(\text{DistanceX}))}{\max(\text{DistanceX})}$ ;
34   |   |   if Deviation > threshold then
35   |   |   |   remove(Ri, Ci) to FilteredCalibrations
36   return FilteredCalibrations;

```

Fig. 5. AutoCalib employs feature tracking techniques and vehicle detection module to design feature tracking-based calibration filters.

Slope-based Filtering: The tracks of the feature points belonging to a vehicle must be parallel w.r.t. each other. To exploit this property, for each calibration belonging to the calibration dataset C , AutoCalib reprojects the feature point tracks in the image plane to the ground plane (line 20). These real-world tracks are analyzed to derive their slope and angle w.r.t. the X-axis of real-world coordinates (line 21). The difference between maximum and minimum of the angles is used as a metric for filtering the erroneous calibrations.

Distance-based Filtering: The distance traversed by the feature points belonging to the same vehicle must be same. To leverage this property, AutoCalib projects the feature tracks in the image using the calibrations from the calibration dataset C . AutoCalib derives the distance traveled by the points belonging to the same vehicle (line 32). The percentage difference between maximum and minimum is used as a metric to identify accurate calibrations and filter the erroneous calibrations (line 35). Note the above assumption of the same velocity for all the points belonging to the vehicle will not be true if the vehicle is undergoing a rotatory motion (instances such as taking a left turn, or right turn). But such scenarios are filtered by observing the linear fitting errors while deriving the feature tracks.

Additional filters: The sign of the distance traveled by the vehicle along the Y-axis can be used as an additional filter. The distance traveled along the Y-axis is positive, if the vehicles are moving away from the traffic camera, since they are moving along the Y-axis. The distance traveled by the feature point along the X-axis should be less than the width of the road. This property can also be used to filter erroneous calibrations.

Limitations of this filters: The above approach assumes the height of the feature point to re-project the points from the camera frame to the ground plane. This height is approximated by the average height of the vehicle points (approximately height of the taillights). Due to this assumption, there will be re-projection errors that will affect the performance of filters. This assumption can be relaxed by tracking annotations. Taillights and side-mirrors belonging to the same vehicle can be tracked across multiple frames whose height is known, thereby reducing the re-projection errors.

4.8 Alternate Techniques and Extensions

Calibration from vehicle to infrastructure communication: AutoCalib uses the selected key points from the vehicles to derive a large set of calibrations. It uses filtering and aggregation algorithm to automatically produce a robust estimate of the camera calibration. This technique can be extended for the cases where there is additional information from the vehicles. Messages from vehicle to infrastructure communication (V2I) can be leveraged to calibrate the traffic cameras.

Vehicles can broadcast information such as the distance between their taillights, speed of the vehicle to the infrastructure cameras. The taillights can be easily identified by the annotation tool or by red-thresholding techniques. The centers of these lights can be tracked across multiple frames and their real-world coordinates can be derived from the V2I messages. The tracks of the lights from frame analysis and their real-world coordinates are given as input to *solvePnP* for deriving camera calibration parameters. In contrast to AutoCalib, this approach is able to make use of key-point annotations at different locations.

Inter-vehicular distance for ADAS applications: Different Advanced Driver Assistance Systems (ADAS) such as, Automatic Cruise Control, Vehicle platooning, EV-Matching [20, 30, 31] can make use of information such as the distance from neighboring vehicles. Currently, this information is derived from expensive sensors such as RADAR and LIDAR. Dashboard camera sensors is an economical solution for deriving this information. However, camera is a two dimensional sensors making measuring 3D distances challenging. Recent attempts such as in Reference [34] derive this information by integrating a dashboard camera with a mirror set up to synthesize additional views and exploiting disparity-based depth estimation techniques. An alternate solution can exploit the key-point detection tool presented by AutoCalib to derive the distance from neighboring vehicles. The key-point annotations from the dashboard camera are given as input to *SolvePnP* along with the vehicular dimensions. Since the dashboard camera captures images of vehicles at a closer distance compared to the traffic cameras, the vehicular classification techniques can be leveraged to identify the make and model of the observed vehicles. The output of *SolvePnP* contains the relative translation and orientation of the neighboring vehicles that can be used by ADAS applications.

5 IMPLEMENTATION

The complete AutoCalib pipeline is implemented in about 6,300 lines of python code. All vector algebra operations are sped up using the NumPy library and OpenCV 3.2 is used for background subtraction and calibration computations. The DNNs are trained and deployed on the TensorFlow framework. To collect human annotated data for the DNN key-point detector, we built our own web-based crowd-sourcing tool on the Django web framework.

AutoCalib is deployed on Microsoft Azure, running on a VM powered by 24 logical CPU cores and 4 Tesla K80 GPUs, with a total of 224GB RAM. For a 1280×720 video frame, this deployment can detect vehicles in the frame in about 400ms. For every detected vehicle, detecting the key-points takes about 50ms, and computing a calibration takes another 0.3ms per model. With this setup, AutoCalib can process 24 hours of 720p traffic video and compute calibration estimates in about 144 minutes. Figure 6 depicts an example calibration produces by AutoCalib.

6 EVALUATION

In our evaluation, we analyze AutoCalib's performance by measuring the accuracy of the final calibration estimates. We also present micro-benchmarks and comparisons at various points in the AutoCalib pipeline to motivate our design decisions.

The Dataset. To evaluate AutoCalib, we collect a total of 350+ hours of video data from 10 public traffic cameras in Seattle, WA [7]. Resolutions of these cameras vary from 640×360 to 1280×720 pixels. Intrinsic parameters of these cameras are derived from their baseline calibrations as computed in Section 6.2.

6.1 Key-point Annotation Accuracy

AutoCalib leverages DNNs to identify key-points on cars and computes calibrations by matching these key-points to their corresponding real-world GCS coordinates. However, annotations from the DNN are prone to errors, which may result in incorrect calibrations. To analyze the DNN's

Fig. 6. Sample calibration estimate from AutoCalib. The green dots are the identified key-points on the car, and the red lines form a 30m x 30m virtual grid derived from these key-points.

performance, we split our human annotated cars dataset into training and test sets following standard practice in DNN evaluation. Because of the computationally intensive nature of DNN training, cross-validation is infeasible and not commonly used in practice [17, 26]. However, we employ Dropout [25] regularization on the fully connected layers in the network to prevent over-fitting.

The dataset used for training and testing the DNN is a collection of 486 car images, with pose metadata and annotations for 6 key-points crowdsourced from 10 humans. This dataset is split into two parts: 90% of the images are used to train the DNN and the remaining 10% are used to test its accuracy. In the testing phase, we present the DNN with the test images and compute the normalized prediction errors for each key-point. The normalized error is defined as

$$E_k^{norm} = \frac{\sqrt{(x_{p,k} - x_{h,k})^2 + (y_{p,k} - y_{h,k})^2}}{w_c}, \quad (3)$$

where E_k^{norm} is the normalized DNN error in annotating key-point k ; $x_{p,k}$ and $y_{p,k}$ represent the DNN predicted x and y coordinates for key-point k ; $x_{h,k}$ and $y_{h,k}$ represent the human annotated x and y coordinates for key-point k ; and w_c is the width of the car, defined as the distance in pixels between the human annotated left lamp (LL) and right lamp (RL) key-points. This metric is chosen because it represents the percentage of deviation in extracted geometry and is independent of both the car size in the image as well as the image resolution.

Figure 7(b) details the DNN performance as a CDF of normalized error for the six key-points described in Section 6.3. The Y-axis represents the count as a fraction of total number of images, while the X-axis is the normalized error. It can be seen that the median error is only 6% of the car width. However, the bottom 20% of all key-points have an error of more than 10%, which may affect the calibration accuracy for those image samples. To discard these poor annotations, AutoCalib utilizes filters and averaging techniques described in Section 4.6.

6.2 Ground Truth for Evaluation

Once the DNN annotates the key-points on the car, AutoCalib starts producing calibration estimates for each car detection. These calibrations are later filtered and averaged to produce one calibration estimate.

Since these cameras are uncalibrated and no ground-truth calibration is available to evaluate estimates from AutoCalib, we establish ground truth by manually calibrating all 10 cameras. To do this, we identified distinguishable key-points (for instance, trees, poles, and pedestrian crossings)

Fig. 7. (a) CDF of DNN annotation error for the six key-points; (b) CDF of normalized DNN annotation error for the six key-points. The DNN can annotate more than 40% of the points with less than 5% car-width normalized error.

Fig. 8. Example of GTKPs marked in a frame. These points are used to compute the ground-truth calibrations and the distance RMSE.

in the camera image and their corresponding real-world coordinates in a common coordinate system by visually inspecting the same location using Google Earth [5]. These key-points, referred to as Ground Truth Key-points (GTKP), provide us a correspondence between image points and real-world coordinates. This correspondence is used to compute reference ground-truth calibrations using *SolvePnP*. For each camera, we collect 10 or more such GTKPs. Figure 8 shows an example of some GTKPs.

We can compute the error for any calibration estimate by calculating the errors in the on-ground distance estimation. To do so, we follow the approach presented in Section 2.3. We *re-project* the GTKPs' 2D image points to 3D coordinates by plugging the calibration's rotation and translation vectors, GTKP's 2D image coordinates and one of the GTKP's x , y , or z 3D coordinates in Equation (1). Since we are interested in measuring distances in the X-Y plane, we fix the z value to the GTKP's 3D z coordinate. This provides us with the re-projected x and y coordinates in the calibration's coordinate system. However, these reprojected 3D coordinates cannot be directly compared with the GTKP's 3D coordinates, since they are defined in different coordinate systems. To compare them, we compute Euclidean distances between pairs of these reprojected 3D

Fig. 9. Accuracy of AutoCalib vs. ground-truth calibration estimates. The distance measurement errors for the ground-truth calibrations, which are indicative of the errors in GTKP annotation, have an average RMS error of 4.62% across all cameras. AutoCalib has an average error of 8.98%.

coordinates in the calibration's coordinate system and measure them against respective pairwise Euclidean distances between GTKP 3D coordinates in the ground coordinate system.

Thus, by analyzing errors in re-projected vs. real distance measurements, we can measure calibration accuracy. This idea forms the basis for our calibration accuracy evaluation metric. Let D^{reproj} be the set of distances between all possible pairs of GTKPs reprojected from 2D image points. Similarly, let D^{real} be the set of distances between all possible pairs of actual GTKPs. For each GTKP pair i , we can compute the normalized error in distance measurement ϵ_i^{norm} as

$$\epsilon_i^{norm} = \frac{d_i^{reproj} - d_i^{real}}{d_i^{real}}, \quad (4)$$

where d_i^{reproj} and d_i^{real} are the re-projected and real distances, respectively, for the pair i . We now define Root Mean Square Error (*RMSE*) for a calibration as

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\epsilon_i^{norm})^2}, \quad (5)$$

where N is the number of possible pairs of GTKPs. This *RMSE* metric provides us an estimate of the accuracy of the calibration.

The manual calibration that we performed for ground-truth estimation is also prone to two sources of errors: (a) Human annotation errors while visually matching points in camera image and Google Earth view and (b) Google Earth distance estimate errors. The errors in manual ground-truth calibrations can be estimated by computing the *RMSE* for the ground-truth calibrations. That is, had the GTKPs been annotated with no errors, the manual calibration computed using the GTKPs would re-project on to the exact same points and the *RMSE* would be zero. There are errors in GTKP annotation, these errors are also reflected in the *RMSEs* shown in Figure 9. Thus, the *RMSEs* for manual ground-truth calibrations can be treated as a benchmark for calibration performance.

6.3 Calibration Accuracy

AutoCalib Distance Measurement Performance: Figure 9 highlights the end-to-end performance of AutoCalib. *RMSEs* across cameras for calibration estimates from AutoCalib have an average of 8.98%, with a maximum error of 12.27%. Note that AutoCalib's calibration errors is just a

Fig. 10. Effect of car key-point choice on calibration results. Removing Left Side Mirror (SWL) and Right Side Mirror (SWR) key-points has a severe effect on the calibration RMSE.

few percent higher than the errors introduced during the manual ground-truth calibration process (average of 4.62%, maximum of 8.20%).

Effect of Car Key-point Choice: AutoCalib utilizes six key-points: Left and Right Taillights centers (LL and RL), License Plate center (LIC), Center Lamp (CL), and Left and Right Side Mirrors (SWL and SWR). These key-points are carefully chosen not only because of their visual distinctness and ease of detection but also because they improve calibration accuracy. To determine the best choice of key-points, we conducted an experiment where we mounted a camera at a known height in a constrained environment. In this scene, the ground-truth distances were accurately measured using a measuring tape. We then added a car with known dimensions in the scene and manually labelled multiple visually distinct key-points on the car. The camera was then calibrated using different combinations of these key-points. We discovered that selecting non-planar and well separated key-points improves calibration accuracy significantly. This is because picking only planar key-points does not provide *SolvePnP* sufficient information about all three dimensions, causing ambiguity in the unit vector for the dimension orthogonal to the plane.

The importance of non-planar key-points is depicted in Figure 10, where we compare the RMS Error CDF of all calibrations obtained from AutoCalib prior to filtering and averaging. On calibrating without SWL and SWR key-points, the curve worsens significantly, with 80% of the calibrations having more than 50% error. Taking the width of the car to be the X-axis, height to be the Z-axis and depth to be the Y-axis, SWL and SWR are the only two points that are sufficiently far in the Y-Z plane from the other points. Most of the other points (LL, RL, LIC, and CL) have very little variance in their Y coordinates, thus *SolvePnP* is unable to resolve the ambiguity in the Y-axis unit vector. However, removing CL and SWR key-points has only a small effect on the calibration accuracies, since the other side view mirror helps disambiguation. Figure 11 shows the importance of non-planar key-points for different cameras. Thus, our choice of key-points helps *SolvePnP* to have a reference point well separated in all three axes, and thus produce consistent estimates for the Y-axis unit vector.

Effect of Filtering Parameters: AutoCalib refines its set of calibrations by discarding potentially poor calibrations. Since AutoCalib has no information about the scene or the ground truth, it uses the outlier filters as defined in Section 4.6. Because these filters rely on the statistical properties of the calibration distribution, there is a tradeoff between aggressiveness and robustness - discarding too many calibrations may also discard the “good” calibrations, but being conservative in filtering might let the poor calibrations slip through. In Figure 12, we compare the non-filtered

Fig. 11. Effect of car key-point choice on calibration results for different cameras. Removing Left Side Mirror (SWL) and Right Side Mirror (SWR) key-points has a severe effect on the calibration RMSE.

Fig. 12. Comparing the effect of filtering parameters across different cameras. Aggressive filtering with lower cutoffs improves performance in few cameras but misses good calibration in other cameras.

Fig. 13. Number of detections required for precise calibration. Precision of the estimated calibration increases with detections, but the effect diminishes after 2,000 detections.

set of calibrations with our preferred conservative filtering approach and an aggressive filtering approach. As described in Section 4.6, AutoCalib applies three filters, an orientation-based filter, a displacement-based filter, followed again by an orientation filter. Both filter types have a percentile cutoff for filtering; changing this cutoff affects the aggressiveness of the filter. For our notation, we name Orientation Filter with top XX% cutoff as OrientXX and Displacement Filter with middle XX% cutoff as DispXX. Here, we compare combinations of Orient75 and Disp50 Filters (Filter Set 1) against Orient35 and Disp30 Filters (Filter Set 2). Filter Set 2 is more aggressive, since it cuts off 65–70% of the data.

As shown in the RMSE CDFs of Figure 12, Filter Set 1 is effective at discarding the poor calibrations while retaining the good ones. Filter Set 2, being more aggressive is able to discard a lot more poor calibrations but fails to preserve good calibrations in certain cameras (e.g., C4, C5, and C8). This hurts the subsequent averaging process, resulting in a poor final calibration estimate. Thus, AutoCalib uses the conservative Filter Set 1 for the filtering process.

Finally, we also found that our conservative filtering was robust to small changes in the percentile used (for e.g., retaining 80 percentile or 70 percentile instead of 75 percentile did not materially change the final accuracy; results not shown due to space constraints).

Number of Detections Required for Precise Calibration: AutoCalib refines the estimated calibration using multiple frames to derive more accurate calibration values. Figure 13 shows calibration estimate RMSE using different number of vehicle detections for camera C1 across 50 trials. Increasing the number of vehicle detections allows for more calibration possibilities, while the filters ensure that the poor calibrations from these added detections are discarded. This results in a more precise calibration output from AutoCalib after the filtering and averaging steps. From our empirical analysis, typical frames from traffic cameras can contain 10s of cars at peak hours, enabling AutoCalib to arrive at a precise estimate of the calibration within an hour.

Number of Vehicle Models: Since car models are difficult to identify in these traffic cameras, AutoCalib uses the most popular car models to compute multiple calibrations and later filters out the mismatches. Figure 14 compares the effect of number of vehicle models used—top 1 vs. top 5 vs. top 10 models—using the RMS distance measurement error of the estimated calibration. Having more car models improves the accuracy in nearly all cameras. This also quantifies the robustness of the calibration filters in picking the correct calibrations; despite having more mismatches as the number of calibration models increase, the filters are able to discard the poor calibrations.

Comparison with Vanishing Point-based Approaches: Prior work such as in Reference [14] assume straight-line motion of the vehicles to derive vanishing points for computing the rotation

Fig. 14. Effect of number of calibration models used on the RMS Error of the estimated calibration. Having more calibration models per detection results in higher accuracy of the estimated calibration.

Fig. 15. Stability of vanishing points derived using Reference [14].

matrix. However, we observed that the derived vanishing points using such an approach are not stable, i.e., their location varies over time. We computed vanishing points using the techniques presented in Reference [14] over multiple 10-min video sequences from a Seattle city traffic camera. Figure 15 shows the estimated vanishing point for different 10-min video sequences. As shown in the figure, the vanishing points are unstable due to vehicle turns, and non-linear motion of the vehicles. From the experiments, we observe that identifying the vanishing points is challenging when the vehicles are changing lanes or making turns or the traffic is changing direction in places such as intersections.

The authors of Reference [14] use these vanishing points to estimate the rotation matrix for the given calibration. Further, their approach does not estimate a translation matrix. Instead, they assume that the distribution of car models (and their dimensions) are known and use that information to compute a scale factor for measuring distances on the road. However, it is not clear how to translate the computed scale factor to a T matrix.

Tilt of the camera is computed as the angle between the Z-axis unit vectors of two calibrations. Taking the Ground Truth Calibration as reference, we compute the tilt of AutoCalib's estimated calibration and compare it with the tilt computed for Reference [14]. Figure 16 shows the tilt estimation accuracy of AutoCalib and vanishing point-based solutions [14]. Across all cameras, we find that AutoCalib's estimated calibrations have an average 2.04° of tilt error, whereas the vanishing points-based approach from Reference [14] has an average tilt error of 4.94° . It is

Fig. 16. Accuracy of distance estimates computed from vanishing points using [14] and manually provided ground-truth camera height vs. AutoCalib. Despite providing the ground-truth camera height, estimates from Reference [14] have a mean RMSE of 21.59%. AutoCalib assumes no prior information about the camera height and has a mean RMSE of 8.98%.

Fig. 17. Tilt estimation error using AutoCalib and using vanishing points-based approach from [14]. AutoCalib has an average tilt error of 2.04° , compared to 4.94° from Reference [14]. Tilt is the angle between the Z-axis unit vectors of two coordinate systems.

important to note that even a few degrees of tilt error can translate into large errors while measuring on-ground distances.

Nevertheless, to provide a point of comparison of errors that can arise in using a vanishing point-based approach, we calculate the T Matrix for the approach in Reference [14] by manually providing the height of the camera based on our ground-truth calibration of the camera. Recall that the R and T matrices for any camera calibration define an affine transform for 3D coordinates from the Ground Coordinate System (GCS) to the Camera Coordinate System (CCS, where the camera is at the origin). Given the height, the 3D coordinates of camera are known and thus we can compute a T Matrix for the given R matrix.

Figure 17 compares the distance RMS error from the vanishing point approach [14] with manually provided camera height and AutoCalib. AutoCalib estimates have lower RMS error across all cameras with an average RMS error of 8.98%, while estimates from Reference [14] have an average RMS error of 21.59% even when one of the key calibration parameters, camera height, has been provided based on ground-truth calibration.

Fig. 18. Tilt estimation error using feature tracking-based filters and statistical filters. The average tilt error of feature tracking-based filters is 6.4° that is higher than statistical filters 2.4° .

Feature tracking-based filters vs. statistical filters: We compare feature tracking-based filters to statistical filters for identifying accurate calibrations. We have used the feature tracking-based filters and eliminated the calibrations with an angular threshold of 2° . The calibrations that result in feature tracks with respective angles (w.r.t. X-axis) differing by more than 2° , are eliminated. The threshold for distance is set to 10%. Different feature point projections belonging to the same vehicle must be of the same length. If the difference between the maximum and minimum of the feature track lengths differ by more than 10%, then these calibrations are filtered out. The rotation matrices from the rest of the calibrations are averaged. This is used to derive the tilt of the ground plane. Figure 18 shows the accuracy of estimating the tilt of the ground plane using feature tracking-based filters and statistical filters for different cameras. Clearly, statistical filters are better than feature tracking-based filters for estimating the orientation of the ground plane.

7 CONCLUSION AND FUTURE WORK

In this article, we propose AutoCalib, a system for scalable, automatic calibration of traffic cameras. AutoCalib exploits deep learning to extract selected key-point features from car images in the video and uses a novel filtering and aggregation algorithm to automatically produce a robust estimate of the camera calibration parameters from just hundreds of samples. Using videos from real-world traffic cameras, we show that AutoCalib is able to estimate real-world distances with an error of less than 12% under a variety of conditions. This allows a range of applications to be built on the AutoCalib framework. One such demo application to measure vehicle speeds is published in Reference [8].

So far, we have trained our annotation tool to identify key-points from only rear images of cars. While this works for many traffic cameras, it will not work for cameras that are positioned such that they do not view the rear of the cars. We plan to address this as part of future work by training our DNN to identify key-points from side-facing and front-facing car images.

ACKNOWLEDGMENTS

This work was partially supported by NSF grant CNS 1618520.

REFERENCES

- [1] [n.d.]. Camera from different cities in United States. Retrieved from www.earthcam.com/.
- [2] [n.d.]. Camera installations in China. Retrieved from <http://blogs.wsj.com/chinarealtime/2014/06/05/one-legacy-of-tiananmen-chinas-100-million-surveillance-cameras/>.
- [3] [n.d.]. Camera installations in Delhi. Retrieved from <http://www.livemint.com/Politics/qNQJHWF25EBpVzFB0kKboN/Glitches-in-CCTV-cameras-blindside-Delhi.html>.

- [4] [n.d.]. Camera installations in Hyderabad. Retrieved from <http://timesofindia.indiatimes.com/city/hyderabad/Cops-hope-to-secure-Hyderabad-with-CCTV-grid/articleshow/45724251.cms>.
- [5] [n.d.]. Google Earth.
- [6] [n.d.]. Road crash statistics. Retrieved from <http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics>.
- [7] [n.d.]. Seattle.gov department of transportation. Retrieved from <http://web6.seattle.gov/travelers/>.
- [8] [n.d.]. TrafficCalib Web Demo. Retrieved from <http://tiny.cc/autocalib>.
- [9] 2009. *Manual on Uniform Traffic Control Devices*. U.S. Department of Transportation.
- [10] Romil Bhardwaj, Gopi Krishna Tummala, Ganesan Ramalingam, Ramachandran Ramjee, and Prasun Sinha. 2017. AutoCalib: Automatic traffic camera calibration at scale. In *Proceedings of the ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys'17)*. ACM.
- [11] Romil Bhardwaj, Gopi Krishna Tummala, Ganesan Ramalingam, Ramachandran Ramjee, and Prasun Sinha. 2017. Demo: AutoCalib: Automatic traffic camera calibration at scale. In *Proceedings of the ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys'17)*. ACM.
- [12] Daniel J. Dailey, Fritz W. Cathey, and Suree Pumrin. 2000. An algorithm to estimate mean traffic speed using uncalibrated cameras. *IEEE Trans. Intell. Trans. Syst.* 1, 2 (2000), 98–107.
- [13] Douglas N. Dawson and Stanley T. Birchfield. 2013. An energy minimization approach to automatic traffic camera calibration. *IEEE Trans. Intell. Trans. Syst.* 14, 3 (2013), 1095–1108.
- [14] Markéta Dubská, Adam Herout, Roman Juránek, and Jakub Sochor. 2015. Fully automatic roadside camera calibration for traffic surveillance. *IEEE Trans. Intell. Trans. Syst.* 16, 3 (2015), 1162–1171.
- [15] Markéta Dubská, Adam Herout, and Jakub Sochor. 2014. Automatic camera calibration for traffic understanding. In *Proceedings of the British Machine Vision Conference (BMVC'14)*.
- [16] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. 2017. Panoptes: Servicing multiple applications simultaneously using steerable cameras. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'17)*. ACM, 119–130.
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'12)*. 1097–1105.
- [18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
- [19] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. 2009. Epanp: An accurate o(n) solution to the pnp problem. *Int. J. Comput. Vis.* 81, 2 (2009), 155–166.
- [20] Dong Li, Zhixue Lu, Tarun Bansal, Erik Schilling, and Prasun Sinha. 2014. ForeSight: Mapping vehicles in visual domain and electronic domain. In *IEEE Conference on Computer Communications (INFOCOM'14)*. IEEE, 1995–2003.
- [21] Franz Loewenherz, Victor Bahl, and Yin Hai Wang. 2017. Video analytics towards vision zero. *Inst. Transport. Eng. J.* 87, 3 (2017), 25.
- [22] Karsten Muller, Aljoscha Smolic, Michael Drose, Patrick Voigt, and Thomas Wiegand. 2005. 3-D reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. *IEEE Trans. Circ. Syst. Vid. Technol.* 15, 4 (2005), 538–549.
- [23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'15)*. 91–99.
- [24] K.-T. Song and J.-C. Tai. 2006. Dynamic calibration of Pan&# 8211;Tilt&# 8211;Zoom cameras for traffic monitoring. *IEEE Trans. Syst. Man Cybernet. B* 36, 5 (2006), 1091–1103.
- [25] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.
- [26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9.
- [27] Richard Szeliski. 2010. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media.
- [28] Gopi Krishna Tummala, Derrick Cobb, Prasun Sinha, and Rajiv Ramnath. [n.d.]. Poster: Soft-swipe: Enabling high-accuracy pairing of vehicles to lanes using COTS technology. In *Proceedings of the ACM International Workshop on Connected and Automated Vehicle Mobility (CarSys'16)*. ACM.
- [29] Gopi Krishna Tummala, Derrick Ian Cobb, Prasun Sinha, and Rajiv Ramnath. 2016. Methods and apparatus for enabling mobile communication device based secure interaction from vehicles through motion signatures. US Patent App. 15/060,494.
- [30] Gopi Krishna Tummala, Dong Li, and Prasun Sinha. [n.d.]. RoadMap: Mapping vehicles to IP addresses using motion signatures. In *Proceedings of the ACM International Workshop on Connected and Automated Vehicle Mobility (CarSys'16)*.

- [31] Gopi Krishna Tummala, Dong Li, and Prasun Sinha. [n.d.]. Roadview: Live view of on-road vehicular information. In *Proceedings of IEEE International Conference on Sensing, Communication and Networking (SECON'17)*.
- [32] Kunfeng Wang, Hua Huang, Yuantao Li, and Fei-Yue Wang. 2007. Research on lane-marking line based camera calibration. In *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety 2007 (ICVES'07)*. IEEE, 1–6.
- [33] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2015. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 3973–3981.
- [34] Vivek Yenamandra, Akshay Uttama Nambi, Venkata Padmanabhan, Vishnu Navda, and Kannan Srinivasan. 2017. CamMirror: Single-camera-based distance estimation for physical analytics applications. In *Proceedings of the 4th International on Workshop on Physical Analytics*. ACM, 25–30.
- [35] Zhaoxiang Zhang, Tieniu Tan, Kaiqi Huang, and Yunhong Wang. 2013. Practical camera calibration from moving objects for traffic scene surveillance. *IEEE Trans. Circ. Syst. Vid. Technol.* 23, 3 (2013), 518–533.

Received January 2018; accepted March 2018