AutoCalib: Automatic Traffic Camera Calibration at Scale

Romil Bhardwaj*
Microsoft Research

Gopi Krishna Tummala*
The Ohio State University

Ganesan Ramalingam
Microsoft Research

Ramachandran Ramjee
Microsoft Research

Prasun Sinha
The Ohio State University

ABSTRACT

Emerging smart cities are typically equipped with thousands of outdoor cameras. However, these cameras are typically 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*, *city road planning*, etc. Unfortunately, robust camera calibration is a manual process today and is not scalable.

In this paper, 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 outputs 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%.

ACM Reference format:
Romil Bhardwaj, Gopi Krishna Tummala*, Ganesan Ramalingam, Ramachandran Ramjee, and Prasun Sinha. 2017. AutoCalib: Automatic Traffic Camera Calibration at Scale. In *Proceedings of The 4th ACM International Conference on Systems for Energy-Efficient Built Environments, Delft, The Netherlands, November 6-8, 2017 (BuildSys),* 10 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Capitalizing on the dramatic reduction in costs of sensing devices, many smart cities are deploying public cameras at large-scale [2, 3]. For example, according to one estimate, there are as many as 100 million public cameras in China [3]. In this paper, we focus on the subset of public cameras that are installed to observe road traffic, called *traffic cameras*.

Video feeds from many such public traffic cameras are freely available today [6]. 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 the traffic videos, thereby enabling development of several novel smart city applications: (1) *Road safety*: Nearly 1.3 million people die due to road accidents every year [5]. 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 [16]; (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.) [16]. This can help city officials in planning, for example, the introduction of special bike lanes. (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 3D-view [17] by fusing the observed images. Finally, automatic calibration equips cameras to steer themselves and provide various services [12].

In this paper, we design and implement a system called AutoCalib, that takes in a video snippet from a given traffic camera and automatically outputs its calibration. AutoCalib uses a custom-trained deep neural network (§ 4), to automatically determine the location of several carefully-selected key-points from the image of a car (e.g., tail lamps, 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 mismatches 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 which are orientation (represented by rotation matrix R) and position of the camera in real-world coordinates (T). In this paper, 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 which 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 [4], and utilizing these real-world coordinates to calculate the extrinsic parameters using a standard vision-based solver (§ 2). This process is error-prone and manual, requiring enormous human effort for calibrating millions of already installed cameras. Further, for advanced cameras

*Co-primary authors

BuildSys, November 6-8, 2017, Delft, The Netherlands

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 [6]).

While the problem of automatic traffic camera calibration has been studied (§ 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 assumes only 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 tail lights of a Honda Civic).

Advances in deep learning has resulted in high accuracy for image classification tasks [14]. Further, there exists pre-trained models for several image classification tasks. For example, CompCars [24] 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 leftmost image in Figure 1.[1] The resolution of the car image is thus of not sufficient 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, tail lights, etc., 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 [14] to build a custom DNN from the pre-trained CompCars DNN using only about 500 manually annotated images containing labels of car features. In § 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 candidates distances (e.g., one each for the top 10 popular car models). The challenge then is to find a camera calibration that results 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, etc. We thus develop a novel filtering and aggregation algorithm that outputs a high accuracy camera calibration by carefully filtering out outliers and then averaging the results of estimated good matches.

We evaluate AutoCalib on ten traffic camera video feeds that vary in camera location and 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% (§ 6). Finally, anonymous viewing of a live demonstration of AutoCalib is available at [1].

In summary, we make the following contributions:

- 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* [22] 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.

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) with 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 focal lengths of camera along x and y axis of 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

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

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 is referred to as 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 point-correspondences, consisting of both the real-world (GCS) coordinates as well as the corresponding image coordinates of $n \geq 4$ 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 [15] and an efficient $O(n)$ implementation is available in OpenCV (the open source computer vision library). We

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

refer to this solution as *SolvePnP* and our auto-calibration makes use of this.

In this paper, we automatically identify n point-correspondences from the image of a car in a traffic video and then use *SolvePnP* to compute a calibration. 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 3-dimensional coordinates to 2-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, in order 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 3-unknowns, s, x and y which can be solved using 3-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 points 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 published at [1].

3 RELATED WORK

The problem of camera calibration has previously been studied in 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, electric poles etc. 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 known from road marking standards or measured using Google Earth[4].

However, in traffic camera views, static features such as markers on the road and traffic signs are often occluded or not available. Moreover, the dimensions of static features may vary across roads depending on the city, mean traffic speed and need for traffic delineation [7]. 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 which is referred as the *vanishing point*. Location of the vanishing point from parallel lines along each axis (X, Y and, Z axis respectively) derives the respective columns 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 etc. 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 [11] where authors assume 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. [10, 25] assumes the height of the camera. Other work such as [9, 23] assume lane marking with known lane width while [8] uses known mean vehicle size and known average vehicle speed, and [19] uses knowledge of road lines.

While the vanishing points based approaches reduce the manual labour 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 [10, 11] may be violated. For example, we observe that the derived vanishing points from [10, 11] 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 papers provide an automatic algorithm for filtering or aggregating calibrations.

4 DESIGN

Similar 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, etc. 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 tail lamps, side mirrors, etc., which we refer

to as key-points, to extract known geometric shape 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 contains large number of vehicles and therefore 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, etc. Automatically filtering outliers and consolidating the remaining calibration values for deriving an accurate camera calibration values 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 Deep Neural Networks (DNN) based techniques to detect instances of vehicles, as explained in §4.3.
(2) For every vehicle instance identified, AutoCalib crops the detected vehicle and identifies a set of key-points such as the centers of tail-lights, license plates etc. This vehicle key-point detection module is also DNN-based and is described further in §4.4.
(3) For each vehicle instance, the extracted key-points are used along with their respective dimensions from ten popular car models to obtain a set of ten calibrations of the camera using *SolvePnP* in §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 §4.6.

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) Deep neural networks (DNN) based techniques. Given the need for robustness across a wide-range of camera resolutions, we combine Faster R-CNN [18], 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 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 5 seconds of 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* [14] 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 [13], which contains 1.2 million images with 1000 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 becomes progressively more specific to the task of interest.

In our work, we reuse the CompCars DNN [24] 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.
Data set 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 tail lamps, 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 300x500 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 produces 24 images using the following transformations: (0) The image; (1) 4 random crops in random aspect ratios by scaling the image; (2) 2 random rotations by any angle between 0 and 360; (3) 5 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 example annotations performed by our annotation tool. Detailed evaluation of the annotation tool is in § 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 art vehicle classifier (Compcars [24]) for identifying vehicle models from different traffic cameras. We observed, that Compcars classifies 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.

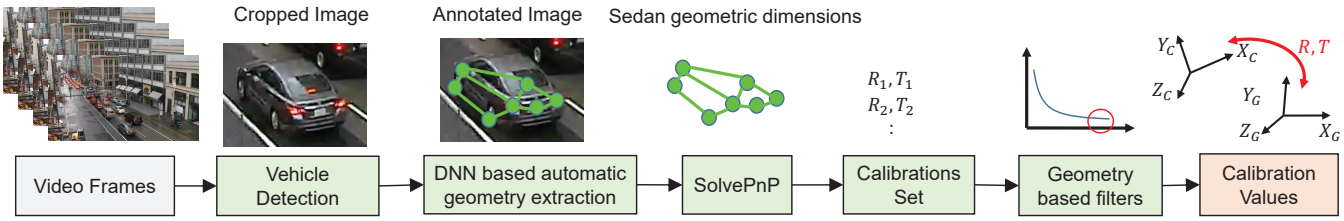


Figure 1: *AutoCalib pipeline for automatic calibration of traffic cameras.*

Figure 2: *Sample annotations from the DNN.*

Given the challenges in identifying the car model, we use a simple heuristic of using the ten most popular sedans as the likely candidate car models of the vehicle. Thus, every identified car image produces ten different calibrations from these ten sedan models: Chevrolet Cruze, Toyota Corolla, Toyota Camry, Toyota Prius, Honda Accord, Honda Civic, Volkswagen Jetta, BMW 320i, Audi A4 and Nissan Altima. Though vehicle distributions may vary across countries, AutoCalib can be easily tailored to different countries and their most popular vehicle models.

4.6 Filtering and Averaging Calibrations

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 keypoints 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 keypoints can vary by a few pixels. Unfortunately, even one or two pixel errors in keypoint 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 vehicles to calibrate the camera. However, the vehicle observed by the camera might not belong to one of the models it uses. Even if the observed vehicle belongs to these models, only one calibration out of 10 produced by AutoCalib is accurate.

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.

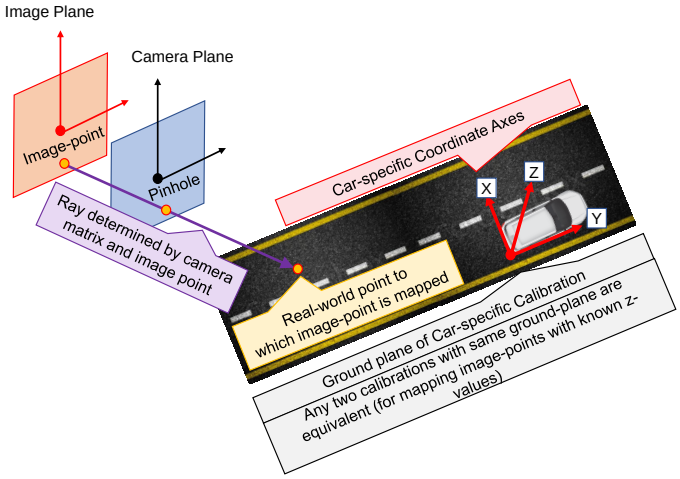


Figure 3: *Using geometries extracted from a vehicle, AutoCalib calibrates the camera w.r.t the car specific GCS.*

The Intuition. The GCS of the calibration produced from a vehicle instance depend on the vehicle position and orientation. Specifically, the 3 axes of this GCS correspond to the 3 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 §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 3 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

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 = `ComputeFocusRegionMidPoint()`
3 C_θ = `OrientationFilter(`$C, 75$`)`
4 $C_{\theta,D}$ = `DisplacementFilter(`$C_\theta, 50, p$`)`
5 $C_{\theta,D,\theta}$ = `OrientationFilter(`$C_{\theta,D}, 75$`)`
6 $\vec{z}_{avg} = \frac{\sum_{i=1}^{n} R^i_{*,3}}{n}, R \in C_{\theta,D,\theta}$
7 R_{avg} = `ComputeAverageRotationMatrix(`$\vec{z}_{avg}$`)`
8 **foreach** $(R^i, T^i) \in C_{\theta,D,\theta}$ **do**
9 $R^i = R_{avg}$
10 D = `ComputeDisplacement(`$C_{\theta,D,\theta}, p$`)`
11 c_{est} = Sort $C_{\theta,D,\theta}$ by D and pick the median element
12 **return** c_{est}
13 **Function** `OrientationFilter(`C, n`)`:
14 $\vec{z}_{avg} = \frac{\sum_{i=1}^{n} R^i_{*,3}}{n}$
15 **foreach** $(R^i, T^i) \in C$ **do**
16 $\vec{z}_i = R^i_{*,3}$
17 $\theta_i = \arccos\left(\frac{\vec{z}_i \cdot \vec{z}_{avg}}{\|\vec{z}_{avg}\| \|\vec{z}_i\|}\right)$
18 C_θ = Sort C by θ and pick the lowest n% values
19 **return** C_θ
20 **Function** `ComputeDisplacement(`C, p`)`:
21 **foreach** $(R^i, T^i) \in C$ **do**
22 p_i = `ReprojectToGround(`p, R^i, T^i`)`
23 d_i = `DistanceToCamera(`p_i, R^i, T^i`)`
24 **return** D
25 **Function** `DisplacementFilter(`C, n, p`)`:
26 D = `ComputeDisplacement(`C, p`)`
27 C_D = Sort C by D and pick the middle n% values
28 **return** C_D

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 road, defined as the regions 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 (line 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 (line 10–11). We use the median rather than the mean because of the non-linear behavior of the *SolvePnP* procedure.

5 IMPLEMENTATION

The complete AutoCalib pipeline is implemented in about 6300 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 the human annotated data for the DNN keypoint 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 224 GB RAM. For a 1280x720 video frame, this deployment can detect vehicles in the frame in about 400 milliseconds. For every detected vehicle, detecting the keypoints takes about 50 milliseconds, and computing a calibration takes another 0.3 milliseconds per model. With this setup, AutoCalib can process 24 hours of 720p traffic video and compute calibration estimates in about 144 minutes.

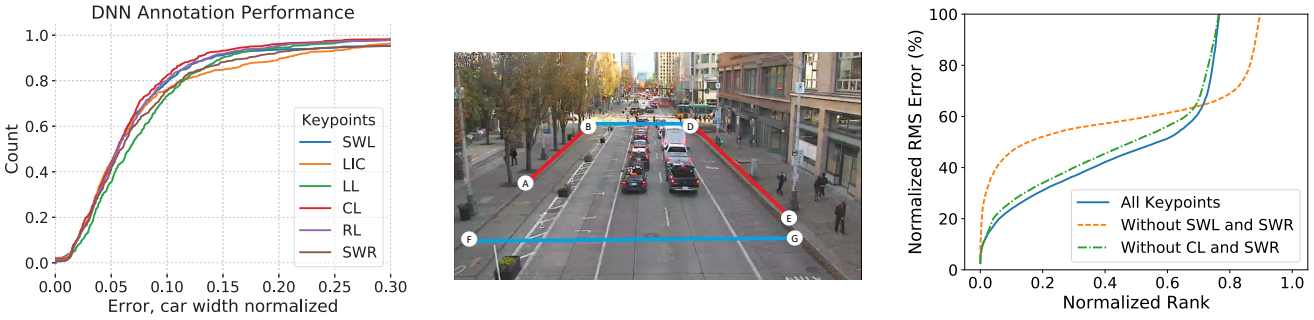
6 EVALUATION

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 [6]. Resolutions of these cameras vary from 640x360 to 1280x720 pixels. Intrinsic parameters of these cameras are derived from their baseline calibrations as computed in § 6.2.

6.1 Key-point Annotation Accuracy

AutoCalib leverages DNNs to identify keypoints on cars and computes calibrations by matching these key-points with 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 performance, we split our human annotated cars dataset into train and test sets following standard practice in DNN evaluation. Because of the computationally intensive nature



(a) (b) (c)

Figure 4: ***a) CDF of normalized DNN annotation error for the six keypoints. The DNN can annotate more than 40% of the points with less than 5% car-width normalized error. (b) Example of Ground Truth Keypoints (GTKPs) marked in a frame. These points are used to compute the ground truth calibrations and the distance RMSE. (c) Effect of car key-point choice on calibration results. Removing SWL and SWR keypoints has a severe effect on the calibration RMSE.***

of DNN training, cross-validation is infeasible and not commonly used in practice [13, 21]. However, we employ Dropout [20] 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 keypoints 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 4(a) details the DNN performance as a CDF of normalized error for the six keypoints described in §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 §4.6.

6.2 Ground Truth for Evaluation

Once the DNN annotates the key-points on the car, AutoCalib starts produces 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 ten cameras. In order to do this, we visually identifying distinguishable key-points (for instance, trees, poles and pedestrian crossings) in the camera image and their corresponding real world coordinates in a common coordinate system by visually inspecting the same location using Google Earth [4]. 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 4(b) 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 §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 coordinates in the calibration's coordinate system and measure them against respective pair-wise 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

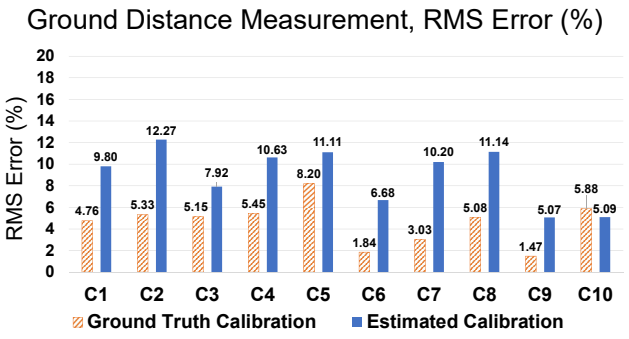


Figure 5: ***Accuracy of AutoCalib vs ground truth calibration estimates. The distance measurement errors for the ground truth calibrations, which are indicative of the errors in GTKP annotation, have an average RMS error of 4.62% across all cameras. AutoCalib has an average error of 8.98%.***

ϵ_i^{norm} as:

$$\epsilon_i^{norm} = \frac{d_i^{reproj} - d_i^{real}}{d_i^{real}} \tag{4}$$

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:

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

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

However, 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. However, since there are errors in GTKP annotation, these errors are also reflected in the *RMSE*s shown in Figure 5. Thus, the *RMSE*s for manual ground truth calibrations can be treated as a benchmark for calibration performance.

6.3 Calibration Accuracy

AutoCalib Distance Measurement Performance: Figure 5 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 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 Tail Lamp 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 4(c), where we compare the RMS Error CDF of all calibrations obtained from AutoCalib prior to filtering and averaging. On calibrating without SWL and SWR keypoints, 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 which 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. On the other hand, removing CL and SWR keypoints has only a small effect on the calibration accuracies, since the other side view mirror helps disambiguation. Thus, our choice of key-points helps *SolvePnP* to have a reference point well separated in all 3 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 §4.6. Because these filters rely on the statistical properties of the calibration distribution, there is a trade-off 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 6, we compare the non-filtered set of calibrations with our preferred conservative filtering approach and an aggressive filtering approach. As described in §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 6, 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, C8). This hurts the subsequent averaging process,

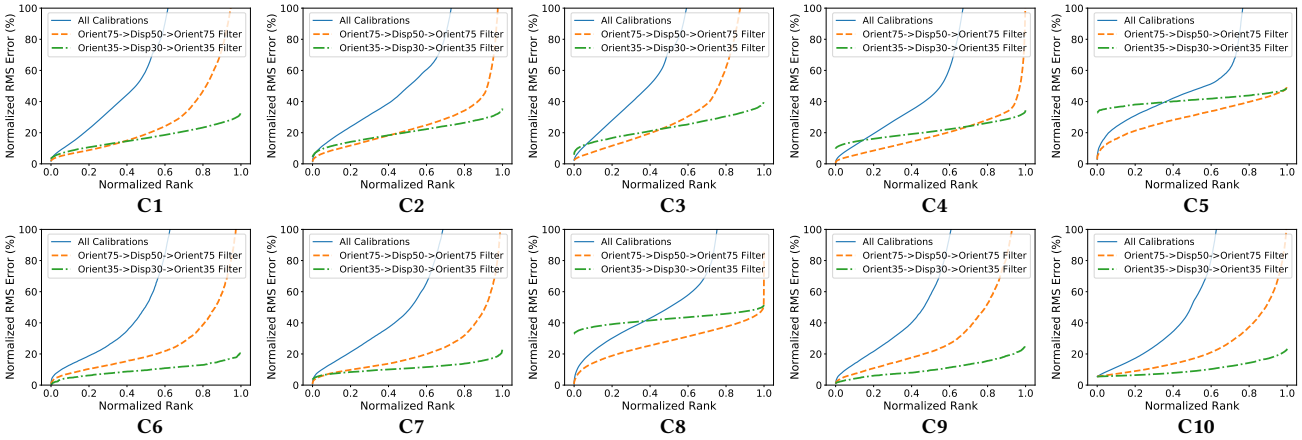


Figure 6: ***Comparing the effect of filtering parameters across different cameras. Aggressive filtering with lower cutoffs improves performance in few cameras but misses good calibration in other cameras.***

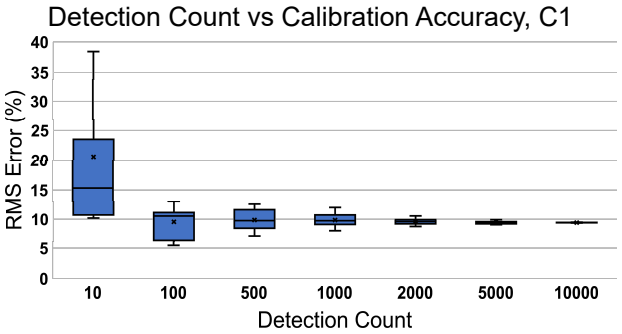


Figure 7: ***Number of detections required for precise calibration. Precision of the estimated calibration increases with detections, but the effect diminishes after 2000 detections.***

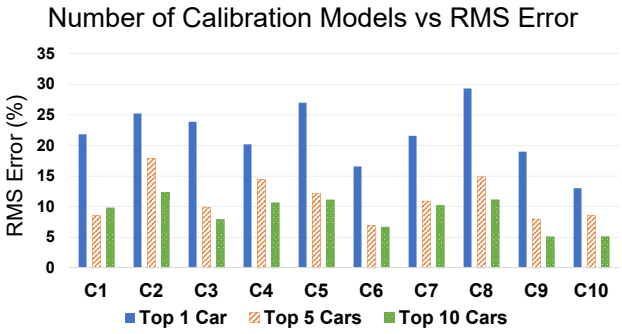


Figure 8: ***Effect of number of calibration models used on the RMS Error of the estimated calibration. Having more calibration models per detection results in higher accuracy of the estimated calibration.***

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 calibrations estimates using multiple frames to derive more accurate calibration values. Figure 7 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 tens 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 8 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 [10] assume straight line motion of the vehicles to derive vanishing points for computing the rotation 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 [10] over multiple 10 min video sequences from a Seattle city traffic camera. 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.

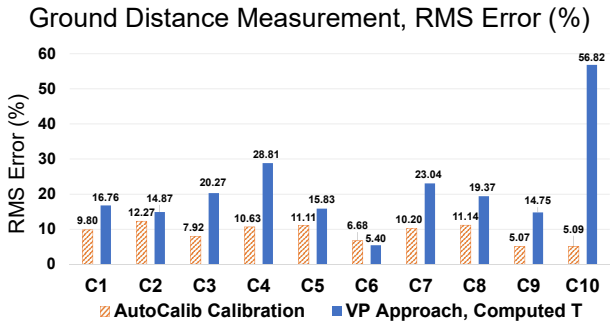


Figure 9: ***Accuracy of distance estimates computed from vanishing points using [10] and manually provided ground truth camera height vs AutoCalib. Despite providing the ground truth camera height, estimates from [10] have a mean RMSE of 21.59%. AutoCalib assumes no prior information about the camera height, and has a mean RMSE of 8.98%.***

Authors in [10] 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.

Nevertheless, in order 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 [10] approach 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 9 compares the distance RMS error from the vanishing point approach [10] 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 [10] 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.

7 CONCLUSION AND FUTURE WORK

In this paper, 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 at [1]

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 car rear. 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.

8 ACKNOWLEDGMENTS

We would like to thank our shepherd, Farrokh Jazizadeh, and the anonymous reviewers of BuildSys '17 for their comments and inputs towards improving the paper.

REFERENCES

[1] Autocalib web demo. http://tiny.cc/autocalib.
[2] Camera from different cities in United States. www.earthcam.com/.
[3] Camera installations in China. http://blogs.wsj.com/chinarealtime/2014/06/05/one-legacy-of-tiananmen-chinas-100-million-surveillance-cameras/.
[4] Google earth. https://www.google.com/intl/en_in/earth/.
[5] Road crash statistics. http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics.
[6] Seattle.gov department of transporation. http://web6.seattle.gov/travelers/.
[7] *Manual on Uniform Traffic Control Devices.* US Department of Transportation, 2009.
[8] Dailey, D. J., Cathey, F. W., and Pumrin, S. An algorithm to estimate mean traffic speed using uncalibrated cameras. *IEEE Transactions on Intelligent Transportation Systems 1*, 2 (2000), 98–107.
[9] Dawson, D. N., and Birchfield, S. T. An energy minimization approach to automatic traffic camera calibration. *IEEE Transactions on Intelligent Transportation Systems 14*, 3 (2013), 1095–1108.
[10] Dubská, M., Herout, A., Juránek, R., and Sochor, J. Fully automatic roadside camera calibration for traffic surveillance. *IEEE Transactions on Intelligent Transportation Systems 16*, 3 (2015), 1162–1171.
[11] Dubská, M., Herout, A., and Sochor, J. Automatic camera calibration for traffic understanding. In *BMVC* (2014).
[12] Jain, S., Nguyen, V., Gruteser, M., and Bahl, P. Panoptes: servicing multiple applications simultaneously using steerable cameras. In *Proc of IPSN* (2017), ACM, pp. 119–130.
[13] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS* (2012), pp. 1097–1105.
[14] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.
[15] Lepetit, V., Moreno-Noguer, F., and Fua, P. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision 81*, 2 (2009), 155–166.
[16] Loewenherz, F., Bahl, V., and Wang, Y. Video analytics towards vision zero. *Institute of Transportation Engineers. ITE Journal 87*, 3 (2017), 25.
[17] Muller, K., Smolic, A., Drose, M., Voigt, P., and Wiegand, T. 3-d reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. *IEEE Transactions on Circuits and Systems for Video Technology 15*, 4 (2005), 538–549.
[18] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS* (2015), pp. 91–99.
[19] Song, K.-T., and Tai, J.-C. Dynamic calibration of pan&# 8211; tilt&# 8211; zoom cameras for traffic monitoring. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 36*, 5 (2006), 1091–1103.
[20] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *JMLR 15*, 1 (2014), 1929–1958.
[21] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proc of CVPR* (2015), pp. 1–9.
[22] Szeliski, R. *Computer Vision: Algorithms and Applications.* Springer Science & Business Media, 2010.
[23] Wang, K., Huang, H., Li, Y., and Wang, F.-Y. Research on lane-marking line based camera calibration. In *Vehicular Electronics and Safety, 2007. ICVES. IEEE International Conference on* (2007), IEEE, pp. 1–6.
[24] Yang, L., Luo, P., Change Loy, C., and Tang, X. A large-scale car dataset for fine-grained categorization and verification. In *Proc of CVPR* (2015), pp. 3973–3981.
[25] Zhang, Z., Tan, T., Huang, K., and Wang, Y. Practical camera calibration from moving objects for traffic scene surveillance. *IEEE transactions on circuits and systems for video technology 23*, 3 (2013), 518–533.