Automatic Camera Calibration Techniques for Collaborative Vehicular Applications

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Gopi Krishna Tummala, B.Tech., M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2019

Dissertation Committee:
Dr. Prasun Sinha, Advisor
Dr. Rajiv Ramnath, Advisor
Dr. Kannan Srinivasan
© Copyright by
Gopi Krishna Tummala
2019
Abstract

In today’s age of rapidly evolving smart city infrastructure, several cutting-edge applications have found interest in the research community, such as traffic monitoring, accident prediction, and prevention. Due to their reduced cost and ease of integration with other hardware, cameras are an integral part of the sensing system in these applications. Calibrating these cameras enables measurement of real-world distances from the video, thereby opening the doorway for a wide range of novel applications. However, the current camera installations are typically not calibrated, i.e., information such as their precise mounting height and orientation is not available or involves a tedious manual process that requires a trained professional who needs to use a known pattern (e.g., chessboard-like) at a calibrated distance. In this proposal, I present the automatic calibration techniques for traffic/infrastructure cameras (AutoCalib) and dashboard cameras (DashCalib). We also propose two applications that make use of the calibrated camera sensors. Soft-Swipe runs on top of calibrated infrastructure cameras and exploits Vehicle to Infrastructure communications (V2I) to enable automatic pairing of vehicles with respective lanes. Road-View uses the information from dashboard cameras by fusing with Intervehicular communication (IVC) messages from neighboring vehicles to generate a global view of the road.

AutoCalib is a system for efficient, automatic calibration of traffic cameras. AutoCalib exploits deep learning to extract selected keypoint 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.

DashCalib is a system for automatic and live calibration of dashboard cameras that always ensures highly accurate calibration values. DashCalib exploits the motion of the vehicle, taillights from the neighboring vehicles, and GPS based odometry to derive a huge dataset of parallel lines along the length and width of the vehicles in the camera frame and uses a novel filtering and aggregation algorithm to automatically produce a robust estimate of the camera calibration parameters.

RoadMap is a system that matches IP addresses observed from Vehicle-to-Vehicle (V2V) communication with respective vehicles observed through a camera. It assumes a dashboard camera deployment and IVC capability of the vehicles. It gives reliable performance with commodity hardware and is designed to work in low adoption rate scenarios. RoadView builds the live map of surrounding vehicles by intelligently fusing the local maps created by individual vehicles.

Finally, the proposal discusses Soft-Swipe, a system for matching the vehicles with their respective lanes for enabling vehicle-based transaction services. Soft-Swipe works by matching motion signatures generated by the vehicle with the same signature detected by the infrastructure camera. It uses the information on the accuracy of different sensors in deriving motion trajectories and makes use of sensor fusion techniques to improve the accuracy of pairing. Soft-Swipe is implemented as an example application of pairing vehicles to respective quality check bays in Honda’s manufacturing plant.
Dedicated to my parents and brother
Acknowledgments

The time I spent at The Ohio State University is an important part of my life and I am thankful to many people who made this journey joyful. This dissertation would not have been possible without the advice, support, and patience of my advisers, Prof. Prasun Sinha and Prof. Rajiv Ramnath. They gave me much freedom and independence in exploring many interesting research topics. Their wisdom, acute sense, and unsurpassed knowledge always impressed me. I would like to thank all of my committee members for their time and tutelage.

I appreciate the suggestions and discussions I had with Derrick Cobb during my work with Honda, Ramjee Ramchandran, Ganesan Ramalingam, and Romil Bhardwaj during my internship at Microsoft Research and Ahmed Sadek, Avdhut Joshi and Arunandan Sharma, during my internship at Qualcomm research.

I owe gratitude to many great teachers from the CSE and Math Departments at Ohio-State. Particularly, I want to thank Prof. Kannan Srinivasan, Prof. Matthew Kahle, Prof. Ten H. Lai, Prof. Ness B. Shroff and Prof. Anish Arora. Their inspiring teaching helped me set up a solid background in both engineering and scientific fields.

I would like to thank my awesome colleagues, Dong Li, Tanmoy Das, Rupam Kundu, Dhrubo, Jiashang Liu, Wenjie, Yousi, Tarun Bansal, Ananya Mahanti, Alex, Vivek, Arjun and Bo Chen. Outside the lab, KG, Vivek, Sundeep, Surya, Kuldeep, Majety, Rahul and others have all been awesome friends with whom I’ll cherish many memories. Finally,
I would like to thank my father Rama Rao Tummala, my mother Giriaja Tummala and, my brother Hari Krishna Tummala. They were always there when I needed warmth and comfort.
Vita

May 2019 ................................. PhD,
Computer Science and Engineering,
The Ohio State University, USA.

Aug 2018 ................................. MS,
Computer Science and Engineering,
The Ohio State University, USA.

Jun 2012 ................................. B-Tech,
Electrical Engineering,
Indian Institute of Technology, Madras, India.

Aug 1991 ................................. Born - Nandivada, AP, India

Publications

Research Publications

Gopi Krishna Tummala, Romil Baradhwaj, Ganesan Ramalingam, Ramjee Ramchandran and Prasun Sinha, “AutoCalib: Automatic calibration of traffic cameras at scale.”, ACM Transactions on Sensor Networking (TOSN), March 2018

Gopi Krishna Tummala, Tanmoy Das, Prasun Sinha and Rajiv Ramnath “SmartDashCam: Automatic Live Calibration for DashCams”, Under Submission


Tanmoy Das, Gopi Krishna Tummala and Prasun Sinha, “TagTone: Scalable RFID Communication through Multi-Frequency Analysis”, IEEE MiSeNet 2018. (Best Paper Award)
Gopi Krishna Tummala, Romil Baradhwaj, Ganesan Ramalingam, Ramjee Ramchandran and Prasun Sinha, “AutoCalib: Automatic calibration of traffic cameras at scale.”, ACM BuildSys 2017 (Best Paper Award)


Gopi Krishna Tummala, Dong Li and Prasun Sinha, “Roadview: Live View of On-Road Vehicular Information”, IEEE SECON 2017


Gopi Krishna Tummala, Dong Li and Prasun Sinha, “RoadMap: Mapping Vehicles to IP Addresses using Motion Signatures”, ACM Carsys (MobiCom 2016 Workshop).


Gopi Krishna Tummala, Romil Baradhwaj, Ganesan Ramalingam, Ramjee Ramchandran and Prasun Sinha, “Demo: AutoCalib: Automatic calibration of traffic cameras at scale.”, ACM BuildSys 2017 (Best Demo Award)


**Fields of Study**

Major Field: Computer Science and Engineering
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xiv</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Proposed Problems to Study</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Contributions of Dissertation</td>
<td>6</td>
</tr>
<tr>
<td>1.2.1 Contributions of AutoCalib</td>
<td>7</td>
</tr>
<tr>
<td>1.2.2 Contributions of DashCalib</td>
<td>7</td>
</tr>
<tr>
<td>1.2.3 Contributions of RoadMap</td>
<td>8</td>
</tr>
<tr>
<td>1.2.4 Contributions of Roadview</td>
<td>9</td>
</tr>
<tr>
<td>1.2.5 Contributions of Soft-Swipe</td>
<td>10</td>
</tr>
<tr>
<td>1.3 Background</td>
<td>10</td>
</tr>
<tr>
<td>1.3.1 Pinhole Camera Model</td>
<td>11</td>
</tr>
<tr>
<td>1.3.2 Camera Calibration Problem</td>
<td>11</td>
</tr>
<tr>
<td>1.3.3 Using the Calibration for Vehicular Applications</td>
<td>12</td>
</tr>
<tr>
<td>1.3.4 Intrinsic Parameters Calibration</td>
<td>13</td>
</tr>
<tr>
<td>1.3.5 Extrinsic Parameters Calibration</td>
<td>14</td>
</tr>
<tr>
<td>1.3.6 Extrinsic Calibration Techniques for Traffic Cameras</td>
<td>17</td>
</tr>
<tr>
<td>1.4 Organization of the Dissertation</td>
<td>21</td>
</tr>
</tbody>
</table>
2. AutoCalib: Automatic Traffic Camera Calibration at Scale .......... 22
  2.1 Related Work .................................. 26
  2.2 Design ...................................... 28
    2.2.1 Challenges ................................ 28
    2.2.2 Overview .................................. 29
    2.2.3 Vehicle Detection ............................ 30
    2.2.4 Automatic Keypoint Annotation .................. 31
    2.2.5 Vehicle Model-based Calibration ................. 33
    2.2.6 Statistical calibration Filters ................... 34
    2.2.7 Feature tracking based calibration filters .......... 39
    2.2.8 Alternate techniques and extensions ............... 44
  2.3 Implementation .................................. 45
  2.4 Evaluation .................................... 46
    2.4.1 Keypoint Annotation Accuracy .................. 47
    2.4.2 Ground Truth for Evaluation .................... 48
    2.4.3 Calibration Accuracy ......................... 52
  2.5 Conclusion and Future work .......................... 63

3. DashCalib: Automatic Live Calibration for DashCams ............... 65
  3.1 Related Work .................................. 69
  3.2 Design ...................................... 71
    3.2.1 Overview .................................. 71
    3.2.2 Challenges ................................ 72
    3.2.3 Forward Vanishing point (FVP) .................. 72
    3.2.4 Lateral Vanishing point (LVP) ................... 75
    3.2.5 Estimating the Height of DashCam ................. 81
  3.3 Evaluation .................................... 87
    3.3.1 Accuracy of FVP ............................ 89
    3.3.2 Accuracy of LVP and Rotation Matrix .............. 91
    3.3.3 Accuracy of Estimated Height .................... 94
    3.3.4 Calibration Accuracy .......................... 95
  3.4 Conclusion .................................... 97

4. RoadMap: Mapping Vehicles to IP Addresses using Motion Signatures 98
  4.1 System Design .................................. 102
  4.2 Local Vehicle Matching ................................ 103
    4.2.1 Background ................................ 104
    4.2.2 The Local Matching Algorithm .................. 105
4.3 Experiments ................................................. 111
4.3.1 Vision Based Vehicle Detection .................... 111
4.3.2 The Proof-of-concept Implementation ............... 114
4.4 Simulations ................................................. 115
4.5 Related Work ................................................ 119

5. Roadview: Live View of On-Road Vehicular Information .......... 121
5.1 System Design .............................................. 125
5.1.1 Background on Graph Matching ..................... 126
5.1.2 Solution Overview of Roadview ....................... 126
5.2 Global Vehicle Merging ................................... 128
5.2.1 Motivation .............................................. 128
5.2.2 The Structures Used in GM ......................... 131
5.2.3 Creating the Association Graph .................... 131
5.2.4 The Global Merging (GM) Algorithm ................ 135
5.3 Simulations ................................................ 138
5.3.1 Evaluation of the GM Component .................. 140
5.4 Related Work ................................................ 142
5.5 Conclusion .................................................. 143

6. Soft-Swipe: Enabling High-Accuracy Pairing of Vehicles to Lanes using COTS Technology ............................... 144
6.1 System Overview ............................................ 148
6.2 Soft-Swipe Components .................................. 150
6.2.1 Automatic Calibration of Camera Exploring V2I links ...... 150
6.2.2 Sensor-Fence: Fine-Grained Motion Profiling with Array of Range Sensors ..................................... 152
6.2.3 Collaboration: Adaptive Weighted Vision & Sensing ...... 155
6.2.4 Weighted Matching of cross domain motion signatures .... 160
6.3 Implementation ............................................. 164
6.4 Evaluation .................................................. 167
6.5 Related Work ................................................ 172

7. Conclusion and Future Work .................................. 174
7.1 Calibration as a server-client application .................. 177
7.2 Automatic LIDAR and camera cross sensor calibration ......... 177

Bibliography ..................................................... 179
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Table listing real-world coordinates (in meters) of different keypoints. 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.</td>
</tr>
<tr>
<td>4.1</td>
<td>Different traffic scenarios in the simulation.</td>
</tr>
<tr>
<td>5.1</td>
<td>Different traffic scenarios in the simulation.</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The positioning map of different calibration techniques.</td>
</tr>
<tr>
<td>2.1</td>
<td>AutoCalib pipeline for automatic calibration of traffic cameras.</td>
</tr>
<tr>
<td>2.2</td>
<td>Sample annotations from the DNN.</td>
</tr>
<tr>
<td>2.3</td>
<td>Different image augmentations performed on an image to enlarge the training sample set.</td>
</tr>
<tr>
<td>2.4</td>
<td>Using geometries extracted from a vehicle, AutoCalib calibrates the camera w.r.t the car specific GCS.</td>
</tr>
<tr>
<td>2.5</td>
<td>AutoCalib employs feature tracking techniques and vehicle detection module to design feature tracking based calibration filters.</td>
</tr>
<tr>
<td>2.6</td>
<td>Sample calibration estimate from AutoCalib. The green dots are the identified keypoints on the car and the red lines form a 30m x 30m virtual grid derived from these keypoints.</td>
</tr>
<tr>
<td>2.7</td>
<td>a) CDF of DNN annotation error for the six keypoints; b) 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.</td>
</tr>
<tr>
<td>2.8</td>
<td>Example of Ground Truth Keypoints (GTKPs) marked in a frame. These points are used to compute the ground truth calibrations and the distance RMSE.</td>
</tr>
<tr>
<td>2.9</td>
<td>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%.</td>
</tr>
</tbody>
</table>
2.10 Effect of car keypoint choice on calibration results. Removing Left Side Mirror (SWL) and Right Side Mirror (SWR) keypoints has a severe effect on the calibration RMSE.

2.11 Effect of car keypoint choice on calibration results for different cameras. Removing Left Side Mirror (SWL) and Right Side Mirror (SWR) keypoints has a severe effect on the calibration RMSE.

2.12 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.

2.13 Number of detections required for precise calibration. Precision of the estimated calibration increases with detections, but the effect diminishes after 2000 detections.

2.14 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.

2.15 Tilt estimation error using AutoCalib and using vanishing points based approach from [75]. AutoCalib has an average tilt error of 2.04°, compared to 4.94° from [75]. Tilt is the angle between the Z axis unit vectors of two coordinate systems.

2.16 Accuracy of distance estimates computed from vanishing points using [75] and manually provided ground truth camera height vs AutoCalib. Despite providing the ground truth camera height, estimates from [75] have a mean RMSE of 21.59%. AutoCalib assumes no prior information about the camera height, and has a mean RMSE of 8.98%.

2.17 Stability of vanishing points derived using [75].

2.18 Tilt estimation error using feature tracking based filters and statistical filters. The average tilt error of feature tracking based filters is 6.4 degrees which is higher than statistical filters 2.4 degrees.

3.1 Manual calibration patterns used today [22].

3.2 DashCalib pipeline for automatic calibration of DashCams.
3.3 An example snapshot of DashCalib while estimating the FVP.  73

3.4 Pipeline for deriving forward vanishing point.  77

3.5 Lines joining the taillights, which are almost parallel to each other.  80

3.6 Technique from Algorithm 1 fails to estimate the LVP. The intersections of lines joining taillights are widely dispersed far from the Ground Truth.  82

3.7 Using velocity of the vehicle to estimate the height of the DashCam.  83

3.8 Depiction of CCS, VCS, and ACCS.  84

3.9 Reactiveness of mobile GPS and OBD  85

3.10 Accuracy of Mobile Phone and OBD-II considering SXBlue II as ground truth  86

3.11 Experiments for evaluating DashCalib are collected at different light conditions, orientations, and at random places on windshield of the DashCam.  87

3.12 Absolute error in estimating the FVP for different experiments (mean of 1.5% along x-axis and 1.4 % along y-axis).  89

3.13 Error in estimating the FVP with the number of frames processed.  90

3.14 Length of video processed for estimating the stable FVP (mean of 167 seconds).  90

3.15 Absolute errors in estimating pitch ($\alpha$, mean of 2.8 degrees), yaw ($\beta$, mean of 1.5 degrees), and roll ($\gamma$, mean of 1.2 degrees) for different experiments.  91

3.16 Absolute errors in estimating slope of LVP by observing lines joining taillights (mean of 0.09).  91

3.17 Absolute errors in estimating pitch ($\alpha$), yaw ($\beta$), and roll ($\gamma$) using IMU sensors and 5-point algorithm [90] based approach.  92

3.18 Absolute errors in height estimation by DashCalib (mean of 0.24m).  94
3.19 Absolute errors of manual calibration (mean of 4.1%), OnlyR (mean of 11.4%), and DashCalib (mean of 5.7%) for measuring distances along the road.

3.20 Absolute errors of manual calibration (mean of 4.1%), OnlyR (mean of 11.4%), DashCalib (mean of 5.7%), and 5-point based approach (mean of 38.0%) for measuring distances along the road.

4.1 (a) Traces of vehicle-i and vehicle-j associated with time-stamps to give unique identities in visual and electronic domain. The vehicle pair (i, j) are in different states over a span of 8 time-slots. (b) These traces are extracted as movement traces $T_i^e$ and $T_j^e$ in electronic domain and $T_i^v$ and $T_j^v$ in visual domains. Short-term noise and long-term trend can be observed in each time-series. The trace $T_i^v$ is more similar to $T_j^e$ at time slot-3, but long term trend of $T_i^v$ is more similar to $T_i^e$ which leads to correct match.

4.2 VID-EID matching using movement traces. When all the vehicles are identical their random movements give unique identity based on history.

4.3 Experiment result. LM outperforms Foresight with median precision improvement of 20% (7% higher F-Score than Foresight).

4.4 The simulation results of the LM algorithm.

5.1 The system architecture.

5.2 An example of correcting matching errors with collaboration. The circles in the dotted rectangle represent the relative locations and matching result of the three vehicles. Merging their matching results can correct C’s incorrect matching result.

5.3 An example of the GM algorithm. Initially vehicle C has three VIDs $\{c_1, c_2, c_3\}$, and $\mathcal{G}$ has five nodes. The red doted nodes and edges in Structure $\mathcal{M}$ and $\mathcal{G}$ indicate the same sub-structure shared by $\mathcal{M}$ and $\mathcal{G}$. In Step2, we assume only vertex-pairs $(C, g_2)$, $(c_1, g_4)$, $(c_1, g_5)$ and $(c_2, g_3)$ have similarities that satisfy the constraints in Algorithm 7. Therefore, only four nodes exist in the association graph $\mathcal{A}$. 

xvii
5.4 The number of vehicles sensed by different algorithms (average degree of
the reporter nodes in global structure $\mathcal{G}$). GM improves sensing by a factor
of 2. ................................................................. 141

6.1 Soft-Swipe architecture .............................................. 149

6.2 Sensor fence design. It provides: 1) Highly accurate shape and speed
estimation of vehicles; and, 2) Distinguishes very close-by vehicles. .... 154

6.3 Speed calibration from sensors: a) Two points A,B on the vehicle close
to each-other can be used to measure the slope of the plane. b) Speed of
vehicle is measured using this slope and rate of change of depth observed
by sensors. ................................................................. 154

6.4 Simulating sensor array with different angles. The higher the angle the
better the accuracy of slope estimation. .................................. 156

6.5 Figure representing data-flow while estimating weights for MMSE estima-
tion from history table. ................................................. 159

6.6 Camera speed estimation error variance is plotted with vehicle-position
from camera frame over 25 experiments. .................................. 160

6.7 Optical-flow vectors of a moving vehicle created by observing the motion
vectors of selected feature points. ........................................ 165

6.8 Sensor fence deployed with four ultrasonic sensors. ......................... 166

6.9 Speed estimation variance-plots of vision system (experiments) with average
standard deviation 1.6 kmph, sensor system (simulation and experiments)
with average variance of 2 kmph, and adaptive algorithm with average
variance of 1 kmph from Indoor low speed experiments. The adaptive weight
algorithm combines sensor-simulated results and vision experimental results
for estimating the motion profile and reduces the error by more than 50 %. 167

6.10 Motion profile from vehicular Electronic messages, sensor-system, vision-
system, and Adaptive-weight algorithm. ................................. 168

6.11 Weighted matching algorithm is evaluated for different metrics using vision-
only, sensor-fence, and Adaptive weight(AW) algorithm. ................. 171
7.1 (a) 360 degree view generated by Mercedes AMG S 63 [19]; (b) Camera placements for generating stitched top-view [8].

7.2 Different modules for providing calibration service.
Chapter 1: Introduction

Driven by plummeting hardware costs, many smart cities are installing large-scale camera networks [4, 5] for traffic monitoring and surveillance. According to one survey, there are as many as 100 million public cameras in China [5]. Cameras are also employed by toll-collection systems [38, 41, 43], parking space management systems [9], and traffic control systems [10, 15, 17, 32]. Video feeds from many public traffic cameras are freely available today [25].

Similarly, more vehicles are being equipped with cameras for enhancing Advanced Driver Assistance Systems (ADAS) [21, 34]. The US Department of Transportation issued a new rule requiring car manufacturers to include rear-view cameras in all cars manufactured after May 1, 2018 [121]. Meanwhile, smartphones, which are typically equipped with cameras, GPS, and radio interfaces, are available to more than 62.5% of the U.S. population [66]. Worldwide smartphone sales accounted for 55% of overall mobile phone sales in the third quarter of 2013 [81]. In vehicles, smartphones can be mounted on the dashboard to provide services such as navigation, over speed warning, and traffic alerts. These smartphones can be leveraged to communicate with neighboring vehicles Inter Vehicular Communication (IVC) and are equipped with cellular connectivity. Smartphone apps [26, 27] can transform a smartphone into a DashCam, providing video information along with information from sensors and IVC capability.
To facilitate building applications that need precise measurements from various video feeds, these cameras need to be calibrated. Calibration allows automatic estimation of real-world distances in the videos, thereby enabling development of several novel smart city applications. The IVC capability along with the relative vehicular localization from calibrated cameras can be explored to build cooperative vehicular safety applications [109]. To this end, we list the following applications enabled by the calibrated cameras and intervehicular communication as follows:

- **Road safety**: Nearly 1.3 million people die due to road accidents every year [24]. Calibrated traffic cameras can help in automatic estimation of vehicle speed, enabling always-on enforcement of road speed limits. Calibrated cameras can also estimate intervehicular and human-vehicle separations, which can help accident prediction and prevention applications [113].

- **City planning**: Calibrated cameras can help to generate automatic traffic reports such as the count of various vehicles (car, bus, bicycle) and their flow (speed, direction, turns, etc.) [113]. This can help city officials in planning, for example, the introduction of special bike lanes.

- **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 [120] by fusing the observed images. Finally, automatic calibration equips cameras to steer themselves and provide various services [94]. Views from multiple calibrated dashboard cameras can also be used to synthesize a bird’s eye view of the vehicle (a feature already available in some vehicles such as Audi [1] and Mercedes [20]).
1.1 Proposed Problems to Study

In this section we briefly introduce different application scenarios. Then we will propose the fundamental problems for enabling these applications. We explain these applications in the context of the user Lakshmi.

- **Traffic speeding**: Lakshmi drives an old Honda Civic and wants to use her smartphone for driver assistance. Lakshmi can view the traffic intersections on her route and get speed updates. She expects the traffic cameras to enforce safe driving habits for defensive driving.

- **Safe driving distance**: Lakshmi wants to know the safe following distance, measure the distances from neighboring vehicles and receive a warning beep if she is very close (unsafe distance) to other vehicles.

- **Smartphone-based toll payments**: When Lakshmi visits a toll-station or a parking meter, she wishes to pay the toll fee using her mobile phone or enable automatic transaction using her car so that she can just drive-by.

- **Vehicle Matching**: Lakshmi wants to find vehicles (say Alice’s car) going to the same destination that she is going to.

- **Collaborative vehicular applications**: Lakshmi wants to know the position and speed information of neighboring vehicles for safe driving. However, her smartphone can only observe vehicles ahead of her. Lakshmi needs to collaborate with neighboring vehicles to receive precise position and identities of her neighboring vehicles.

These applications require different sensors installed on Lakshmi’s vehicle, IVC capability, collaboration with neighboring vehicles, and services from traffic cameras. Enabling
these applications is challenging today as the DashCam app used by Lakshmi’s smartphone needs to be calibrated whenever she installs the phone. Traffic cameras must be calibrated in order to measure speeds of vehicles for enforcing the safe driving habits and providing speed updates.

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 thesis, we focus on estimation of the extrinsic parameters of 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 because it requires manual effort. When the cameras are installed, the primary objective is visibility of the scene. Physically measuring the 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, calibration is a tedious process at present. Further, different cameras require different types of calibration. These two calibration problems (traffic camera and dashboard camera) are of different nature depending on the type of the camera. Traffic cameras are static cameras, connected to a wired backbone, and can support heavy computational algorithms, whereas dashboard cameras are mobile cameras, they often lack the computational resources and require light-weight calibration solutions.

The third and fourth applications require lane-based pairing. GPS or other wireless location services are not accurate enough to distinguish vehicles that are closely spaced [110]. Camera and radar may not work correctly because of light conditions and blocking caused by neighboring vehicles. In the last application, Lakshmi needs to collaborate with other
vehicles to extend the sensing region of the camera. Toward this goal, this proposal studies
the following problems:

- **Traffic camera calibration**: Traffic 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 [16], and utilizing
  these real-world coordinates to calculate the extrinsic parameters using a standard
  vision-based solver. This process is error-prone and requires 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 [25]).

- **Dashboard camera calibration**: DashCams must be checked for calibration errors,
  and recalibrated continuously. Calibration errors will translate to catastrophic safety
  issues in different ADAS applications. New DashCam installation, changes in sus-
  pension of the vehicle, windshield installations, changes in tire pressure, installation
  of portable DashCams, and manual placement of smartphone based DashCams on
  the smartphone holder are some events that necessitate periodic recalibration of the
  dashboard camera. Continuous vehicular movements may also reorient the camera
  and possibly change its position. DashCams are calibrated today by positioning a
  chessboard-like pattern at a calibrated distance by a highly trained technician [30]
  (costs about 300-400 USD [11]).

- **Smartphone based payments**: For enabling smartphone-based pairing, vehicles
  need to be matched with their respective lanes in a multi-lane vehicular service station.
Cameras or infrastructure sensors can observe the presence of the vehicles, but they need to be matched with the electronic messages from the vehicles. This matching is challenging because the vehicles might be similar visually and the use of GPS or other wireless location services is inaccurate to distinguish vehicles that are closely spaced [110].

- **Matching vehicles observed by a DashCam:** Vehicles can employ vehicle detection, classification, and tracking techniques to observe neighboring vehicles. By exploiting the calibration, a vehicle can derive the relative location of neighboring vehicles. It can receive information of other vehicles over its radio. Without matching the vehicles between different devices, it is challenging, if not impossible, to combine and use the information provided by these devices. In addition, the presence of legacy vehicles makes it challenging to get high-precision vehicle matching results.

- **Extended sensing:** The detection results from different vehicles can be combined to improve the sensing range of vehicles and matching accuracy. The improved sensing region enables vehicles to observe neighboring vehicles that are not in the line of sight (NLoS). However, the errors in the individual matching results and presence of the legacy vehicles make fusing matching results from the individual vehicles challenging.

### 1.2 Contributions of Dissertation

This dissertation addresses the above challenges by designing the automatic calibration techniques for traffic/infrastructure cameras (AutoCalib) and dashboard cameras (DashCalib). It also presents two applications that can make use of the calibrated cameras. Soft-Swipe runs on top of calibrated infrastructure cameras and exploits Vehicle to Infrastructure communications (V2I) to enable automatic pairing of vehicles with respective lanes.
Road-View uses the information from dashboard cameras by fusing with IVC messages from neighboring vehicles to generate a global view of the road.

### 1.2.1 Contributions of AutoCalib

AutoCalib is a system for scalable, automatic calibration of traffic cameras. AutoCalib exploits deep learning to extract selected keypoint 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. AutoCalib is implemented 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%. The following are the contributions made by AutoCalib:

- First work to present automatic camera calibration techniques intended for infrastructure cameras by exploiting keypoints of the vehicles.

- Presents a novel Convolutional Neural Network (CNN)-based annotation tool that is built by employing transfer learning on a vehicle classifier CNN. This tool is first of its kind to automatically annotate the vehicular images for deriving the keypoints.

- Presents different metrics for refining a large set of calibration values obtained by analyzing a huge set of vehicular images.

### 1.2.2 Contributions of DashCalib

DashCalib is a system for automatic and live calibration of dashboard cameras that always ensures highly accurate calibration values. DashCalib leverages collecting images of a large number of vehicles appearing in front of the camera and using their coarse geometric
shapes to derive the calibration parameters. In sharp contrast to the manual calibration process, we are proposing the use of a large amount of data and machine learning techniques to arrive at calibration accuracies that are comparable to the manual process. DashCalib implemented using commodity dashboard cameras estimates real-world distances with mean errors of 5.7% which approaches the 4.1% mean error obtained from traditional manual calibration using known patterns. In summary, DashCalib makes the following contributions:

- Techniques to derive the rotation matrix by exploiting the relative motion and position of taillights of neighboring vehicles by smartly leveraging map information.

- Techniques to derive the height of the camera by comparing monocular visual odometry with GPS-based odometry.

- First robust automatic calibration system for dash cameras with mean distance estimation errors of 5.7%.

### 1.2.3 Contributions of RoadMap

RoadMap is a system that matches IVC messages with respective vehicles observed through a camera. It assumes a smartphone or a dashboard camera is deployed in vehicles, to identify the vehicles in field of view (FoV), and IVC capability. It runs in the adopted vehicles and accurately matches information obtained through multiple sensing modalities (e.g., visual and electronic). RoadMap matches the motion trajectories of vehicles observed from the dash-board camera with the motion trajectories transmitted by other vehicles. To the best of our knowledge, RoadMap is the first work to explore motion trajectories of vehicles observed from a camera to create a map of vehicles by smartly fusing electronic and visual information. It has low hardware requirement and is designed to work in low adoption
rate scenarios. Through real-world experiments and simulations, RoadMap matches IP-Addresses with camera-observed vehicles with a median matching precision of 80%, which is 20% improvement compared to existing schemes.

The contributions of this RoadMap include the following:

• We have designed a novel algorithm that determines the identities of the neighboring vehicles by exploring the movement pattern of the vehicles along with their visual features.

• We conducted a proof-of-concept experiment for the RoadMap system and observed median matching precision of 80% which is 20% higher than existing schemes.

• We simulated RoadMap with high-fidelity configurations. RoadMap simulated in different traffic scenarios and different system adoption rates outperformed existing schemes.

1.2.4 Contributions of Roadview

Roadview is a system that builds the live map of surrounding vehicles by intelligently fusing the local maps created by individual vehicles. Roadview runs on top of existing local vehicular matching systems (LM) such as Foresight [111] or RoadMap [167]. Roadview is the first work that provides a live map of vehicles by leveraging collaboration across vehicles. Our simulations show that for different adoption rates and traffic densities, Roadview can robustly fuse information from a collection of local maps and enhance vehicles to sense 1.8x (average) number of immediate neighboring vehicles compared to state of art LM algorithms.

The contributions of Roadview are as follows:
• First work to study the challenges involved in building a global information view of the road.

• Proposes a novel GM algorithm that enhances the capability of vehicles to sense 1.8x (average) more neighboring vehicles compared to state of the art LM algorithms. Note these neighboring vehicles may not be in FoV of vehicular sensors.

• Evaluates the system with extensive trace-driven simulations and different LMs.

1.2.5 Contributions of Soft-Swipe

Soft-Swipe works by matching natural signatures (specifically, motion signatures) generated by the target object with the same signature detected by simple instrumentation of the environment (a video camera or an inexpensive sensor array). To evaluate its performance, we consider an example application of pairing vehicles to respective quality check bays in an automobile manufacturing plant. Soft-Swipe implemented in a vehicle testing station performed pairing with median F-score of 96% using a vision-only system, 92% using a sensor-only system, and 99% using both.

Soft-Swipe makes the following contributions to the field:

• Presents automatic calibration techniques for infrastructure sensors such as camera and sensors array exploiting V2I links.

• Presents sensor fusion techniques by studying individual sensor characteristics

• Presents a configurable matching system with precision touching 100% for reliable financial transactions.

1.3 Background
In this section, we describe the pinhole camera model (§1.3.1), provide some background on the camera calibration problem (§1.3.2), and describe how calibration can be used in vehicular applications (§1.3.3). This section also presents background on intrinsic parameters (§1.3.4) and extrinsic parameters (§1.3.5) and different approaches to calibrate these parameters. §1.3.6 contrasts the calibration techniques presented in this thesis with the works in the literature.

1.3.1 Pinhole Camera Model

The pinhole camera model [159] 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 $(u, v)$ to its respective GCS coordinates $(x, y, z)$ as,

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

(1.1)

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.

1.3.2 Camera Calibration Problem

Equation 1.1 can be written compactly in matrix-notation as follows:

$$
sp_c = C[R|T]p_w.
$$

(1.2)
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.

### 1.3.3 Using the Calibration for Vehicular Applications

Once the camera parameters ($C$, $R$, and $T$) are known, Equation 1.2 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, because we map from 3-dimensional (3D) coordinates to 2-dimensional (2D) coordinates.

Given a point $P(u, v)$ in the image (the camera frame), Equation 1.2 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 1.1 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 taillight, in two different frames at two different time instances $t_1$ and $t_2$. We map the image of the taillight to its GCS coordinate at $t_1$ and $t_2$, and
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.

**The Direct Linear Transformation:** Equation 1.1 can be modified to measure the distances on the ground plane \((z = 0)\) as,

\[
\begin{bmatrix}
\frac{u}{s} \\
\frac{v}{s} \\
1
\end{bmatrix} =
\begin{bmatrix}
\frac{fx}{0} & \frac{cx}{0} & \frac{r_{11}}{r_{12}} & \frac{t_1}{1} \\
\frac{0}{fy} & \frac{cy}{0} & \frac{r_{21}}{r_{22}} & \frac{t_2}{1} \\
\frac{0}{0} & \frac{0}{1} & \frac{r_{31}}{r_{32}} & \frac{t_3}{1}
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}.
\]

(1.3)

The above equation can be modified to derive the \(x\) and \(y\) coordinates of the ground point \(P\) from respective its pixel position \((u, v)\) as,

\[
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix} =
\begin{bmatrix}
\frac{r_{11}}{r_{12}} & \frac{t_1}{1} \\
\frac{r_{21}}{r_{22}} & \frac{t_2}{1} \\
\frac{r_{31}}{r_{32}} & \frac{t_3}{1}
\end{bmatrix}^{-1}
\begin{bmatrix}
\frac{fx}{0} & \frac{cx}{0} & \frac{1}{1} \\
\frac{0}{fy} & \frac{cy}{0} & \frac{1}{1} \\
\frac{0}{0} & \frac{0}{1} & \frac{1}{1}
\end{bmatrix}
\begin{bmatrix}
\frac{u}{s} \\
\frac{v}{s} \\
1
\end{bmatrix}.
\]

(1.4)

The above equation can be written as,

\[
x = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + h_{33}} \quad \text{(1.5)}
\]

\[
y = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + h_{33}} \quad \text{(1.6)}
\]

Using the above equation, the distances on the ground can be measured for different vehicular applications.

### 1.3.4 Intrinsic Parameters Calibration

Intrinsic parameters are dependent on the camera and can be obtained from the specifications of the camera or calibrated by using a chessboard pattern-based procedures once prior to the camera usage. The distortion of the camera is more prevalent in 360-degree cameras and wide-angle cameras, which are referred to as fish eye cameras. Most of the commodity cameras such as smartphone cameras are devoid of the distortion effects. This distortion causes image points to be displaced in a nonlinear fashion from their ideal position in the
pinhole camera model. There are two common types of distortions due to the lens optical aberration.

1. **Barrel distortion**: The pixels are pushed outward due to this distortion. There will be more magnification in the center of the image compared to peripherals.

2. **Pincushion distortion**: The pixels are pushed inward due to this distortion. There will be more magnification in the peripherals compared to image center.

Both of these distortions can also happen at the same time, one type of distortion along the horizontal section and the other along the vertical section. However, radial distortion is observed most often. There are different models for articulating the distortions. Odd Polynomial Model [115, 150, 156, 162], Division Model [78], Polynomial Fish-Eye Transform [50, 145, 146, 169] as well as non-polynomial models Fish-Eye Transform [54], Field-of-View Model, and Perspective Model [93] are some models for the radial distortion. Odd Polynomial Model and Polynomial Fish-Eye Transform are popularly used for correcting the distortion of the camera. In addition to the distortion parameters, the focal length of the camera \( f_x, f_y \), optical center \( c_x, c_y \), and the skew factor \( k \) must be calibrated.

### 1.3.5 Extrinsic Parameters Calibration

In this section two different ways of solving the extrinsic parameters are presented. The first approach solves the extrinsic parameters by using the pixel positions of points in the camera frame and their respective real-world coordinates. The second approach solves the vanishing points along two axes to derive the rotation matrix between the two coordinate systems.
1.3.5.1 Perspective-n-point problem based approaches

Given \( n \) \((n \geq 4)\) point-correspondences, consisting of both the real-world coordinates and 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 (1.1). This is known as the perspective-n-point problem. Several solutions exist for this problem [108] and an efficient \( O(n) \) implementation is available in the open source computer vision library (OpenCV). We refer to this solution as SolvePnP and our auto-calibration makes use of this. Perspective-n-point problem deals with estimation of camera’s extrinsic parameters \((R\) and \(T\)) of the camera by using the points observed in the camera frame and their respective coordinates in the real world. Below are the solutions for two cases where the first solution uses three \( n = 3 \) points belonging to camera frame, later one is the solution for \( n \geq 4 \).

**Three point solution:** Consider three points \( A(u_1, v_1), B(u_2, v_2) \) and \( C(u_3, v_3) \) in camera frame with respective real world coordinates as \( A_g(x_1, y_1, z_1), B_g(x_2, y_2, z_2), \) and \( C_g(x_3, y_3, z_3) \) respectively. Let the camera be located at \( P(x, y, z) \). With the knowledge of the image points and real-world coordinates, first the distances \( PA_g = X, PB_g = Y, \) and \( PC_g = Z \) are solved using below equations.

\[
X^2 + Y^2 - 2XY \cos(\alpha) = (A_gB_g)^2 \tag{1.7}
\]
\[
X^2 + Z^2 - 2XZ \cos(\beta) = (A_gC_g)^2 \tag{1.8}
\]
\[
Y^2 + Z^2 - 2YZ \cos(\gamma) = (B_gC_g)^2 \tag{1.9}
\]

where \( \alpha, \beta, \) and \( \gamma \) are the angles between \((PA_g, PB_g), (PA_g, PC_g), \) and \((PB_g, PC_g) \)^1 respectively. With the obtained solution, the distance of the camera with respect to three points is obtained and this can be used to triangulate. Due to the quadratic nature of the

---

^1These angles can be solved using pixel positions and focal-length of the camera.
above set of equations, there will be four possible solutions of $R$ and $T$ matrices. More information about the solutions are presented in [80].

**N-point solution**: The N-point solutions considers $n$ ($n$-point set, $n \geq 4$) number of image points and their respective real-world coordinates, to solve the $R$ and $T$. A optimal implementation is presented in [108] and solves for coordinates of control points. Let $(u_1^c, v_1^c), (u_2^c, v_2^c), (u_3^c, v_3^c)$ and $(u_4^c, v_4^c)$ are four control points belonging to the real world coordinates be $(x_1^c, y_1^c, z_1^c), (x_2^c, y_2^c, z_2^c), (x_3^c, y_3^c, z_3^c)$ and $(x_4^c, y_4^c, z_4^c)$ respectively. Given a point $(x, y, z)$ in real world, it can be expressed as a linear combination of four control points. The same can be done with the points in image plane. Different points observed from the $n$-point set can be expressed as linear equations of control points, both in image plane and real world. Essentially, by solving the linear equations, N-Point solution solves for control points in image plane and real world. Using the solved locations of control points, the rotation and translation matrix are solved. An efficient $O(n)$ order implementation of N-point solution is available in OpenCV. Let us refer to this solution as SolvePnP where the $R$ and $T$ values are solved using more than three points. Using the above control-points approach is shown to be less effected by Gaussian noise in [80].

**1.3.5.2 Vanishing point based approaches**

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. 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 because two columns of the rotation matrix automatically determine 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...
$[X Y Z]^T$, $u$, and $v$, respectively, in Equation 1.1. The translation matrix is then typically obtained using knowledge of real-world length of some geometric fixture in the image.

### 1.3.6 Extrinsic Calibration Techniques for Traffic Cameras

Figure 1.1 depicts the positioning map of different classes of calibration techniques (for extrinsic parameters). The x-axis of the positioning map signifies the ease of calibration and the y-axis signifies the accuracy of the calibration. The techniques that have high accuracy and are easy to be employed are preferred for calibration needs. This thesis presents Soft-Swipe, DashCalib, and, AutoCalib to derive highly accurate calibration values and are easy
to employ. These techniques are highlighted with a yellow box under their labels. The following techniques are depicted in the positioning map:

- **Chessboard based techniques**: This is the naive way of calibrating both intrinsic and extrinsic camera parameters [135]. These techniques are based on [184] and can be performed using software such as MATLAB [59, 140] and OpenCV Library [159]. These techniques are highly accurate but need dedicated chessboard patterns for each camera installation. All the techniques based on this approach are shown with blue colored boxes in the Figure 1.1. Employing chessboard-based techniques for large scale traffic camera and DashCam calibrations is challenging because they require the user to take chessboard images for each camera installation.

- **Landmark based calibration techniques**: These techniques are similar to chessboard-based calibration techniques where they exploit the 3D coordinates of known landmarks or objects to calibrate a camera. Today, the traffic camera calibration is performed by identifying four or more landmarks in the image and manually measuring their real-world coordinates by using tools such as Google-maps [55]. These techniques are accurate but affected by measurement errors and image annotation errors. Employing these techniques at a large scale is challenging because they require the user to annotate images and manually measure distances between the identified landmarks.

- **Structure from Motion (SFM) and Map matching based calibration techniques**: These techniques are similar to landmark-based calibration techniques. Instead of using landmarks, they exploit the 3D map of the environment to derive the calibration parameters of a DashCam. Hanel et al. [89] generates a SFM [33] 3D map of a
calibration room instead of using calibration patterns. Generating such 3D maps is a laborious process involving data collection of large number of images [33]. Additionally, the vehicle needs to be driven to the calibration stations.

- **Road-markers based**: These techniques exploit the properties of the road to calibrate the camera. The height of the camera and the parallel property of the lane markers are exploited to derive the orientation of the DashCam in Catala et al. [62] and Nieto et al. [122]. The length and width of the lane markers are exploited to calibrate the DashCam in [86]. The lane boundary detection algorithms and the road width and speed of the vehicle are used to calibrate the DashCams [71, 134]. Grassi et al. [84] assumes the alignment of DashCam’s axes with the vehicle and exploits road-side markers (such as stop-signs) for estimating the height of the camera. Wang et al. [175] and Dawson et al. [69] assume lane marking with known lane width and Song et al. [152] uses knowledge of road lines for traffic camera calibration. The major drawback using the properties of the road for DashCam calibration is that the vehicle needs to be aligned with the road. Additionally, identifying lane markings or other geometric features automatically can be error-prone. 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 [35].

- **Vanishing point-based camera calibration techniques**: A set of parallel lines in the real-world coordinates when projected to the camera frame intersect at a unique point that is referred to as a **vanishing point**. The location of vanishing points is analyzed to derive the rotation matrix of camera coordinate system with respect to
the real-world coordinate system. Dubska et al. [76] assume straight-line vehicular motion for computing the vanishing points and use known average sizes of vehicles (width, height, and length) to automatically calibrate the traffic camera. Although the vanishing point-based approaches reduce the manual labor of identifying points in the real-world and their relative distances, they make several assumptions that render them less robust when employed in the diverse settings that we studied in [55].

- **IMU sensors-based calibration techniques**: The IMU sensors installed with the DashCam can also be exploited for calibration. The gravity vector’s direction, which is estimated in CCS, gives the direction of VCS’s Y-axis, which can be used to calibrate the DashCam.

- **Modeling-based calibration techniques**: These techniques are designed for calibrating the stereo-camera. Sappa et al. [138, 139] fits a plane (road) along the 3D point cloud generated by the stereo-camera and derives the orientation and height of the camera with respect to the road plane.

- **Communication-based**: The real-world coordinates of the calibration target can be obtained by communicating with it. Pixel positions of the calibration object along with the corresponding real world coordinates can be used to calibrate the camera. Our proposed solution (Soft-Swipe [163, 165]) exploits the V2I communication and uses a vehicle as a calibration object for infrastructure/traffic camera calibration. Zhang et al. [181] and Kuo et al. [105] exploit visual light communication to derive the position and orientation of the camera with respect to the visual lights bulbs.

- **Opportunity-based calibration techniques**: DashCalib analyzes the map information and the speed from GPS to identify the instances where accurate calibrations can be
achieved. It uses these opportunities to trigger different modules of the calibration pipeline to produce highly accurate calibration values.

- **Analysis-based calibration techniques**: AutoCalib [55] 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. Autocalib analyzes a number of calibrations derived from observing different vehicles to produce an accurate calibration value. Because the AutoCalib only depends on the images captured by the camera, it is easy to employ compared to calibration techniques designed in Soft-Swipe which depend on communication from the vehicle, or the techniques presented in DashCalib which depend on the GPS and navigation map information.

### 1.4 Organization of the Dissertation

This dissertation is organized as follows. Chapter 2 presents AutoCalib an automatic traffic camera calibration technique. Chapter 3 presents DashCalib an automatic calibration technique for dashboard cameras. Chapters 4 and 5 present RoadMap and RoadView systems. Chapter 6 presents Soft-Swipe. Finally, chapter 7 presents the conclusion and future work.
Capitalizing on the dramatic reduction in cost of sensing devices, many smart cities are deploying public cameras at large-scale [4–7]. For example, according to one estimate, there are as many as 100 million public cameras in China [5]. In this chapter, 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 [25]. 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 [24]. 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 [113]

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.) [113]. 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 [164, 165]

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 [120] by fusing the observed images. Finally, automatic calibration equips cameras to steer themselves and provide various services [94].

In this chapter, we design and implement a system called AutoCalib, that takes in a video snippet from a given traffic camera and automatically computes its calibration parameters. AutoCalib uses a custom-trained deep neural network (§ 3.2), 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 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 which are orientation (represented by rotation matrix $R$) and position of the camera in real-world coordinates ($T$). In this chapter, 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 [16], and utilizing these real-world coordinates to calculate the extrinsic parameters using a standard vision-based solver (SolvePnP). 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 [25]).

While the problem of automatic traffic camera calibration has been studied (§ 3.1), 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 [107]. Further, there exists pre-trained models for several image classification tasks. For example, CompCars [177] 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 2.1.² 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).

²The cameras are typically mounted to provide coverage over as large an area as possible.
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 [107] to build a custom DNN from the pre-trained CompCars DNN using only about 500 manually annotated images containing labels of car features. In § 3.3, 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 estimate the calibration parameters 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 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. We study different properties of accurate calibrations to design calibration filters (§ 2.2.7). 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 § 2.2.8.

We evaluate AutoCalib on ten 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% (§ 3.3). Finally, a live demonstration of AutoCalib is available at [2].

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.

2.1 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, 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 [16].

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 [35]. 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 to as the *vanishing point*. The 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 1.1 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 [76] 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. Dubská et al. [75] and Zhang et al. [182] assume knowledge of the height of the camera. Other work such by Wang et al. [175] and Dawson et al. [69] assume lane marking with known lane width while Dailey et al. [68] uses known mean vehicle size and known average vehicle speed, and Song et al. [152] 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 [75, 76] may be violated.
For example, we observe that the derived vanishing points from [75, 76] 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.

2.2 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.

2.2.1 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 keypoints:** We use points of interest such as tail lamps, side mirrors, etc., which we refer to as keypoints, 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 keypoints in the image of a vehicle. The video stream from a traffic camera may contain a large number of vehicles which 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 value is a challenge.

![AutoCalib pipeline for automatic calibration of traffic cameras.](image)

2.2.2 Overview

Our pipeline for automatic calibration of traffic cameras consists of the following steps, as illustrated in Figure 2.1.

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 §2.2.3.
2. For each vehicle instance identified, AutoCalib crops the detected vehicle and identifies a set of keypoints such as the centers of taillights and license plates. This vehicle keypoint detection module is also DNN-based and is described further in §2.2.4.

3. For each vehicle instance, the extracted keypoints are used along with their respective dimensions from ten popular car models to obtain a set of ten calibrations of the camera using \textit{SolvePnP} in §2.2.5.

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 §2.2.6.

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 §2.2.7.

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 which are described in § 2.2.8.

7. AutoCalib can be extended for dashboard camera installations to support Advanced Driver Assistance Systems (ADAS) applications which is presented in § 2.2.8.

\textbf{2.2.3 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 [133], 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.

2.2.4 Automatic Keypoint Annotation

Once a vehicle is identified, the next step involves identifying and locating keypoints of the vehicle in the image. We built a custom-trained DNN for this task by leveraging transfer learning [107] in order to reduce the training data and DNN training time.

Designing a DNN for vehicular keypoint 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 [103], 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 become progressively more specific to the task of interest.

In our work, we reuse the CompCars DNN [177] 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.
Figure 2.2: *Sample annotations from the DNN.*

Figure 2.3: *Different image augmentations performed on an image to enlarge the training sample set.*

**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 keypoints 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 keypoint 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; (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 2.3 shows different augmentations performed by AutoCalib to create the training image set. Figure 2.2 shows example annotations performed by our annotation tool. Detailed evaluation of the annotation tool is in § 3.3.

2.2.5 Vehicle Model-based Calibration

Once we locate the vehicle keypoints, 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 [177]) 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 ten most popular sedans as the likely candidate car models of the vehicle. Thus, each 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. Table 2.1 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.

2.2.6 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 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 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 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
<table>
<thead>
<tr>
<th>Model</th>
<th>Left taillight (LL)</th>
<th>Right taillight (RL)</th>
<th>License plate center (LIC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevrolet Cruze</td>
<td>(0, 0, 0.89)</td>
<td>(1.44, 0, 1.89)</td>
<td>(0.72, 0, 0.77)</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>(0, 0, 0.86)</td>
<td>(1.45, 0, 0.86)</td>
<td>(0.72, 0, 0.80)</td>
</tr>
<tr>
<td>Toyota Camry</td>
<td>(0, 0, 0.88)</td>
<td>(1.35, 0, 0.88)</td>
<td>(0.68, 0, 0.76)</td>
</tr>
<tr>
<td>Toyota Prius</td>
<td>(0, 0, 0.90)</td>
<td>(1.29, 0, 0.90)</td>
<td>(0.64, 0, 0.81)</td>
</tr>
<tr>
<td>Honda Accord</td>
<td>(0, 0, 0.88)</td>
<td>(1.52, 0, 0.88)</td>
<td>(0.76, 0, 0.81)</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>(0, 0, 0.87)</td>
<td>(1.38, 0, 0.87)</td>
<td>(0.69, 0, 0.79)</td>
</tr>
<tr>
<td>Volkswagen Jetta</td>
<td>(0, 0, 0.85)</td>
<td>(1.29, 0, 0.85)</td>
<td>(0.64, 0, 0.77)</td>
</tr>
<tr>
<td>BMW 320i</td>
<td>(0, 0, 0.83)</td>
<td>(1.33, 0, 0.83)</td>
<td>(0.67, 0, 0.79)</td>
</tr>
<tr>
<td>Audi A4</td>
<td>(0, 0, 0.87)</td>
<td>(1.37, 0, 0.87)</td>
<td>(0.68, 0, 0.82)</td>
</tr>
<tr>
<td>Nissan Altima</td>
<td>(0, 0, 0.88)</td>
<td>(1.52, 0, 0.88)</td>
<td>(0.76, 0, 0.81)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Center Lamp (CL)</th>
<th>Left Side Mirror (SWL)</th>
<th>Right Side Mirror (SWR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevrolet Cruze</td>
<td>(0.72, 0.43, 1.12)</td>
<td>(-0.15, 2.79, 1.02)</td>
<td>(1.59, 2.79, 1.02)</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>(0.72, 0.36, 1.10)</td>
<td>(-0.17, 2.83, 1.03)</td>
<td>(1.61, 2.83, 1.03)</td>
</tr>
<tr>
<td>Toyota Camry</td>
<td>(0.68, 0.46, 1.07)</td>
<td>(-0.24, 3.02, 1.02)</td>
<td>(1.60, 3.02, 1.02)</td>
</tr>
<tr>
<td>Toyota Prius</td>
<td>(0.64, 0.21, 1.16)</td>
<td>(-0.26, 2.66, 0.99)</td>
<td>(1.55, 2.66, 0.99)</td>
</tr>
<tr>
<td>Honda Accord</td>
<td>(0.76, 0.38, 1.1)</td>
<td>(-0.10, 2.88, 1.06)</td>
<td>(1.63, 2.88, 1.06)</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>(0.69, 0.35, 1.07)</td>
<td>(-0.11, 2.64, 1.04)</td>
<td>(1.49, 2.64, 1.04)</td>
</tr>
<tr>
<td>Volkswagen Jetta</td>
<td>(0.64, 0.37, 1.06)</td>
<td>(-0.25, 2.84, 1.10)</td>
<td>(1.54, 2.84, 2.10)</td>
</tr>
<tr>
<td>BMW 320i</td>
<td>(0.67, 0.38, 1.07)</td>
<td>(-0.24, 2.72, 1.08)</td>
<td>(1.57, 2.72, 1.08)</td>
</tr>
<tr>
<td>Audi A4</td>
<td>(0.68, 0.42, 1.11)</td>
<td>(-0.26, 2.78, 1.01)</td>
<td>(1.63, 2.78, 1.01)</td>
</tr>
<tr>
<td>Nissan Altima</td>
<td>(0.76, 0.38, 1.1)</td>
<td>(-0.10, 2.88, 1.06)</td>
<td>(1.63, 2.88, 1.06)</td>
</tr>
</tbody>
</table>

Table 2.1: Table listing real-world coordinates (in meters) of different keypoints. 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.
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 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 §1.3.3: 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 2.4 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 $3 \times 3$ matrix and $T$ is a $3 \times 1$ 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 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.
Algorithm 1 AutoCalib Filtering and Averaging

**input**: Set of all calibrations $C = [(R^1, T^1) .. (R^n, T^n)]$

**output**: Calibration Estimate $c_{est}$

**Function** Main($C$):

- $p = \text{ComputeFocusRegionMidPoint()}$
- $C_\theta = \text{OrientationFilter}(C, 75)$
- $C_{\theta, D} = \text{DisplacementFilter}(C_\theta, 50, p)$
- $C_{\theta, D, \theta} = \text{OrientationFilter}(C_{\theta, D}, 75)$
- $\bar{z}_{avg} = \frac{\sum_{i=1}^{n} R_i}{n}, R \in C_{\theta, D, \theta}$
- $R_{avg} = \text{ComputeAverageRotationMatrix}(\bar{z}_{avg})$
- $\text{foreach } (R^i, T^i) \in C_{\theta, D, \theta} \text{ do}$
  - $R^i = R_{avg}$
- $D = \text{ComputeDisplacement}(C_{\theta, D, \theta}, p)$
- $c_{est} = \text{Sort } C_{\theta, D, \theta} \text{ by } D \text{ and pick the median element}$
- **return** $c_{est}$

**Function** OrientationFilter($C, n$):

- $\bar{z}_{avg} = \frac{\sum_{i=1}^{n} R_i}{n}$
- $\text{foreach } (R^i, T^i) \in C \text{ do}$
  - $\bar{z}_i = R^i, 3$, $\theta_i = \arccos\left(\frac{\bar{z}_i \cdot \bar{z}_{avg}}{||\bar{z}_{avg}|| \cdot ||\bar{z}_i||}\right)$
- $C_\theta = \text{Sort } C \text{ by } \theta \text{ and pick the lowest } n\% \text{ values}$
- **return** $C_\theta$

**Function** ComputeDisplacement($C, p$):

- $\text{foreach } (R^i, T^i) \in C \text{ do}$
  - $p_i = \text{ReprojectToGround}(p, R^i, T^i)$
  - $d_i = \text{DistanceToCamera}(p_i, R^i, T^i)$
- **return** $D$

**Function** DisplacementFilter($C, n, p$):

- $D = \text{ComputeDisplacement}(C, p)$
- $C_D = \text{Sort } C \text{ by } D \text{ and pick the middle } n\% \text{ values}$
- **return** $C_D$

38
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 \( \vec{z}_{\text{avg}} \) denote the average of all \( \vec{z}_i \). The angle between \( \vec{z}_i \) and \( \vec{z}_{\text{avg}} \) 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 10–11). We use the median rather than the mean because of the non-linear behavior of the \texttt{SolvePnP} procedure.

2.2.7 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.
Figure 2.5: AutoCalib employs feature tracking techniques and vehicle detection module to design feature tracking based calibration filters.

- 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 4 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 2). The bounding boxes derived from the vehicle detection module (line 2) are used to group the feature points belonging to the same vehicle (line 2). Figure 2.5 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 2). 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 3). These feature point tracks are projected to the ground plane using the calibrations form the abovementioned step and their real-world tracks are derived (line 2).

The real-world feature tracks are used in two ways to identify the accurate calibrations. Firstly, 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 x-axis and y-axis.

**Slope-based Filtering:** The tracks of the feature points belonging to a vehicle must be parallel w.r.t each other. In order to exploit this property, for each calibration belonging to the calibration data-set $C$, AutoCalib reprojects the feature point tracks in the image plane to the ground plane (line 2). These real-world tracks are analyzed to derive their slope and angle w.r.t the x-axis of real-world coordinates (line 2). The difference between maximum and minimum of the angles is used as a metric for filtering the erroneous calibrations.
Algorithm 2: Feature Tracking based Calibration filters

**Input:** Video Frames $frames$

**Output:** Tracks of feature points $L$

**Function** `ComputeImageTracks(frames)`:

Features = `ComputeFeaturestoTrack(frames)`

Tracks = `DeriveFeatureTracks(Features)`

Cars = `DetectVehicles(frames)`

Dynamic Tracks = `DynamicFeatures(Tracks)`

Grouped Tracks = `GroupFeatures(DynamicTracks, Cars)`

Filtered Tracks = `LengthFilters(GroupedTracks)`

Image Lines = $\phi$

for $k \in $ Filtered Tracks do

$\begin{align*}
    m, c, err &= \text{LinearFit}(k) \\
    \text{if err} &< \varepsilon \text{ then} \\
    \quad \text{add} (m, c, \text{start}, \text{end}, \text{CarID}) \text{ to Image Lines}
\end{align*}$

end

return Image Lines

**Function** `ComputeRealWorldTracks(R, T, Image Lines)`:

foreach line $\in$ Image Lines do

Real World Lines$_{i}$ = `ReprojectToGround(line, R, T)`

end

return Real World Lines

**Function** `SlopeFilter(Image Lines, C, Angle threshold)`:

foreach $(R_{i}, T_{i}) \in C$ do

Real World Lines = `ComputeRealWorldTracks(R_{i}, T_{i}, Image Lines)`

Angles = `GetAngle(Real World Lines)`

Deviation = (max(Angles) - min(Angles))

if Deviation < Angle threshold then

add $(R_{i}, C_{i})$ to Filtered Calibrations

end

end

return Filtered Calibrations

**Function** `DistanceXFilter(Image Lines, C, threshold)`:

Filtered Calibrations = $C$

foreach $(R_{i}, T_{i}) \in C$ do

Real World Lines = `ComputeRealWorldTracks(R_{i}, T_{i}, Image Lines)`

foreach Car ID $\in$ Image Lines do

Vehicle Lines = `GetVehicleLines(Car ID)`

DistanceX = `GetFeatureMovementsX(Vehicle Lines)`

Deviation = $\frac{\max(\text{DistanceX}) - \min(\text{DistanceX})}{\max(\text{DistanceX})}$

if Deviation > threshold then

remove $(R_{i}, C_{i})$ to Filtered Calibrations

end

end

end

return Filtered 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 form the calibration data-set $C$. AutoCalib derives the distance traveled by the points belonging to the same vehicle (line 2). The percentage difference between maximum and minimum is used as a metric to identify accurate calibrations and filter the erroneous calibrations (line 2). 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 which 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.
2.2.8 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 \texttt{solvePnP} for deriving camera calibration parameters. In contrast to AutoCalib, this approach is able to make use of keypoint annotations at different locations.

Inter-vehicular distance for ADAS applications: Different Advanced Driver Assistance Systems (ADAS) such as Automatic Cruise Control (ACC), Vehicle platooning, EV-Matching (such as Foresight, RoadView, RoadMap etc.) 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 [178] derive this information by integrating dashboard camera with a mirror-set up to synthesize additional view and exploiting disparity based depth estimation techniques. An alternate solution can exploit the keypoint detection tool presented by AutoCalib to derive the distance from
neighboring vehicles. The keypoint annotations from the dashboard camera are given as input to \textit{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 \textit{SolvePnP} contains the relative translation and orientation of the neighboring vehicles which can be used by ADAS applications.

\subsection{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 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. Figure 2.6 depicts an example calibration produces by AutoCalib.
2.4 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 [25]. Resolutions of these cameras vary from 640x360 to 1280x720 pixels. Intrinsic parameters of these cameras are derived from their baseline calibrations as computed in § 2.4.2.
2.4.1 Keypoint Annotation Accuracy

AutoCalib leverages DNNs to identify keypoints on cars and computes calibrations by matching these keypoints 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 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 [103, 158]. However, we employ Dropout [154] 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 keypoint. The normalized error is defined as:

\[
E_{k}^{\text{norm}} = \sqrt{\left( x_{p,k} - x_{h,k} \right)^2 + \left( y_{p,k} - y_{h,k} \right)^2} / w_c
\]  

where \( E_{k}^{\text{norm}} \) 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 keypoint \( k \), \( x_{h,k} \) and \( y_{h,k} \) represent the human annotated x and y coordinates for keypoint \( k \), 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) keypoints. 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 2.7b details the DNN performance as a CDF of normalized error for the six keypoints described in §2.4.3. 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 keypoints 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 §2.2.6.

### 2.4.2 Ground Truth for Evaluation

Once the DNN annotates the keypoints 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 ten cameras. In order to do this, we identified distinguishable keypoints (for instance, trees,
Figure 2.8: Example of Ground Truth Keypoints (GTKPs) marked in a frame. These points are used to compute the ground truth calibrations and the distance RMSE.
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 [16]. These keypoints, referred to as Ground Truth Keypoints (GTKP), provide us a correspondence between image points and real-world coordinates. This correspondence is used to compute reference ground truth calibrations using \textit{SolvePnP}. For each camera, we collect 10 or more such GTKPs. Figure 2.8 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 §1.3.3. 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 1.1. 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 $D_{\text{reproj}}$ be the set of distances between all possible pairs of GTKPs reprojected from 2D image points. Similarly, let $D_{\text{real}}$ 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
Figure 2.9: 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%.

We now define Root Mean Square Error (RMSE) for a calibration as:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\varepsilon_{i}^{\text{norm}})^2}$$  \hspace{1cm} (2.3)$$

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 2.9. Thus, the RMSEs for manual ground truth calibrations can be treated as a benchmark for calibration performance.

2.4.3 Calibration Accuracy

**AutoCalib Distance Measurement Performance:** Figure 2.9 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 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 Keypoint Choice:** AutoCalib utilizes six keypoints: Left and Right Tail Lamp centers (LL and RL), License Plate center (LIC), Center Lamp (CL), and Left and Right Side Mirror (SWL and SWR) keypoints has a severe effect on the calibration RMSE.
Figure 2.11: Effect of car keypoint choice on calibration results for different cameras. Removing Left Side Mirror (SWL) and Right Side Mirror (SWR) keypoints has a severe effect on the calibration RMSE.
Right Side Mirrors (SWL and SWR). These keypoints 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 keypoints, 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 keypoints on the car. The camera was then calibrated using different combinations of these keypoints. We discovered that selecting non-planar and well separated keypoints improves calibration accuracy significantly. This is because picking only planar keypoints does not provide \textit{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 keypoints is depicted in Figure 2.10, where we compare the RMS Error CDF of all calibrations obtained from AutoCalib prior to filtering and averaging. On calibrating without SWL and SWR 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 \textit{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. Figure 2.11 shows the importance of non-planar keypoints for different cameras. Thus, our choice of keypoints helps \textit{SolvePnP} to have a reference point well separated in all 3 axes, and thus produce consistent estimates for the Y axis unit vector.
Figure 2.12: 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.
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 §2.2.6. 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 2.12, we compare the non-filtered set of calibrations with our preferred conservative filtering approach and an aggressive filtering approach. As described in §2.2.6, 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 2.12, 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, 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).
Figure 2.13: Number of detections required for precise calibration. Precision of the estimated calibration increases with detections, but the effect diminishes after 2000 detections.
**Number of Detections Required for Precise Calibration:** AutoCalib refines the estimated calibration using multiple frames to derive more accurate calibration values. Figure 2.13 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 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 2.14 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 [75] 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 [75] over multiple 10 min video sequences from a Seattle city traffic camera. Figure 2.17 shows the estimated vanishing point for different 10 min video sequences. As shown in the figure, the vanishing points are unstable due to vehicle
Figure 2.14: 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.

Figure 2.15: Tilt estimation error using AutoCalib and using vanishing points based approach from [75]. AutoCalib has an average tilt error of 2.04°, compared to 4.94° from [75]. Tilt is the angle between the Z axis unit vectors of two coordinate systems.
Figure 2.16: Accuracy of distance estimates computed from vanishing points using [75] and manually provided ground truth camera height vs AutoCalib. Despite providing the ground truth camera height, estimates from [75] have a mean RMSE of 21.59%. AutoCalib assumes no prior information about the camera height, and has a mean RMSE of 8.98%.

Figure 2.17: Stability of vanishing points derived using [75]
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.

Authors in [75] 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 [75]. Figure 2.15 shows the tilt estimation accuracy of AutoCalib and vanishing point based solutions [75]. Across all cameras, we find that AutoCalib’s estimated calibrations have an average 2.04 degrees of tilt error, whereas the vanishing points based approach from [75] has an average tilt error of 4.94 degrees. It is important to note that even a few degrees of tilt error can translate into large errors while measuring on-ground distances.

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 approach in [75] 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 2.18: Tilt estimation error using feature tracking based filters and statistical filters. The average tilt error of feature tracking based filters is 6.4 degrees which is higher than statistical filters 2.4 degrees.

Figure 2.16 compares the distance RMS error from the vanishing point approach [75] 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 [75] 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.

**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 two degrees. The calibrations that result in feature tracks with respective angles (w.r.t x-axis) differing by more than two degrees, 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 2.18
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.

2.5 Conclusion and Future work

In this chapter, 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.

Limitations of AutoCalib: Here we present some of the limitations of AutoCalib and approaches to address these limitations which are left for future work.

- 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.

- AutoCalib uses top ten vehicular models for calibrating the camera. This is because identifying the type of a vehicle is challenging due to poor resolution of current traffic camera installations. In the future, the cameras are expected to have better resolutions and vehicle classification techniques can be employed to identify the exact make and
model of the vehicle. With the make and model of the vehicle, the geometry from the respective specifications document can be used to calibrate the camera.

- AutoCalib designs a custom DNN for keypoint annotation. For this, we have annotated 486 car images. The size of this dataset can be extended to improve the accuracy of the annotation tool. However, these images must be diverse to improve the accuracy of the annotation tool. For the current implementation, we have characterized the diversity of the dataset with the human inspection. The size of the cropped image, distance between taillights etc., quantify the diversity of the dataset in terms of size. Also, the average intensity of the background can be used to quantify the diversity of the dataset in terms of light conditions. Using such automatic diversity quantifiers, a diverse set of vehicular annotations can be used to build the annotation tool.
Chapter 3: DashCalib: Automatic Live Calibration for DashCams

With reduced cost of cameras, many vehicular manufactures and drivers are deploying dashboard cameras in vehicles [12]. The dashboard camera is a key sensor in all autonomous navigation systems being designed today. DashCams can be classified into three categories. The first category of cameras are fixed to the vehicle’s body (often to the windshield) by the manufacturer (e.g., Toyota safety sense technology [31], Subaru EyeSight [28], KIA Drive Wise [18]). The second category consists of cameras that can be bought and installed by the user [21, 34]. The third category is based on smartphone apps [26, 27] that can transform a smartphone into a DashCam. In this chapter, we focus on calibrating the three categories of DashCams to support a wide range of vehicle safety applications.

![Manual calibration patterns used today](image)

Figure 3.1: Manual calibration patterns used today [22].
DashCam calibration is an essential step for emerging Advanced Driver Assistance Systems (ADAS) applications such as Forward Collision Warning (FCW) [21, 34, 67], Lane Departure Warning (LDW), Pedestrian and Cyclist Detection and Collision Warning (PCW) [23]. It can also be used for parking assistance. Views from multiple calibrated cameras can also be used to synthesize a bird’s eye view of the vehicle (a feature already available in some vehicles, such as Audi [1] and Mercedes [20]). With calibrated DashCams, different real-world distances on the road can be measured. This in turn enables geotagging of events (such as accidents), and creation and maintenance of 3D maps. Grassi et al. [84] exploits the calibration of the camera to map free parking spaces. Essentially, DashCam calibration is a crucial step upon which several ADAS applications depend.

DashCams must be checked for calibration errors and recalibrated continuously. Calibration errors will translate to catastrophic safety issues in different ADAS applications. New DashCam installations, windshield installations [3], collisions, blown airbags [14], installation of portable DashCams, and manual placement of smartphone-based DashCams on the smartphone holder are some events that necessitate periodic recalibration of the dashboard camera. Additionally, continuous vehicular movements may also reorient the camera and possibly change its position.

Camera calibration involves estimating two types of camera parameters: the intrinsic parameters such as focal length and distortion matrix of the camera; and the extrinsic parameters, which are orientation (represented by a rotation matrix $R$) and position of the camera in vehicle coordinates ($T$). In this chapter, we focus on automatic estimation of the extrinsic parameters (also refereed to as pose estimation) of dashboard cameras and assume that the intrinsic parameters, which are based on the camera’s make/model, are known. $R$ is a function of three Euler angles ($\alpha$, $\beta$, and $\gamma$) defined as the yaw, pitch, and
roll of camera coordinates with respect to the vehicle’s coordinates. \( T \) has three unknowns that are translations along the three axes. However, the height of the camera \( h \) (translation along \( y \)-axis) and the angles \( (\alpha, \beta, \gamma) \) are sufficient for measuring distances between any two points on the ground plane. Thus, a total of four unknowns \( (\alpha, \beta, \gamma, h) \) need to be solved for calibrating a DashCam.

Estimation of camera’s extrinsic parameters is challenging because it requires manual effort. Physically measuring the orientation of the installed camera is challenging because commodity compasses are affected by electromagnetic properties of the environment, leading to poor accuracy. Cameras are calibrated by positioning a chessboard-like pattern at a calibrated distance by a highly trained technician [30] (costs about 300-400 USD [11]). Different patterns used for calibration today are depicted in Figure 3.1. Automatic and live calibration can substantially reduce manual effort and cost involved in the calibration process. This can enable a wide range of vehicle-safety applications on commodity portable DashCams and smartphones. Further, the changes in camera position and orientation can be detected on the fly and calibration parameters can be kept up-to-date for the smooth functioning of ADAS applications.

Prior research for automatic camera calibration has assumed the properties of the road and road markers to calibrate the DashCams. In [86], the length and width of the lane markers are exploited to calibrate the DashCam. De et al. [71] and Ribeiro et al. [134] exploit lane boundary detection algorithms and use the road width and speed of the vehicle to calibrate DashCams. Catalá et al. [62] and Nieto et al. [122] assume the height of the camera and exploit the parallel property of the lane markers to derive the orientation of the camera. The major drawback to using the properties of the road is that the vehicle needs to be aligned with the road and should have the knowledge of road dimensions and lane
marker lengths. Additionally, errors in calibration are exacerbated by detection errors. In contrast, DashCalib assumes only the speed of the vehicle, which can be obtained from GPS or the OBD-II port.

We design and implement a system called DashCalib, that takes in a video snippet from a DashCam and computes its calibration parameters. Parallel lines in the Vehicular Coordinate System (VCS), when projected onto the camera frame, intersect at a point referred to as a vanishing point. Identifying the vanishing point along the length of the road (z-axis of VCS), is used to estimate the orientation of the z-axis in the Camera Coordinate System (CCS). DashCalib derives two vanishing points along the length and width of the vehicle which suffice to estimate $R$. The vanishing point along the length is referred to as the forward vanishing point (FVP) and the vanishing point along the width is referred to as the lateral vanishing point (LVP). DashCalib exploits the observed motion of static feature points on the road or the roadside to derive the FVP. We present a simple pipeline to extract lines joining taillights of other vehicles to derive the LVP. However, in our experiments, we face several challenges in estimating LVP using the lines joining taillights as such lines are almost always parallel which is referred to as ill-conditioned vanishing point [92, 187]. We propose approximation techniques to derive Euler angles ($\alpha$, $\beta$, and $\gamma$) in such cases.

DashCalib uses filtering and aggregation algorithms that exploit map information smartly by identifying intervals when accurate calibration values can be derived. DashCalib derives the height of the camera with respect to the ground by fitting the information derived from visual odometry to that of GPS-based odometry. DashCalib uses observed static points on the road along with the speed of the vehicle from the odometer to estimate the height of the camera.
We evaluate DashCalib using video feeds from twenty one DashCams that are placed at different orientations, and different lighting conditions. DashCalib is able to estimate the FVP with errors less than 4% and estimate the Euler angles ($\alpha$, $\beta$, and $\gamma$) with mean error of 2.0 degrees. This is in comparison to MonoSLAM-based approaches that have mean error of 9.7 degrees. Finally, we show that DashCalib is able to produce accurate calibration values with mean distance estimation errors of 5.7%, while manual calibrations have mean error of 4.1%. For anonymous viewing, a demonstration of DashCalib is available at [13].

In summary, we make the following contributions:

- Techniques to derive the rotation matrix by exploiting the relative motion and position of taillights of neighboring vehicles by smartly leveraging map information.

- Techniques to derive the height of the camera by comparing monocular visual odometry with GPS-based odometry.

- First robust automatic calibration system for dash cameras with mean distance estimation errors of 5.7%.

3.1 Related Work

*Dashboard camera calibration:* Most prior research has leveraged the road markers to calibrate DashCams. Catalá et al. [62] and Nieto et al. [122] assume the height of the camera and exploit the parallel property of the lane markers to derive the orientation of the camera. The length and width of the lane markers are exploited to calibrate the camera in [86]. De et al. [71] and Ribeiro et al. [134] exploits lane boundary detection algorithms and use the road width and speed of the vehicle to calibrate the DashCams. Grassi et al. [84] assumes the alignment of DashCam’s axes with the vehicle and exploits road-side markers (such as
stop-signs) for estimating the height of the camera. Gräter et al. [85] employs MonoSLAM techniques to estimate the ground plane and uses lines in the scene to estimate the Euler angles w.r.t the road. Hanel et al. [89] generates SFM (Structure from Motion [33]) 3D Map of calibration room instead of using calibration patterns for calibrating a DashCam. Generating such 3D maps is a laborious process involving a huge number of images [33]. Additionally, the vehicle needs to be driven to the calibration stations. In contrast, DashCalib attempts to remove the necessity of calibration stations by designing automatic calibration techniques.

Traffic camera calibration: The closest work to ours is by Dubska et al. [76] where the authors assume straightline motion of vehicles for computing the two vanishing points and use known average sizes of vehicles (width, height, and length) to automatically calibrate the traffic camera. AutoCalib [55] exploits deep neural networks (DNN’s) for identifying keypoints of neighboring vehicles and uploads a large number of images from an edge node (traffic camera) to a central server to derive accurate calibration values.

Drawbacks of prior work: The major drawback using the properties of the road is that the vehicle needs to be aligned with the road. Additionally, identifying lane markings or other geometric features automatically can be error-prone. Techniques such as those described in [68, 76] which rely on vehicle’s dimensions, are challenging to implement because the DashCam might be observing different types of vehicles with varying dimensions. Techniques that exploit DNN-based keypoint detection algorithms [55] are computationally intensive to incorporate on a DashCam. In contrast to these works, DashCalib use simple red thresholding and lightweight vehicle detectors for deriving the LVP.
3.2 Design

This section presents the design challenges and the design details of DashCalib.

3.2.1 Overview

The DashCalib pipeline is depicted in Figure 3.2 and has the following steps:

- The map information is analyzed to identify straight road segments that are ideal for performing calibration. Based on the trigger, video frames are processed by tracking different feature points to derive the FVP (§3.2.3).

- Taillights of neighboring vehicles are extracted and processed to estimate the LVP. Also, approximation techniques are proposed when these lines are almost parallel for estimating the Euler angles (§3.2.4).

- GPS odometry and feature tracking techniques are exploited to estimate the height of the camera. Errors incurred in this process are eliminated by identifying instances when GPS odometry is accurate (§3.2.5).

Figure 3.2: DashCalib pipeline for automatic calibration of DashCams.
3.2.2 Challenges

DashCalib faces the following key challenges in designing techniques for automatic DashCam calibration:

- **Ill-conditioned LVP**: We observed for most of the DashCam installations, that the lines joining the taillight pairs (in the camera frame) are almost parallel leading to erroneous estimates of LVP. This is exacerbated by the errors in taillight-detection techniques.

- **Consolidating the estimates**: Vanishing points estimated by observing different road segments and neighboring vehicles (taillights) might be different. Additionally, the rotation matrix $R$ derived from vanishing points might not give a stable estimate of the height. We observed that errors in $R$ translate to a time-varying estimate for the height.

- **Errors in GPS-based odometry**: DashCalib relies on GPS odometry to derive the height of the camera. However, we observed significant errors in GPS odometry, which translated to errors in the estimated height.

3.2.3 Forward Vanishing point (FVP)

Estimating the FVP is a non-trivial problem due to lack of calibrated geometric shapes on the ground. Most prior work has focused the problem of estimating road direction by performing frame-by-frame analysis. Lane marker detection techniques [47, 122] can be exploited by fitting lines across them and solving for the FVP as in Seo et al. [144] and Suttorp et al. [157]. Lane markers might not be visible, or might not be present in all parts of the road. Therefore, previous research has employed computationally expensive scene
analysis [101,116,118] to derive the vanishing point. However, the vanishing point detection techniques for DashCam must be lightweight. Employing such techniques is challenging because the vehicle might not be aligned with the road/lane-markers, or the road might have curvature leading to incorrect vanishing point estimates. Additionally, most lane-detection techniques (such as described by Aly et al. [47]) depend on camera calibration.

DashCalib exploits vehicular motion to derive the FVP; it calibrates the vehicle with respect to its body instead of the road. For the sake of simplicity, let us assume that the vehicle is moving on a straight line aligned with the length of the vehicle and the direction of the road. In the VCS, all the stationary points belonging to the road and surroundings move with the vehicle’s velocity but in the opposite direction. The path traced by these points are parallel straight lines, which are also parallel to the length of the vehicle. DashCalib tracks different feature points to derive the FVP. Figure 3.3 depicts tracking of different feature points to derive the FVP.

Figure 3.3: An example snapshot of DashCalib while estimating the FVP.
DashCalib faces several errors in identifying the stationary feature points and misalignments of the vehicle’s body w.r.t its direction of motion. Additionally, time-varying calibrations due to change in camera placement and orientation pose a big challenge in estimating a stable vanishing point. If the vehicle’s motion is not on a straight line, then the path traced by the feature points when projected on the camera frame creates complex patterns, making vanishing point estimation challenging. DashCalib addresses these challenges by designing the \textit{ComputeFwdVP Algorithm} (Algorithm 3), which identifies straight road segments for calibration, and presents statistical filters to remove outliers arising due to misalignments, tracking errors, and nonstationary points.

\textbf{Identifying the straight road segments:} First, the road segments that are straight are identified by analyzing the map information. With high probability, the vehicle will be navigating on this road segment by aligning with the road. We can assume the vehicle’s motion is along its length. This method is implemented in Line 5.

\textbf{Statistical filters to remove outliers:} Once the straight road segments are identified, DashCalib analyzes the frames by employing feature-tracking algorithms (line 3) to derive tracks $K$ of feature points. Essentially, different distinguishing points on the road are tracked across the frames. These tracks are passed through a linear fitting module (line 3). Nonstationary points, and matching errors are some of the errors that can be eliminated by measuring the error incurred during the linear fitting process. Due to matching errors, different feature tracks are concatenated as a single track, or past tracks gets interchanged. These tracks have different line equations, therefore employing linear-fitting techniques will result in a fitting error, which is eliminated (line 3). The pairwise intersection points of the filtered lines are computed and the vanishing point is estimated by deriving its cluster center. This estimated vanishing point is compared with the past estimates and rejected if it
is by far a value greater than threshold $\varepsilon$ (5% of frame width is used as a threshold in our experiments). The outlier estimates caused by lane changes and sudden turns are eliminated by this filter. The vanishing point estimate is triggered-based on the map information. The estimated vanishing point is then passed through an exponential moving average filter (line 3). This filtering process updates the vanishing point estimate by giving more weight to recent observations. Whenever the DashCam calibration changes, the change in vanishing point is detected and the camera is recalibrated with the most recent estimate of the vanishing point.

**Euler angles from FVP:** The direction of the FVP is equated to the direction of the z-axis, i.e., the third column of the rotation matrix [64] to solve the Euler angles as,

$$
\begin{bmatrix}
\cos(\alpha) \sin(\beta) \cos(\gamma) + \sin(\alpha) \sin(\gamma) \\
\sin(\alpha) \sin(\beta) \cos(\gamma) - \cos(\alpha) \sin(\gamma) \\
\cos(\beta) \cos(\gamma)
\end{bmatrix}
= \begin{bmatrix}
k_1 \frac{(u_z - c_u)}{f_u} \\
k_1 \frac{(v_z - c_v)}{f_v} \\
k_1
\end{bmatrix}
$$

(3.1)

where, $(u_z, v_z)$ is the FVP and $k_1 = \frac{1}{\sqrt{1 + \frac{(u_z - c_u)^2}{f_u^2} + \frac{(v_z - c_v)^2}{f_v^2}}}$. In Equation 3.1 above, the Right Hand Side (RHS) is known because it is based on the camera’s intrinsic parameters and the estimated location of the FVP. The above equation gives two constraints, by equating first two rows from Left Hand Side (LHS) with first two rows of RHS. The third row of the equation is dependent on the first two rows, because the length of directional vectors is one. Therefore, in Equation 3.1 we effectively have two constraints and three unknowns. In the next section, we present one more equation relating the LVP and the Euler angles. With these three equations, DashCalib can solve for the three Euler angles.

### 3.2.4 Lateral Vanishing point (LVP)

Estimating the LVP is challenging due to the difficulty in finding appropriate lines that are parallel to the vehicle’s width. Different lines along the width of the vehicle,
Algorithm 3 ComputeFwdVP Algorithm

Input : Video Frames $V$
Output : Vanishing Point $u_x, v_x$

WaitForStraightRoadSegment()

$F \leftarrow \text{ComputeFeaturestoTrack}(V)$

$K \leftarrow \text{DeriveFeatureTracks}(F)$

$L \leftarrow \phi$
// Lines along the feature tracks.

$C \leftarrow \phi$
// Estimated vanishing points list.

for $k \in K$

do

$m, c, \text{err} \leftarrow \text{LinearFit}(k)$

if $\text{err} < \varepsilon$

then

add $(m, c)$ to $L$

end

end

if $|L| \geq 2$

then

$I \leftarrow \text{FindPairwiseIntersections}(L)$

$C_x, C_y \leftarrow \text{ComputeCentroid}(I)$

if $\text{dist}((C_x, C_y), (u_x, v_x)) < \varepsilon_1$ or $C = \phi$

then

add $(C_x, C_y)$ to $C$

end

$u_x, v_x \leftarrow \text{ExpMovingAvg}(C)$

end
such as train tracks, lines at the traffic intersection (if marked with horizontal lines), or roadside advertisement boxes (if aligned with the road) can be exploited to estimate the LVPs. However, these approaches are dependent on particular markings on the road and therefore are not suitable for calibration needs of ADAS applications. Lack of lines on the roads that are parallel to the width of the roads makes LVP estimation challenging.

DashCalib estimates the LVP by identifying lines joining taillights of other vehicles traveling in the same direction when the road segment is straight. These lines are expected to be parallel to the width of the vehicle on a straight road. Also, due to relative motion of the vehicles, multiple snapshots of these lines at various relative positions can be captured. These lines are parallel to each other and the width of the vehicle. DashCalib estimates the vanishing point along the width of the vehicle by computing the intersection of these lines when projected to the camera frame.
Algorithm 4 Compute LVP Slope

**Input**: Video Frames V  
**Output**: Slope of LVP m

`WaitForStraightRoadSegment()`

`LineSet ← φ`

**for each frame in V do**

- `RedImage ← RedThresholding(frame)`
- `Contours ← FindContours(RedImage)`
- `Centers ← FindContourCenters(Contours)`
- `Boxes ← HarrVehicleDetector(frame)`
- `RedLightPairs ← FindPairs(Boxes, Centers)`
- `Lines ← ComputeLines(RedLightPairs)`
- add `Lines` to `LineSet`

**end**

`m ← ComputeAvgSlope(LineSet)`

---

The *Compute LVP Slope* algorithm (Algorithm 4) identifies the taillights of neighboring vehicles and estimates the slope of the vanishing point (described later in this section) for deriving the Euler angles. We employed taillights-detection techniques described in Yenamandra et al. [178] by using a red-thresholding color filter (line 4). At each pixel, the camera captures 8-bit color values along red (R), green (G), and blue (B) dimensions. By red thresholding, we can filter out all the colors that are not in the red space. This step detects all the red colored blobs in the image. If a car is red in color, then different blobs belonging to it are also detected. Once the red blobs are identified, contours are drawn (line 4) and analyzed to identify their respective centers (line 4). Taillights belonging to the same vehicle must be paired up in order to draw the lines joining them. DNN-based annotation tools [55, 56] are computationally expensive for performing this operation. DashCalib uses a lightweight Haar-based vehicle detector [91] to group red blobs belonging to the same vehicle (line 4).
Solving for $\alpha$, $\beta$, and $\gamma$: With the obtained vanishing points, DashCalib solves for the rotation matrix between camera coordinates and VCS. The direction of the LVP is equated to the direction of VCS’s x-axis, i.e., the first column of the rotation matrix \([149]\) is given by,

\[
\begin{bmatrix}
\cos(\alpha) \cos(\beta) \\
\sin(\alpha) \cos(\beta) \\
-\sin(\beta)
\end{bmatrix}
= \begin{bmatrix}
\frac{u_x-c_u}{f_x} \\
\frac{v_x-c_v}{f_v} \\
k
\end{bmatrix}
\] (3.2)

where, \(k = \frac{1}{\sqrt{1 + \frac{(u_x-c_u)^2}{f_x^2} + \frac{(v_x-c_v)^2}{f_v^2}}} \) [64] and $\alpha$, $\beta$, $\gamma$, and $(u_x, v_x)$ are the Euler angles of $R$ and the LVP (along x-axis) respectively. In Equation 3.2 above, the RHS is known because it is based on the camera’s intrinsic parameters and the estimated location of LVP. From the equation, we can derive $\alpha$ and $\beta$. The above equation equates the first column of $R$ with the estimated LVP to solve for the Euler angles. The above equation gives two constrains, by equating the first two rows from LHS with the first two rows of RHS. The third row of the equation is dependent on the first two rows, because the length of directional vectors is one.

**Algorithm 5** ComputeEulerAngles Algorithm

**Input**: Video Frames $V$

**Output**: Euler angles $\alpha$, $\beta$ and $\gamma$

WaitForStraightRoadSegment()

$(u_x, v_x) \leftarrow \text{ComputeFVP}(V)$

$m \leftarrow \text{ComputeLVPSlope}(V)$

Compute $\alpha$ using Equation 3.3.

Compute $\beta$, $\gamma$ using first two rows in Equation 3.1.

**Erroneous LVP estimates**: The parallel lines that join the taillights of the other vehicles, when projected to the camera frame intersect at the LVP. The technique described for FVP can be used to estimate the LVP from the set of lines joining the taillights. But, during our experiments with different placements of the DashCam, we observed the projected lines were
Figure 3.5: *Lines joining the taillights, which are almost parallel to each other.*

almost parallel leading to a large deviation in the vanishing point estimate. This is referred to as *ill-conditioned* vanishing point [92, 187]. Distance between such parallel lines is exploited in work done by He et al. [92] to calibrate the traffic camera. However, the distance between taillight pairs is not known because the velocity of taillight pairs is not known and they can be from different vehicles. These vanishing points are usually outside the frame at a far distance. Errors in identifying the center of taillights translate to a significant error in the estimated vanishing point. Figure 3.5 shows a sample output of the taillight detection pipeline where the lines joining taillights are almost parallel. We attempted the estimate with different statistical techniques such as average of the estimated LVPs (AVLP). However, these techniques are very sensitive to errors from the taillight detection pipeline. Because the lines in Figure 3.5 are almost parallel, the vanishing points computed by the pairwise intersections are widely dispersed and their centroid is far from the ground truth. Thus, the technique similar to FVP estimation, does not work here. However, we observe that the line with average slope of the lines joining taillights of vehicles does pass through the ground truth vanishing point. We show that, in fact, this direction suffices to compute
the Euler angles and is a more robust approach. Figure 3.6 shows the estimated LVPs by observing pairs of taillights, the average slope of lines joining taillights, and the vanishing point estimated from the ground truth calibration. Using the first two sub-equations of Equation 3.2, we arrive at the following equation,

\[
\tan(\alpha) = \frac{v_x - c_v}{u_x - c_u} \frac{f_u}{f_v} \approx \frac{v_x f_u}{u_x f_v} \approx \frac{m f_u}{f_v} \quad (3.3)
\]

where \(m\) is the average slope of the lines joining taillights (estimated in line 4). We can use the average slope of lines joining the taillights to approximate the slope of the vanishing point when the vanishing point is far from the camera frame center at a large distance compared to width and length of the frame. The ALVP \((\bar{u}_x, \bar{v}_x)\) is different from the ground truth location of the vanishing point (depicted in Figure 3.6). We observe that LVP and ALVP are inaccurate and far from LVP estimate using ground-truth calibration. Substituting these values in the above equations will give erroneous results. The Euler angles computed using the slope of the lines are more accurate, as depicted in Figure 3.17.

**Putting it all together:** The *ComputeEulerAngles* algorithm (Algorithm 5) for solving the Euler angles is triggered upon identification of straight road segments (line 5). Once the straight road segments are identified, the FVP estimation is triggered (line 5). LVP estimation process is triggered to derive the slope of the LVP (line 5). By observing the slope \(\bar{m}\) we can solve for \(\alpha\) using Equation 3.3 (line 5). By substituting the FVP and \(\alpha\) in the first two sub-equations of Equation 3.1, we can solve for the other two angles (line 3.3.2).

### 3.2.5 Estimating the Height of DashCam

The height of the camera is an important parameter for measuring distances on the ground. Images contain 2D information and, by observing just the image information,
it is challenging to measure real-world distances such as height. Manual measurements are highly error-prone for estimating the height because it often requires measuring the dimension of the vehicle, height of tires, etc.

DashCalib solves for the height of the camera by using visual odometry techniques on the odometer readings from the GPS or OBD-II port. If the height of the camera is accurately known, and if we employ visual odometry techniques [153] and track the static feature points on the road, the distance traversed by the vehicle can be estimated. Due to large smoothing windows of these sensors, we observe that the odometer measurements are often inaccurate. DashCalib studies these sensors and identifies instances where odometer sensors can accurately perform height estimation.

Figure 3.6: Technique from Algorithm 1 fails to estimate the LVP. The intersections of lines joining taillights are widely dispersed far from the Ground Truth.
3.2.5.1 Estimating height from ground feature points

*Intuition:* DashCalib analyzes points belonging to the road/ground plane to derive the height of the camera. Let \( P(u, v) \) be the feature point belonging to the road/ground plane. Essentially, a pinhole model represents a ray emanating from the camera center toward feature point \( P \). Let the vehicle’s velocity be \( V \), which can be obtained from GPS-based odometry. Because \( P \) is stationary with respect to the ground, its velocity in VCS will be \( V \), but directed in the opposite direction of the vehicle’s motion. Consider two instances of a feature point at times \( t_1 \) and \( t_2 \). Figure 3.7 depicts the motion of DashCam with respect to \( P \). By solving the intersection of two rays separated by a distance of \( V(t_2 - t_1) \), we can solve for the height of the camera, \( h \).

*Mathematical formulation:* For the sake of simplicity, let us analyze the feature point \( P \) in ACCS (depicted in Figure 3.8). The y-coordinate of \( P \) in ACCS is negative of the height of the camera i.e., \( -h \). Because ACCS is collocated with CCS, the translation \( T \) between the two coordinate systems is a null vector. We rewrite the pinhole equation as,

\[
\begin{bmatrix}
  x \\
  -h \\
  z
\end{bmatrix}
= R^{-1}M^{-1} \begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix}.
\] (3.4)
where \((x, -h, z)\) and \((u, v)\) are coordinates of \(P\) in ACCS and pixel position, respectively. Because we know one coordinate of \(P\) (y-coordinate: \(-h\)), we can solve for the other two coordinates in terms of \(h\). The value of \(s\) is derived in terms of known parameters \(u, v, \) and \(h\) by observing the second row of the above equation. Using this value of \(s\), \(z\)-coordinate, which signifies the distance between \(P\) and the DashCam along the length of the road, \(z = d\), can be derived from the above equation as,

\[
d = -\frac{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}^T \begin{bmatrix} R^{-1}M^{-1} & u & v \\ u & v & 1 \end{bmatrix}}{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R^{-1}M^{-1} & u & v \\ u & v & 1 \end{bmatrix}}. \tag{3.5}
\]
For the sake of simplicity, let us write the above Equation as, \( d = -hf(u,v) \). The distance between two pixel positions \( P_1(u_1, v_1) \) and \( P_2(u_2, v_2) \) can be estimated using equation 3.5 as,

\[
d_{12} = -h(f(u_2,v_2) - f(u_1,v_1)).
\] (3.6)

where \( h \) is the height of the DashCam. If the height \( h \) of the camera with respect to the ground plane is known, then the velocity of the vehicle can be estimated by observing the rate of change of \( d \). However, we know that the velocity of the feature point is \(-V\), where \( V \) is the velocity of the vehicle. The height of the camera can be estimated by observing \( P \)'s motion between time \( t_1 \) and \( t_2 \) from Equation 3.6 as,

\[
h = \frac{V(t_2 - t_1)}{f(u_2,v_2) - f(u_1,v_1)}
\] (3.7)

where \((u_1, v_1)\) and \((u_2, v_2)\) are pixel positions of \( P \) at times \( t_1 \) and \( t_2 \), respectively.

### 3.2.5.2 Accuracy of \( V(t_2 - t_1) \)

We evaluate the accuracy of two different techniques for measuring the speed, namely using GPS and using the OBD II port of the vehicle. The ground truth for this experiment...
is measured using a high-precision GPS device (SXBlue II [29]). The experiments are conducted using these three devices in a car. The data is based on 5 miles of driving on a state road and 12 miles of driving on a highway. The speed data is extracted from the three devices and time aligned by correcting for their time offsets.

Figure 3.9 shows how the speed measured by the three devices varies with time for a highway scenario. We observe that the speed measurement using the GPS as well as OBD-II lag in comparison to the ground truth especially when accelerating and decelerating. The OBD-II computes the speed of the car based on the average distance traveled by the four tires. Both devices use a smoothing function over time [176] that results in the observed lagging phenomenon. Figure 3.10 shows how the standard deviation of the distance measurement error (expressed as a percentage of the distance) changes with the measured distance. We observe that the error is lower for highways compared to state roads. This is due to lag in measuring speed when accelerating and decelerating, which is encountered more frequently in state roads.
DashCalib triggers the height estimation process whenever the velocity of the vehicle is uniform over a long time. This happens particularly on the highways compared to state roads. The points belonging to the road can be detected by different techniques. DashCalib employs lane marker identification techniques [47] and tracks the centers of lanes for estimating the height of the camera. Also, the ground points that are close to the vehicle are selected to derive the height of the experiments.

### 3.3 Evaluation

**Implementation**: The complete DashCalib pipeline is implemented as a python code. All vector algebra operations are sped up using the NumPy library and OpenCV 3.2 is used for calibration computations. For vehicle detection, we used a custom trained Haar-based vehicle detection module. DashCalib is deployed on a laptop running on Intel core i7 CPU and 16 GB RAM. For a 1920x1080 video frame, this deployment runs at a rate of 30 fps for identifying FVP, 3 fps for estimating LVP, and 30 fps for estimating the height.

Figure 3.11: Experiments for evaluating DashCalib are collected at different light conditions, orientations, and at random places on windshield of the DashCam.
In our evaluation, we analyze DashCalib’s performance by measuring the accuracy of the final calibration estimates. We also present evaluation of different intermediate parameters involved in DashCalib.

**Ground Truth and Dataset for Evaluation**: To evaluate DashCalib, we performed a total of 21 experiments, each of them 10-15 minutes long during different times of the day and light conditions. During each experiment, we recorded the video from DashCam and GPS readings. The DashCam is placed at different orientations with respect to the vehicle facing in the forward direction. Figure 3.11 shows the snapshot captured by DashCalib for different experiments (E1-E21) depicting the light conditions and installations with respect to the road. E1 is taken in morning daylight conditions, E2 is tilted toward the right, E3-E8 are performed during afternoons on different days, E10-E14 are performed on cloudy days, E15 is rotated toward the right, E16 is tilted forward, E17-E20 are performed on different days during evening times, E21 is performed during bad light conditions. For each experiment, the camera is installed at a random place on the windshield. A Galaxy S8 phone is used as a DashCam to record video of the traffic and GPS traces. Intrinsic parameters of the DashCam are estimated by using chessboard images (only once) before the start of the experiments.

**Manual Calibration**: For each experiment, we performed manual calibration using 15 chessboard images taken from a distance of 1 meter to 15 meters (1 meter of separation). The chessboard images taken at different distances are also used for estimating the accuracy achieved by DashCalib for measuring distances on the ground. We also used manual calibration as ground truth for studying the performance of various intermediate parameters involved in DashCalib.
3.3.1 Accuracy of FVP

Figure 3.12: Absolute error in estimating the FVP for different experiments (mean of 1.5\% along x-axis and 1.4 \% along y-axis).

Figure 3.12 shows the absolute percentage error of FVP estimated by DashCalib for different experiments. The absolute percentage error for the x-coordinate of vanishing point is defined as, \[ \left| \frac{VP_x^s - VP_x^g}{W} \right| \times 100 \] where \( VP_x^s \) and \( VP_x^g \) are x-coordinates of FVP estimated by DashCalib and using manual calibration, respectively, and \( W \) is the width of the frame. Similarly, the percentage error for the y-coordinate is defined by normalizing with the height of the frame. We selected this metric because it is independent of the resolution of the image. DashCalib estimates the FVP with errors less than 4\%. We observed that experiments performed on straight roads and in bright light conditions have fewer errors (E1-E10: average error of 1.1\%) compared to other experiments (E11-E21: average error of 1.7\%).

Number of frames processed: To measure the time required for estimating the FVP, we evaluated the performance of DashCalib for varying number of frames. Figure 3.13 shows the percentage error (normalized with the frame width and height) of DashCalib in estimating the x-coordinate and y-coordinate of the forward vanishing point with the
number of frames processed. The estimated vanishing point converges to an accurate and stable vanishing point with more frames. Figure 3.14 shows the length of video required by DashCalib for estimating the stable FVP. DashCalib estimated the stable vanishing point with less than 4-minutes of video for all the experiments. The number of frames required for estimating a stable vanishing point depends on the map information and the number of lane changes. We observed that the computations performed on highways (such as E14, E20) converge quickly (in about a minute) to the stable estimates as the vehicle’s trajectories on highways are approximately straight lines.

Figure 3.14: Length of video processed for estimating the stable FVP (mean of 167 seconds).
3.3.2 Accuracy of LVP and Rotation Matrix

For estimating the Euler angles, DashCalib uses the slope of the LVP, derived by approximating it to the average slope of lines joining taillights. Figure 3.16 shows the error incurred in estimating the slope of vanishing points by analyzing lines joining taillights. The slope from manual calibration is used as the ground truth. Multiple lights on each vehicle, multiple lights from nearby vehicles, and erroneous detections created outliers in this estimation process.

Using the LVP and FVP estimates, DashCalib estimates the yaw, pitch, and roll of VCS with respect to the CCS. The errors of the estimates are compared using angles
estimated from manual calibration as ground truth. Figure 3.15 depicts the estimation error of DashCalib for computing the three Euler angles. The angles derived by DashCalib have errors less than 7 degrees. *Yaw* (α rotation around z-axis) is estimated based on the slope of LVP (Equation 3.3), therefore its error is correlated with errors from slope estimation (depicted in Figure 3.16). Errors in *Pitch* (β) and *Roll* (γ) are correlated with the FVP. This can be observed in E12-E20, where the errors in FVP are high, leading to the significant errors in pitch and roll.

![Average absolute error graph](image)

**Figure 3.17**: *Absolute errors in estimating pitch (α), yaw (β), and roll (γ) using IMU sensors and 5-point algorithm [90] based approach.*

**Comparison with inertial sensors-based calibration**: The IMU sensors installed with the DashCam can also be exploited for calibration. The gravity vector’s direction, estimated in the CCS, gives the direction of VCS’s y-axis, which can be used to calibrate the DashCam. It can be used along with the FVP or LVP (slope of the taillights) for calibrating the DashCam. The average LVP (ALVP) from the taillights can be used in place of LVP for calibration. Figure 3.17 shows the comparison of estimating the orientation of the CCS using different approaches. For these experiments, we have collected the accelerometer readings (from smartphone) when the vehicle is stationary for deriving the direction of the gravity vector.
Comparison with MonoSLAM-based calibration: Epipolar geometry [183] can be exploited to solve the direction of VCS’s y-axis. The road plane can be assumed as a flat 2D plane, and the vehicle’s motion can be modeled as a translatory and rotatory on the two dimensional plane. The rotation matrix and translation matrix between two camera views can be derived by employing 5-point algorithm [123, 155] on the common feature points. We have used the algorithm presented in Nister et al. [123] for comparison. This technique is used widely in visual odometry and Mono-SLAM based applications. The vehicle’s axis of rotation will be along the vertical direction (perpendicular to the road plane), which can be estimated in the CCS to give the direction of VCS’s y-axis. We have employed this approach to derive the axis of rotation at turns where there is a significant amount of rotation. This technique is applied on different video segments and the average vertical axis direction is derived based on 50 runs (on 50 different video segments) per experiment. The obtained vertical axis direction is used along with the FVP for estimating the Euler angles. Figure 3.17 shows the average absolute errors of the Euler angles estimated by this approach. DashCalib outperforms the MonoSLAM-based approaches because these approaches are affected by feature tracking errors, potholes and irregularities on the road. DashCalib employs simple taillight detection and associates the lights based on the vehicle detection bounding boxes.

Yaw (α) is estimated based on the direction of VCS’s x-axis. DashCalib and LVP+gravity use accurate slope of the LVP for deriving the yaw and therefore outperform the other approaches. As mentioned in § 3.2.4, the slope of ALVP is different from slope of LVP, therefore using ALVP will result in errors in estimated Euler angles. For this reason, ALVP+FVP has high error in estimating the yaw. Pitch (β) and Roll (γ) are based on VCS’s y-axis (gravity direction) and z-axis (FVP direction). Errors from the commodity
accelerometer translate to orientation errors of the CCS, leading to bad performance for
the Slope+Gravity and FVP+gravity approaches. Estimated yaw ($\alpha$) is employed for
calculating roll and pitch for ALVP+FVP. Errors in yaw translate to errors in pitch and roll
for ALVP+FVP. Because of these factors, DashCalib outperforms the IMU sensors based
approaches for DashCam calibration.

### 3.3.3 Accuracy of Estimated Height

We derive the height of the camera with respect to the ground by fitting the information
derived from visual odometry to that of GPS-based odometry. Figure 3.18 shows the errors
of DashCalib for estimating the height of the camera. Essentially, DashCalib uses GPS
speed readings, derived $R$, and feature tracks of ground points for the derivation of height.
Errors in each of these parameters will contribute to errors observed in the estimated height.
We observe that the experiments performed during bright light conditions (E1-E9) have
fewer errors (average of 14 cm).

![Figure 3.18: Absolute errors in height estimation by DashCalib (mean of 0.24m).](image)

94
3.3.4 Calibration Accuracy

This section presents the performance study of the following approaches in measuring real-world distances: (i) manual calibration; (ii) DashCalib, and (iii) only rotation matrix (OnlyR): The rotation matrix estimated from DashCalib is used along with the translation matrix $T$ estimated from manual calibration. We used two points separated by 5-meters along the length of the vehicle to evaluate the accuracy achieved by DashCalib with respect to the other techniques. Figure 3.19 shows the accuracy of measuring distances using calibrations from manual calibration, OnlyR, and DashCalib.

**Manual calibration:** We observe that manual calibration has a mean error of 4.1% error.

**DashCalib:** The performance of DashCalib depends on the accuracy of the estimated parameters ($\alpha$, $\beta$, $\gamma$, and $h$). DashCalib measures the distances on the ground with mean error of 5.7% which is 1.6% more than manual calibration. We observe that DashCalib outperforms OnlyR even when there are errors in the estimated parameters. To explain this, consider the experiments E10-E18, where DashCalib outperforms OnlyR. This counterintuitive behavior is explained by the fact that the errors in estimated $R$ and $h$ get canceled.
To understand this, consider two feature points $P_1$ and $P_2$ on the ground plane along the length of the road. We want to measure distances between $P_1$ and $P_2$. DashCalib uses GPS odometer readings to derive the distance between $P_1$ and $P_2$ and thus derives $h$. Essentially, if there is an error in $R$ that shortens the estimated distances on the ground, then using the same $R$, DashCalib estimates a greater value of $h$ (E10-E14). Therefore, DashCalib is able to measure distances accurately despite slight orientation errors. We observe that the calibration accuracy is better during afternoon light conditions (good light condition). This can be seen in E1-E10 (average of 5.1%) compared to E11-E21 (average of 6.3%).

**OnlyR:** For the cameras fixed by the manufacturer and other fixed mount cameras that do not have access to odometer information, OnlyR signifies the accuracy achieved by DashCalib. For these cameras, the height of the camera remains unchanged and the rotation matrix $R$ needs to be estimated periodically. We observe that the accuracy of OnlyR is correlated with the accuracy of the estimated FVP. Errors in the FVP will reorient the VCS with respect to the CCS. This is the reason we observe high variance in the performance of OnlyR. For experiments with bright light conditions (E1-E10), we observe that the estimated $R$ has significantly fewer errors, leading to better performance (average error of 6.1%) compared to other experiments (E11-E21: average error of 16.2%).

**Comparison with MonoSLAM based calibration:** The technique described in the above section 3.3.2 is used to derive the Euler angles by employing the 5-point algorithm at turns. Figure 3.20 shows the performance of this approach in comparison with DashCalib and OnlyR-based approaches. This result can be expected because a few degrees of error in the Euler angles translates to significant errors in distances measurements.
Figure 3.20: Absolute errors of manual calibration (mean of 4.1%), OnlyR (mean of 11.4%), DashCalib (mean of 5.7%), and 5-point based approach (mean of 38.0%) for measuring distances along the road.

3.4 Conclusion

In this chapter, we propose DashCalib, a system for scalable, automatic calibration of DashCams. DashCalib exploits the motion of the vehicle, taillights from neighboring vehicles, and GPS-based odometry to derive a large dataset of parallel lines along the length and width of a vehicle in the camera frame and uses a novel filtering and aggregation algorithm to automatically produce a robust estimate of the camera calibration parameters. DashCalib implemented using commodity DashCams estimates real-world distances with a mean error of 5.7%. The mean error of manual calibration is 4.1%. DashCalib can replace manual calibration and enable a wide range of ADAS applications. Demonstration of DashCalib (for anonymous viewing) to measure distances on the ground for different DashCam videos can be found in [13]
Chapter 4: RoadMap: Mapping Vehicles to IP Addresses using Motion Signatures

The popularity of in-vehicle cameras and smartphones provides an opportunity to implement cooperative vehicular applications. The US Department of Transportation issued a new rule requiring car manufacturers to include rearview cameras in all cars manufactured after May 1, 2018 [121]. Meanwhile, smartphones, which are typically equipped with cameras, GPS and radio interfaces, are available to more than 62.5% of the U.S. population [66]. Worldwide smartphone sales accounted for 55% of overall mobile phone sales in the third quarter of 2013 [81]. In vehicles, smartphones can be mounted on the dashboard to provide services such as navigation, over speed warning, and traffic alert. Also, these smartphones can be leveraged to communicate with neighboring vehicles (IVC) and are equipped with cellular connectivity.

Emerging cooperative vehicular applications, such as vehicle platooning [36], adaptive cruise control and autonomous vehicle, can potentially benefit from the information of the vehicles multiple hops away, as well as the information of the immediate neighboring vehicles. The adaptive cruise control system can adjust a vehicle’s speed if it knows the acceleration and speed of the vehicles in front. However, in these applications, collaboration can only benefit a vehicle if the relative locations and communication identities (e.g. IP addresses) of the other vehicles are known. For example, vehicles typically use RADAR
and LIDAR to scan neighboring vehicles in Line-of-Sight (LoS). However these sensors cannot identify the IP-Address of neighboring vehicles. Communicating with vehicles that have known relative locations can further expand the scanned region. But it is difficult to utilize information provided by vehicles with inaccurate relative location.

The identities of neighboring vehicles can be obtained by leveraging QR-Codes, Ultrasonic communication, Visual Light communications, Wi-Fi MIMO based Angle of Arrival (AoA), or video captured by cameras. Employing these modes of communications can potentially give the identity of vehicle in FoV along with its relative location. However, these schemes require neighboring vehicles to have additional hardware upgrades. With minimal assumption of a dashboard camera, computer vision techniques which identify a vehicle based on visual features (color, aspect-ratio, SIFT features etc.) can be employed. However, there can be multiple such vehicles with the same identical visual features. Additionally, when legacy vehicles co-exist, detecting the relative locations and communication identities of the collaborating vehicles is a challenging problem. In this chapter, we attempt to solve this problem by assuming minimal hardware requirement (such as a simple smartphone or a deployed dashboard camera). Essentially, RoadMap matches the motion-traces of the vehicles observed from a camera with the motion-traces received from IVC. By doing so, RoadMap solves two major problems involved in cooperative vehicular applications. First, relative localization problem: Can the neighboring vehicles be localized with respect to a given vehicle? Second, targeted communication problem: Can a vehicle communicate with a vehicle at a given relative location (e.g., vehicle in front)?.

Existing schemes that focus on addressing only one of the problems cannot satisfy the requirement of cooperative vehicular applications. Many schemes have been proposed for vehicle localization such as GPS based localization, map matching, and dead reckoning [60].
These systems cannot determine the relative locations of legacy vehicles. Devices such as camera and RADAR do not require cooperation from other vehicles. But, they do not know which vehicle they are localizing. Schemes based on radio RSSI [110, 129] can potentially localize vehicles not in LoS. The problem is that such schemes do not work for legacy vehicles. By using the emerging IVC techniques, vehicles can collaborate to extend the capability of their sensing devices. To take advantage of information provided by neighboring vehicles, the design must address the following challenges:

- **Lack of observable identities**: A vehicle observes other vehicles through its camera or RADAR, without knowing their global identities (such as MAC addresses or IP addresses) of the detected vehicles. Observable features such as color, aspect ratio of vehicle and radar-signature can correspond to multiple vehicles. Thus, a vehicle cannot use its radio to directly communicate and collaborate with a particular vehicle detected through the camera.

- **Errors in GPS measurements**: Errors in GPS readings make it challenging to associate unique and unambiguous positions to vehicles observed using IVC. Commodity GPS receivers (Standard Positioning Service (SPS)-GPS) have error of 4 meters standard deviation [82] and this error can go beyond several meters in downtown areas due to multi-path effect. Li et al. [110] conducted an experiment in which two GPS devices placed in the same car reported that they are in different lanes in 46% of the cases.

- **Lack of distinguishability in a camera frame**: Vehicles might not be distinguishable in a camera frame due to identical visual features, close spacing or, partial occlusion
by another vehicle. Vehicle tracking errors can also lead to discontinuous or erroneous views of a vehicle.

- **Low adoption rate**: A vehicle can only cooperate with other vehicles that have adopted the same or compatible systems. Schemes that require additional software or hardware will not be adopted by all vehicles instantly. So a practical scheme needs to consider the presence of legacy vehicles.

To address these challenges and support cooperative vehicular applications, we seek to provide a global view of the vehicles on road. Global view is represented by a graph-like rigid structure with nodes representing vehicles and their corresponding positions. Each node is bundled with associated information such as, IP-Address, GPS, color, or possibly destination of the trip (can be used by platooning applications). For building the global-view, the map of vehicles around a vehicle is the building block. Let us refer to this map of vehicles around a vehicle as a local-map. For building the local-map, a vehicle must perform relative vehicle localization and should be able to associate the localized vehicles with their identifiers such as IP-Address, or MAC-Address. Let us refer to the later problem as IP-Address vehicle matching problem which is defined as follows: *In a heterogeneous system adoption environment, given a vehicle which can detect the relative location of its neighboring vehicles by sensors (e.g., camera, RADAR) and can communicate with other vehicles with IVC, how to determine the communication identity of the vehicles detected by the sensors?*

Addressing this problem is significantly important for cooperative vehicular applications. Knowing the relative location of the vehicles that are not in the sensing region expands the sensing region. To address this problem, we designed a system called RoadMap. The key contributions are as follows:
• We have designed a novel algorithm that determines the identities of the neighboring vehicles by exploring the movement pattern of the vehicles along with their visual features.

• We conducted a proof-of-concept experiment for the RoadMap system and observed median matching precision of 80% which is 20% higher than existing schemes.

• We simulated RoadMap with high-fidelity configurations. RoadMap simulated in different traffic scenarios and different system adoption rates outperformed existing schemes.

4.1 System Design

**System Requirements and assumptions:** RoadMap assumes minimal hardware which comprises of a camera, a radio and a GPS receiver. Since a typical smartphone has all these components, RoadMap can be implemented in a smartphone. The low hardware requirement will help to increase the adoption rate of the RoadMap system. RoadMap also accounts for legacy vehicles in its design.

RoadMap uses the camera to detect vehicles. We call a camera-detected vehicle as a visual neighbor, and assign a unique VID (Visual Identity) to it. Note that a VID is only defined locally within the vehicle that detected this vehicle. If two vehicles detected the same vehicle, these two vehicles might assign a different VID in each of their systems. In the following, we assume that each vehicle only has one camera. In fact, multiple cameras facing different directions can be used in one vehicle to expand the viewing area of the vehicle. The radio is used to communicate with neighboring vehicles. The WiFi module of a smartphone and the DSRC [172] technology can be used as the wireless radio. To be discovered by other vehicles, a vehicle will broadcast its identity, GPS location and visual
features to other vehicles. The radio can detect other vehicles by receiving the broadcast information. We call a vehicle received over the radio as an electronic neighbor or an EID (Electronic Identity). Unlike the VIDs, an EID is globally unique. Therefore, different vehicles can easily check if they have common EIDs\(^3\). The GPS receiver can be used to estimate the GPS coordinates of a vehicle.

**RoadMap novel Local Matching (LM) component:** The LM component works as follows: a vehicle in RoadMap periodically uses its camera to detect other vehicles, and uses its radio to broadcast its own ID and related information to allow itself to be discovered by other vehicles. At the same time, the vehicle receives the broadcast information from other vehicles over the radio. LM needs to match the vehicles observed through the camera with the vehicles identified through radio communication. Besides using visible features such as color and shape of the vehicle, LM also detects and tracks the movement history of the vehicles in camera. Each vehicle also broadcasts its own visible features and movement trace. After receiving such information from a vehicle over the radio, LM employs matching algorithms to find similarities between the detected visible information and the information received over the radio. The similarity value is calculated to indicate whether the vehicle received over the radio is one of the vehicles in the visual field. In reality, legacy vehicles can be detected in the camera, and the vehicle received over the radio may not be in the view of the camera. In addition, the camera may have low detection accuracy. LM is designed to work in these scenarios.

### 4.2 Local Vehicle Matching

This section presents the LM algorithm which matches the VIDs and EIDs.

\(^3\)In the remaining chapter, the terms visual neighbor and VID and electronic neighbor and EID are used interchangeably.
4.2.1 Background

Assume a vehicle $C$ has electronic neighbors $E(C)$ and visual neighbors $V(C)$. To identify the IP addresses of the vehicles in $V(C)$, and estimate the relative location of vehicles in $E(C)$, we have to create a match between the two sets of vehicles based on the features of the vehicles. Examples of such features include GPS coordinates and color. In fact, any feature that can be observed or measured by the vehicle itself, and can be observed by its neighboring vehicles can be used in RoadMap. The features of vehicles in $V(C)$ are called visual features, and the features of vehicles in $E(C)$ are called electronic features. The accuracy of these features are limited by the observational variance of the respective sensors. These features are used to find the similarity of an electronic neighbor $e \in E(C)$ with a visual neighbor $v \in V(C)$. Prior works such as Zhang et al. [180] uses camera to help improve the accuracy of wireless localization by comparing the visual distance with electronic distance. ForeSight [111] presents a thorough study of different algorithms to combine visual feature vectors with electronic feature vectors at a given time stamp. It proposes the AdaptiveWeight algorithm to match $E(C)$ and $V(C)$ for vehicle $C$ based on their similarities at a given time. The AdaptiveWeight algorithm will adaptively calculate the weighted mean of the feature similarities to combine different types of features and derive the similarity between a VID and an EID. The weight of each type of feature is calculated based on the distribution of the feature values. For example, if the GPS coordinates of the VIDs are similar, but the colors are distinct, then the color feature is given a higher weight than the GPS feature. Additionally, several works such as [65, 106] have identified the vehicles based on movements and orientation on the road. Nevertheless, these works are limited to visual domain identification of vehicles.
The performance of ForeSight is limited by the lack of temporal information in the system design. The historical matching results are not exploited for the further vehicle matching. Further when the traffic density is high (\(|E(C)|/|V(C)|\) are large), ForeSight performs poorly as there are more number of vehicles matching same feature description.

This thesis exploits the uniqueness of vehicle movement traces when associated with time. All the features from camera like color, position etc., may not uniquely identify a particular vehicle in visual domain. But the recent movement trace of a vehicle is more likely to be unique and can be measured. This thesis takes advantage of the past observations in the visual and electronic domains, and proposes a novel matching algorithm (LM Algorithm) which matches the VIDs and EIDs based on historical movements of vehicles along with the visual features.

### 4.2.2 The Local Matching Algorithm

This section presents the importance of trace similarity compared to other features followed by a description of challenges to extract trace similarities between EIDs and VIDs are presented. Finally, this section presents the LM algorithm that takes the movement traces of the VIDs and EIDs, and computes the trace similarity. Then LM algorithm combines trace-similarity with additional features using Adaptive Weight Algorithm from [111] to get matching result.

**Motivation:** Assume \(e_i, e_j \in E(C)\), and \(v_i, v_j \in V(C)\) corresponds to vehicle-\(i\) and vehicle-\(j\). The movement traces associated with time will provide unique identity of the vehicle, which is observed in electronic and visual domains, by vehicle \(C\). Figure 4.1(a) shows the movement traces of vehicle-\(i\) and vehicle-\(j\) and their relative distances over
Figure 4.1: (a) Traces of vehicle-i and vehicle-j associated with time-stamps to give unique identities in visual and electronic domain. The vehicle pair (i, j) are in different states over a span of 8 time-slots. (b) These traces are extracted as movement traces $T^e_i$ and $T^e_j$ in electronic domain and $T^v_i$ and $T^v_j$ in visual domains. Short-term noise and long-term trend can be observed in each time-series. The trace $T^v_i$ is more similar to $T^e_j$ at time slot-3, but longterm trend of $T^v_i$ is more similar to $T^e_i$ which leads to correct match.
8 time-slots. Based on the distance between the vehicles, the vehicle pair \((i, j)\) can be classified into one of the following states:

- **State-1**: The vehicles are far enough and are distinguishable in both visual and electronic domains. It gives the correct matching result. It corresponds to the green line in the Figure 4.1(a).

- **State-2**: The vehicles are very close and cannot be differentiated in both visual and electronic domains. It corresponds to the pink line in the Figure 4.1(a).

- **State-3**: The vehicles are distinguishable in one domain but not in the other domain. It corresponds to the yellow line in the Figure 4.1(a).

The distances between vehicle pairs \((i, j)\) will increase or decrease with time, depending on the relative velocity between the vehicles. With time, the vehicle pair \((i, j)\) moves from one state to another. If the vehicles are in **State-2** and there is relative velocity between them, then after considerable amount of time, they will be in **State-1**. When the vehicles are in **State-1** they can be differentiated. Similarly, the past information can be used to differentiate the vehicles at current time.

Matching based only on current state will be inaccurate, when the vehicle pairs are in **State-2** since there is no conclusive matching result. But past state (**State-1**) information can be used to differentiate the vehicles. By matching the movement traces which have past positions, one can compute the average distance between these observations over a time interval. Matching two movement traces \(T^e_i\) and \(T^e_j\) in electronic domain with two-time series in visual domain \(T^v_i\) and \(T^v_j\) over a time interval \(t\) will give the average distances between observations. This average distance is more accurate, and so can be used for matching. Figure 4.1(b) shows the movement traces in visual and electronic domains. These
movement traces have two notable characteristics: one is short-term fluctuations introduced by measurement errors, and the other one is long term trends which capture the movement of the vehicles. Matching the movement traces in visual domain with electronic domain over an interval can remove the effect of short term fluctuations due to errors in GPS measurements and events of indistinguishable vehicles in camera frame.

The LM Algorithm: The LM Algorithm (see Algorithm 6) addresses the abovementioned challenges and matches EIDs and VIDs. The information from different inertial sensors can be merged to smooth the traces of EIDs. These sensor data received from broadcast is smoothed using the Kalman filter [96]. Line 6 of Algorithm 6 takes the electronic traces like position, velocity, acceleration and smoothens out these electronic traces. Later, multiple hypothesis tracking (MHT) for multiple target tracking as described in [57] is used by Line 6 to trace multiple vehicles in the visual domain. This step uses the history information to resolve the conflicts of tracking (errors in tracking) a vehicle. Subsequently, line 6 uses exponential moving average to compute the average distance between visual trace and electronic trace. Moving averages are frequently used with time series data to smooth out short-term fluctuations and highlight longer-term trends. The GPS errors and camera errors are short-term fluctuations whereas actual movement trace of the vehicle is a long term trend. The effect of short term fluctuations can be smoothed out by using moving average. Matching error propagation problem is avoided as the erroneous event weight decreases with time, and the current event is given a higher weight. Also in the scenarios of exchanged VID due to tracking errors, LM corrects the matching since it considers the history information for performing matching. The similarity of the two traces based on the average distance is computed in line 6. The similarity in other features are computed by line 6. This trace similarity is combined with the similarities in the other domains using
Figure 4.2: VID-EID matching using movement traces. When all the vehicles are identical their random movements give unique identity based on history.

AdaptiveWeight algorithm mentioned in [111] to give matching result at line 6. Finally, this step helps in identifying two vehicles when they are clearly distinguishable using additional features compared to their motion-traces by giving them more weight. The similarity matrix which is output of the LM algorithm is used to determine the matching by using threshold.

The following example illustrates the matching performed by vehicle A using the moving average algorithm that avoids the effect of short term fluctuation.

Example: Figure 4.2 shows the matching of vehicles B and C done by vehicle A. Vehicle A can hear from both the B and C and also can see them in visual domain. Figure 4.2(a) shows the true movements of A, B and C on the road with time. A which is moving on the left lane observes passing-by vehicles, B and C over 5 time-slots. Figure 4.2(b) shows the visual information and electronic information errors bounds in the system deployed in

\(^4\)A, B and C refers to vehicle A, vehicle B and vehicle C respectively.
Algorithm 6 LM Algorithm

// \(P_v(t)\) Positions of VIDs at time \(t\).
// \(P_e(t)\) Positions of EIDs at time \(t\).
// \(P(t-1)\) Average distance between EIDs and VIDs until time \(t-1\)
// \(f'(t)\) Visual Feature vector at time \(t\).
// \(f''(t)\) Electronic Feature vector at time \(t\).

**Input:** \(P_v(t), P_e(t), P(t-1), \alpha, w, f'(t), f''(t)\)

**Output:** Similarly Matrix \(S\)

// Smoothing the electronic trace
\(P_e(t) \leftarrow \text{Kalman}(P_e(t), V_e(t), A_e(t))\)

// MHT Multi-Hypothesis tracking [57]
\(P_v(t) \leftarrow \text{MHT}(P_v(t))\)

for \(P_v^i(t) \in P_v(t)\) and \(P_e^j(t) \in P_e(t)\) do

if \(t \neq 0\) then

// \(\alpha\) is the weight given to the past event.
\(P_{ij}(t) \leftarrow \alpha P_{ij}(t-1) + (1-\alpha)|P_v^i(t) - P_e^j(t)|\)
else

// Default weight for new EID-VID match
\(P_{ij}(t) \leftarrow w\)
end

end

// \((P_{ij}(t))\) is the average distance between \(i^{th}\) visual trace and \(j^{th}\) electronic trace

for \(i = 1: \text{Size}(P_v(t))\) and \(j = 1: \text{Size}(P_e(t))\) do

// \(S_T(i,j,t)\) is the trace similarity between \(v_i\) and \(e_j\) at time \(t\).
\(S_T(i,j,t) \leftarrow \text{GetSimilarity}(P_{ij}(t))\)
end

for Feature \(f\) do

// Similarity in feature \(f\) at time \(t\).
\(S_f = \text{GetSimilarity}(f'(t), f''(t))\)
end

// Combining similarities across different features
\(S = \text{AdaptiveWeight}(S_f, S_T)\)
A. The system captures the movement history from the broadcast messages to match their respective VID with EIDs. The following observations happen at

- Time $T = 1, 2$: Both $B$ and $C$ are far apart as shown in Figure 4.2(b), are in **State-1**. $A$ can match EIDs with VID correctly.

- Time $T = 3$: The vehicles are very close and cannot be differentiated in visual domain. Based on the GPS, $A$ can deduce relative position of $B$ and $C$, but it cannot map to corresponding visual neighbors. The vehicle pair $(B, C)$ are in **State-3**, By taking the mean distance across the trace, and based on the mean the vehicle pair is converted to **State-1**, thereby the VID are correctly matched to VID.

- Time $T = 4, 5$: Similar to $T = 1, 2$, vehicles are far apart leading to correct match and so are in **State-1**. If there is wrong match in time $T = 3$, due to the current event weight, it will not be propagated in time. This way error propagation can be avoided.

### 4.3 Experiments

In this section, we first introduce our implementation of the vision based vehicle detection and tracking system. The reason for implementing our own vision based vehicle detection system is the lack of a working open-source vehicle detection software. Based on the vehicle detection system, we performed proof-of-concept experiments to evaluate RoadMap.

#### 4.3.1 Vision Based Vehicle Detection

In our vehicle detection system, we mounted a smartphone camera on the dashboard of a vehicle to detect vehicles visible in the camera’s field of view as well as the lane markers on the road. Currently, the system is only designed for daylight condition. Techniques such as rear light based vehicle detection [125] can be implemented in the future to detect vehicles.
at night. The system has three major components: lane detection, vehicle detection and vehicle tracking, which are briefly described below.

**Lane detection:** Lane detection is an effective way to remove many of the false positives that would have normally been detected when searching for vehicles. Lane detection searches for all straight lines in an area in front of the user’s vehicle. We prune the lines that could not indicate a lane marker due to length, location, or angle. The pruned lines are sorted and combined into lane markers based on proximity to each other. This algorithm is able to determine if the lane detected is a solid line or a dashed line. This information allows the vehicle detection algorithm to ignore anything that is outside of solid lines (meaning it is off the road) but still identify vehicles in neighboring lanes.

**Vehicle detection:** Vehicle detection relies on a tiered system of key characteristics to identify vehicles in each frame of the video. The first identifier is used to contrast between the bottom of the vehicle and the road. During preliminary work it was discovered that the region directly beneath the car was significantly darker than the rest of the road even during cloudy conditions. These dark areas provide a region in which more computationally intensive algorithms can be run more efficiently. The region is determined based on the size of the dark area identified. The next identifier determines if the dark areas are symmetric enough to be considered vehicles. This is a successful identifier because rear views of cars are symmetric while tree lines (the most common false positive identified in the previous tier) are not. Next, each region of interest is scanned to determine the edges of objects in the region. From this scan, we try to identify the top of the vehicle. This is accomplished by searching for horizontal lines in a certain portion of the region. This location is predetermined based on the size of the region. The length of each horizontal line is scaled based on its distance from the expected location of the top of the car. This
technique prevents the algorithm from incorrectly identifying the horizon or a bridge as the top of a vehicle simply because it is a long straight line in the region. If no sufficiently long line is found it is determined that the region does not contain a vehicle and is removed. Finally, any remaining regions are returned as containing one vehicle.

**Relative localization:** The distance to the car in front of the user is estimated using lane markers as a measurement. Lane markers on freeways have fixed length (3 foot) and have a 9-foot gap between the lane markers [171]. This standard creates a relationship between vertical position in image and distance in real world. The position of the vehicle in image is estimated based on the relative size of the lane markers.

**Color Estimation:** To estimate the color of a vehicle, we used the k-means algorithm implemented in [70] to cluster the colors on the detected vehicle to find the largest color cluster. The mean of the largest cluster is used as the color of the vehicle.

**Vehicle tracking:** Vehicle tracking provides an added level of confidence that the regions detected are, in fact, vehicles. Tracking is done by comparing the detection results of the neighboring frames. We assign a score for each region that has been identified to be a vehicle. The score increases if the region is identified as a vehicle in the next frame, and decrease otherwise. Only the regions in a frame that have scores above a threshold are reported as detected vehicles.

**Vehicle detection result:** To evaluate our vehicle detection system, we used videos pre-recorded by a Samsung GS4 phone mounted on a car’s dashboard. For each video, we sample 200 frames and examine the detection result by counting the number of correctly detected vehicle, the false positives and the false negatives. Then we compute the precision and recall of the detection result. Based on analysis, the algorithms have achieved a precision of 82% and a recall of 76% for the video recorded under favorable conditions (sunny).
4.3.2 The Proof-of-concept Implementation

**Experiment details:** Evaluating RoadMap using real-world driving requires multiple cars and devices, and the cooperation of the drivers. All such factors make the experiment extremely difficult to conduct. Instead, we performed a small scale driving experiment with three different cars in real-world traffic. The goal of the experiment is to show the performance of RoadMap in real-world scenarios. One car is used as the observer car, and it is always behind the other two cars. We mounted a smartphone in each of the cars, to record the GPS positions. The smartphone in the observer car is also used to record videos (at the default resolution of the smartphone) and GPS coordinates during the driving experiment. The ground truth color of each car is known beforehand. In summary, during the experiment, we recorded the driving videos from the observer car and the GPS coordinates of the three cars. In this experiment, the observer car has two EIDs. The electronic features of the EIDs are the recorded GPS coordinates and the ground truth color. The VIDs are the vehicles detected by our vehicle detection system. We extract the visual features from the video by estimating the color and the relative location of the vehicles.

**Processing:** The obtained traces are processed offline, where the communication across different vehicles is emulated using NS3 simulation. Along with RoadMap, ForeSight is implemented for sake of comparison. ForeSight and RoadMap will match the two EIDs onto some of the detected vehicles in the video. The other two vehicles have no VIDs and their structures will only have themselves. To evaluate the matching result, we randomly selected 200 images from the video and manually labeled the correct matches. Next, we used a varying threshold to eliminate low similarity matching results and obtain different recall rates. In addition, we also tried to match the vehicles by only using GPS coordinates and only using color. Figure 4.3 shows the precision for matching vehicles using LM and
4.4 Simulations

**Simulation set-up:** Evaluating RoadMap with large scale real-world driving requires multiple drivers and vehicles, which makes it difficult to conduct in practice. Instead, we implemented high fidelity simulations using SUMO [102] and NS-3 [161]. SUMO is an open source simulator that can create customized road network and vehicle traffic on demand. NS-3 is a network simulator commonly used to simulate communications between wireless devices. We record the driving traces of the vehicles in SUMO, and simulate each vehicle as a node that moves following the SUMO traces in NS-3. The nodes use 802.11b IBSS mode for communication. Since we mainly compare the performance of our work with ForeSight, we use the same simulation parameters as ForeSight. Colors and GPS
Table 4.1: Different traffic scenarios in the simulation.

<table>
<thead>
<tr>
<th>Traffic Condition</th>
<th>Light 238.13</th>
<th>Medium 349.97</th>
<th>Heavy 749.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. # of vehicles at each time instance</td>
<td>238.13</td>
<td>349.97</td>
<td>749.33</td>
</tr>
<tr>
<td>Avg. # of EIDs (100% adoption rate)</td>
<td>8.59</td>
<td>12.74</td>
<td>28.01</td>
</tr>
<tr>
<td>Avg. # of VIDs</td>
<td>2.39</td>
<td>3.50</td>
<td>6.28</td>
</tr>
</tbody>
</table>

coordinates are selected as the two types of features used in LM for vehicle matching. The same configuration is used for the color detection error model and GPS receiver’s error model.

Simulation details: We first use SUMO to generate a road map that has a square shape. The length of each edge in the square is 2 kilometers, and the total length of the road in the map is 8 kilometers. There are five one-way lanes on each edge, and the speed limit is 50 km/h. Based on this map, SUMO simulates the traffic and logs the position of each vehicle at each time instant (every second). We used three representative traffic scenarios in the simulation: light traffic, medium traffic and heavy traffic. The simulation period is 500 seconds. We skipped the first 200 seconds of the traces because the traffic condition is unstable in the beginning of the traces. Table 5.1 summarizes the basic information for different traffic conditions. These traces are used as input to the NS-3 simulator to simulate the mobility of the vehicles in NS-3.

In NS-3, we install RoadMap on a randomly chosen set of vehicles to simulate different adoption rates. Vehicles that have installed RoadMap will periodically estimate their own GPS coordinates and detect vehicles in LoS. In the simulation, we temporarily set the period to 5 seconds. We modeled the geometric shapes of each vehicle to simulate the visual
blockage. Each vehicle is modeled as a rectangle ($3.8m \times 1.75m$). Cameras are installed in the front center of the vehicle. A vehicle $C$ can only see a vehicle in front of the camera if vehicles rectangle has at least one edge visible from $C$’s position. The vehicle will broadcast their own GPS coordinates and color to its neighboring vehicles. After receiving new EIDs, vehicles with RoadMap will match the VIDs and EIDs using the LM algorithm. We randomly select a fraction of the adopted vehicle as the vehicles that has access to a global server. Such vehicles will send their detection results to the global server after executing the LM algorithm.

**Results:** We have simulated the LM algorithm in different traffic situations and measured for recall and precision. Since not all vehicles on road can have a system deployed, we study the LM for adoption rates of 20% and 80%. Figure 4.4 shows the simulation results with different traffic densities. The F-score is calculated for each pair of recall and precision, and we show the best F-score for each case in the Figure. By exploring the movement signatures, LM performs better than ForeSight. In the light traffic and medium traffic case, the F-score improvements are between 1% and 2%. In the heavy traffic case, the F-score is improved by 3%. One observation is that the F-score decreases as the traffic density increases. From Table 5.1 we can observe that as traffic increases, the the average number of EIDs per VID increases. For the heavy traffic case the number is 4.5, whereas for medium traffic situations is 3.6. As the number of EIDs per VID increases, there will be more VIDs matching the color and position for each VID.
Figure 4.4: The simulation results of the LM algorithm.
4.5 Related Work

DashCalib allows vehicles to find the IP addresses of their neighboring vehicles which is related to works in matching information in different domains and vehicle localization.

**Matching Information in Different Domains:** The LM component has the same objective as ForeSight [111]. ForeSight is the first work that implemented the unicast communication by vehicle matching. However, ForeSight only considers the features available in each time instance. We observed that the movement histories of the vehicles provides a rich set of information that can be leveraged. There are other works that matches information obtained in different domains [112,160,180]. Zhang et al. [180] use camera to help improve the accuracy of wireless localization. The proposed EV-Loc system compares the moving traces obtained by wireless AP with the people’s moving tracing in camera to improve the localization. These works only consider the location features in the matching. RoadMap can automatically obtain the electronic features given the model and color of the vehicle.

**Vehicle Localization:** Many schemes have been proposed for vehicle localization, such as GPS, map matching and dead reckoning. These systems cannot determine the relative location of the legacy vehicles. Devices such as camera and RADAR can be used for relative localization and do not require cooperation of other vehicles. These devices have limited angle-of-view, and can only detect objects in LoS. Schemes based on radio RSSI described by Li et al. [110] and Parker et al. [129] can potentially localize vehicles not in LoS. The problem is that such schemes do not work for legacy vehicles. Recent vehicular relative localization techniques based on vehicle collaborations include [77,97,136]. Fenwick et al. [77] introduced a scheme to allow autonomous vehicles to collaboratively create the road map and localize themselves. Karam et al. [97] and Richter et al. [136] presented schemes.
for relative localization by exchanging GPS and motion estimations. Their systems assume the vehicles have 100% adoption ratio.
Chapter 5: Roadview: Live View of On-Road Vehicular Information

Each year worldwide road accidents lead to USD 518 billion in losses and 1.3 million deaths. An additional 20-50 million are injured or disabled. Intelligent road transportation offers the promise to sharply cut down these numbers and revolutionize how we travel on the roads. More specifically, intelligent navigational and driving control decisions automatically made by vehicles can lead to reduced chance of accidents, stress-free driving, increased passenger comfort, increased fuel-efficiency and reduced travel time. Some of the high-end cars on our roadways are already equipped with various semi-autonomous features. Tesla S is one such car which supports an autopilot mode with features such as driving within a lane, changing lanes, and managing speed by using active cruise control. Recent works such as Foresight [111] and RoadMap [167] have shown the potential of using various sensing modalities such as RADAR, LIDAR, and cameras to build a local-map of neighboring vehicles. In this chapter, we attempt to produce a global information view of vehicles by fusing multiple such local-maps. Roadview is the first work which leverages the sensing capabilities of multiple vehicles to build a collaborative map which also includes legacy vehicles.

The following classes of applications can benefit from such a global map:

- **Traffic statistics based applications**: Existing route planning applications such as Google/Apple maps can benefit from this global map for purposes of road traffic
analytics. Additionally, with live traffic count, traffic deadlocks which are prominent in many major cities can be predicted and the traffic can be efficiently routed to alleviate such situations. Applications such as Automatic Traffic Control (ATC) can benefit from information of incoming traffic. The count of vehicles moving from one location to another is vital for planning enhancements to roadways and public transportation facilities.

- **Enhancing safety**: Imminent accidents can be predicted and prevented. Applications such as Adaptive cruise control can benefit from data observed by surrounding vehicles.

- **Energy efficient route planning**: Different vehicles traveling to the same destination can be grouped together to form a fuel efficient formations such as a vehicle platoon [36].

For the above applications, collaboration can only benefit a vehicle if the relative locations and communication identities of the neighboring vehicles are known. For identifying the communication identities of vehicles which are in the Field-of-View (FoV) of a vehicle’s sensors, different Local Matching Algorithms (LMs [111, 167]) can be explored. The identities of neighboring vehicles can be obtained by exploring QR-Codes, Ultrasonic communication, Visual Light communication, MIMO with Wi-Fi, radio RSSI [110,129], and visual features (color, aspect ratio, Scale-invariant feature transform (SIFT) features [131]). By employing such techniques the identity of vehicles in FoV along with their relative locations can be obtained. Techniques such as Foresight [111] or RoadMap [167] can be applied to improve the accuracy in localizing and identifying neighboring vehicles. However such techniques can only provide the map of vehicles in the FoV of the sensors in the vehicle.
The sensing region of a vehicle can be enhanced by fusing local maps created by individual vehicles.

However, solutions for fusing the observations from multiple such vehicles have the following challenges:

- **Incomplete local maps due to limited FoV:** The maps produced by individual vehicles are limited by the vehicles they observe using the onboard sensors. Additionally, techniques based on radio RSSI [110, 129] cannot localize non-Line-of-Sight (NLoS) vehicles. Consequently, the local maps may be incomplete which makes it non-trivial to fuse them.

- **The presence of legacy vehicles:** The legacy vehicles may be observed by sensors such as a camera, but they will not be observed in the electronic domain (no messages from such a vehicle).

- **Conflicting observations or errors in LM:** The local maps created at individual vehicles may be inconsistent due to errors in matching vehicles or due to the presence of legacy vehicles.

Thus the problem of Global Matching (GM) is defined as follows: *Given legacy vehicles, time-varying traffic densities, incomplete local maps, and inconsistent local maps, how can the local maps created at the participating vehicles be fused to produce an accurate global map of vehicles?*

We propose Roadview, a system that can provide a global view of the vehicles on road. Roadview works on top of LM algorithms (Foresight [111], RoadMap [167] etc.) and uses novel Global Matching (GM) algorithm to generate a global view of vehicles on road. We call a vehicle, that reports its LM outcome to the server, a reporter. The outcome of
the LM component contains the visual neighbors, electronic neighbors and the matching between these two sets of vehicles. GM maintains a graph-like global structure in which each node represents a physical vehicle, and the edge between two nodes represents the relative location of the two nodes. An edge exists between a pair of nodes only if the relative location between the vehicles has been reported by at least one of the reporters. For each received LM outcome, GM first creates a star-like structure, where the center node is the reporter itself, and the satellite nodes are the visual neighbors of the reporter. There are edges between all pairs of nodes in the structure. GM will merge the created structure with the global structure using a modified solution of the maximum common subgraph problem. The idea is to join the two structures based on their overlaps. After merging the structures, the global structure can have more nodes and edges added. The global structure contains the relative locations between vehicles, and the global identity of each vehicle. It can also be used to correct the errors in the LM’s outcome. GM is an incremental algorithm. By this design, we do not require all reporters to submit their LM outcome at the same time, and can provide real-time response to the reporters. The contributions of this work are as follows:

- First work to study the challenges involved in building a global information view of the road.

- Proposes a novel GM algorithm that enhances the capability of vehicles to sense 1.8x (average) more number of neighboring vehicles compared to state of the art LM algorithms. Note these neighboring vehicles may not be in FoV of vehicular sensors.

- Evaluates the system with extensive trace-driven simulations and different LMs.
5.1 System Design

Roadview divides the map into road segments, a concept commonly used in digital maps. A road segment represents a portion of a road with uniform characteristics. A road segment has no intersections and contains one or more one-way lanes.

We do not assume that all vehicles have adopted the Roadview system. For easy adoption, Roadview has minimal hardware requirements that consists of a camera, a radio, and a GPS receiver. Since a typical smartphone has all these components, Roadview can be implemented in a smartphone.

Let us refer to a camera-detected neighboring vehicle as a visual neighbor, and assign a unique vehicle-specific VID (Visual Identity) to it. Note that a VID is only defined locally
by a reporter. If two reporters detected the same vehicle, they will each assign a different VID in each of their systems. Each vehicle advertises its globally unique electronic identity (EID), e.g., its MAC address, along with some visual [111] and kinematic signatures [167].

5.1.1 Background on Graph Matching

In the graph theory literature, given two graphs \( G \) and \( H \), the association graph \( S \) is created as follows [58, 128]. The vertices of \( S \) correspond to the vertex-pairs \((u, v)\), where \( u \in G \) and \( v \in H \). Vertex \((u, v)\) represents the option of matching \( u \in G \) and \( v \in H \). Therefore the number of vertices in \( S \) is \(|G| \times |H|\). The edges in \( S \) are defined based on the connectivity of \( G \) and \( H \). Assume \( E(G) \) and \( E(H) \) are the edges of \( G \) and \( H \), correspondingly, and \((u_1, v_1)\) and \((u_2, v_2)\) are two vertices in \( S \). There is an edge between the vertex \((u_1, v_1)\) and \((u_2, v_2)\) if and only if one of the two conditions is satisfied: i) \((u_1, u_2) \in E(G) \) and \((v_1, v_2) \in E(H)\); or ii): \((u_1, u_2) \notin E(G) \) and \((v_1, v_2) \notin E(H)\). The way of creating an association graph \( S \) captures the topology constraints when searching for the common subgraph between \( G \) and \( H \). The maximum common subgraph between \( G \) and \( H \) can be found by finding the maximum clique in association graph \( S \).

5.1.2 Solution Overview of Roadview

Roadview has two components: the local matching (LM) component and global merging (GM) component. The objective of the LM component is to match the vehicles detected by camera, with the vehicles that are learnt from the messages received over the radio. The objective of the GM component is to collaboratively create a unique view of the vehicles on the road based on the reported detection results from LM. The LM component is distributed while the GM component is centralized. GM depends on the outcome of LM. Since GM requires having access to a global server, GM does not assume all vehicles will report their
LM outcome to GM. This increases robustness and flexibility. The system architecture is depicted in Figure 5.1.

We have evaluated the GM algorithm with LM algorithms presented in Foresight [111] and RoadMap [167]. After receiving such information from a vehicle over the radio, LM employs matching algorithms to find the similarities between the detected visible information and the information received over the radio. The similarity value is calculated to indicate whether the vehicle learned over the radio is one of the vehicles in the camera’s view. In practice, it is possible that a legacy vehicle is detected by the camera and the vehicle learnt over the radio is not in view of the camera.

Roadview creates a global view represented by a graph-like structure (say, $G$). The concept of a structure is commonly used in the computer vision field. Like a graph, there are nodes and edges in a structure, but the edges have fixed orientations in a n-dimensional space (here, we consider n=2) and lengths. In $G$, each node represents a physical vehicle, and each edge represents the relative orientation and the distance between two vehicles. Roadview builds this global view by employing a novel Global Matching algorithm which incrementally combines the new LM result from a reporter (say, vehicle A) with the Global map ($G$) as follows:

1. **CreateStructure**: It creates a structure $M$ from the output of LM. Roadview fuses $M$ with $G$ by creating an association graph between $M$ and $G$.
2. **CreateAssociationGraph**: It creates a graph which has nodes representing all potential associations between the visual neighbors of a vehicle and the nodes in $G$. Each node in the new association graph represents a pair of nodes, where one is a visual neighbor of the reporting node and the other is an existing node in $G$.
3. **FindMaximumWeightedClique**: It finds a maximum weighted clique in the association graph by defining the weight based on the following two notions of
similarity: (i) **NodeSimilarity**: quantifies similarity between two nodes in an association graph based on adaptive weight algorithm [111, 167]. (ii) **EdgeSimilarity**: is a metric quantifying the association between pairs of vertices based on the rigidity of $\mathcal{G}$. These two similarities are combined adaptively to find the maximum weighted clique. This step leverages feature similarity matrices of vehicle $A$ and Global view $\mathcal{G}$. Thus this step resolves any *conflicting observations* by giving more weight to more accurate matching.

Essentially, steps (2) & (3) leverage similarity with a modified version of the *maximum common subgraph problem* for obtaining maximum overlap based on underlying LM results to resolve *conflicting observations*. Finally, the maximum clique is added to the Global map $\mathcal{G}$ and this process is repeated whenever a vehicle adds its LM result to $\mathcal{G}$. Note that a vehicle that has not adopted the Roadview system can also appear in $\mathcal{G}$ if it is detected by other vehicles. Based on $\mathcal{G}$, a vehicle can identify the relative location of another vehicle, and find its identity (IP address) if it has adopted the system.

### 5.2 Global Vehicle Merging

#### 5.2.1 Motivation

The LM algorithms [111, 167] focus on exploring the features associated with each vehicle to perform vehicle matching. The detection result of a vehicle $C$ from LM algorithms [111, 167] contains visual neighbors and electronic neighbors represented as $V(C)$ and $E(C)$ respectively, and the matching between vehicles in $V(C)$ and $E(C)$. We observed that merging the detection results of neighboring vehicles can help each vehicle to identify and localize more vehicles. In addition, it can potentially correct the matching results of individual vehicles. Here we present two examples. In the first example, a vehicle $C$ is not able to match a VID $D$ because $D$ is far away from $C$. If $C$ has a correctly matched neighbor
A's matching 

B's matching 

C's matching 

Figure 5.2: An example of correcting matching errors with collaboration. The circles in the dotted rectangle represent the relative locations and matching result of the three vehicles. Merging their matching results can correct C’s incorrect matching result.

E that is located between C and D, then E could help C to match D. The second example is illustrated in Figure 5.2, where each vehicle can observe the other two vehicles, and vehicle C has incorrect matching result. If vehicles A and B forward their matching results to C, then C can find that there is a conflict between C’s matching result and the other matching results. The two examples show that if we have access to the detection results of multiple vehicles, we will have more opportunities to discover neighboring vehicles and increase the accuracy of identified vehicles.

The GM algorithm does not assume that it has the detection results from all vehicles in the road segment. There are several reasons that a vehicle may not be able to report its matching results to the global server: the vehicle does not have the Roadview system; or the vehicle has the system but does not have network connectivity, or the vehicle does not
Figure 5.3: An example of the GM algorithm. Initially vehicle C has three VIDs \{c_1, c_2, c_3\}, and G has five nodes. The red dotted nodes and edges in Structure M and G indicate the same sub-structure shared by M and G. In Step2, we assume only vertex-pairs (C, g_2), (c_1, g_4), (c_1, g_5) and (c_2, g_3) have similarities that satisfy the constraints in Algorithm 7. Therefore, only four nodes exist in the association graph A.

have matching result to report. Here we summarize the challenges in merging the detection results:

- Vehicles cannot directly compare whether they have common VIDs. In the first example, C and E cannot guarantee that they are matching the same VID D. E’s matching can be incorrect if D has not adopted the Roadview system.

- The ad hoc approaches introduced in the two examples only apply to scenarios when specific conditions are satisfied. It is challenging, to enumerate all scenarios in which conflicts can happen, especially when detection results from multiple vehicles are used.

- When comparing the detection results of multiple vehicles, the conflicts could be correlated. Correcting one conflict could introduce other conflicts.
5.2.2 The Structures Used in GM

A structure $\mathcal{S}$ has a set of nodes $N(\mathcal{S})$ and edges $E(\mathcal{S})$. Each node $n \in \mathcal{S}$ has a set of VIDs and EIDs, denoted by $V_n$ and $E_n$, respectively.

As introduced before, the Local Matching (LM) result of $C$ contains $V(C)$, $E(C)$ and a set of matching pairs $M(C) = \{(e, v)\}$, where pair $(e, v)$ indicates that $e \in E(C)$ is matched to $v \in V(C)$. If we draw $C$ and $C$’s visual neighbors $V(C)$ on a 2D plane using the GPS coordinate system, we can get a star-like structure, where the center node is $C$, and the satellite nodes are $V(C)$. We create structure $\mathcal{M}$ based on the detection result of $C$. Then we mark node $C$ as the reporter node in $\mathcal{M}$, because $C$ reports matching ($M(C)$) to the global server. Note that in $\mathcal{M}$ each node has only one VID and at most one EID except node $C$. The EIDs of node $C$ which are not matched to any EID by LM are excluded from creating Global view. Creating the structure $\mathcal{M}$ from matching result $M(C)$ is implemented in CreateStructure method. This structure $\mathcal{M}$ is used to update the global view $\mathcal{G}$.

5.2.3 Creating the Association Graph

In this section, we introduce the key techniques used in merging the structures. We create an association graph $\mathcal{A}$ based on two structures $\mathcal{M}$ and $\mathcal{G}$, then find the maximum weighted clique $C$ in $\mathcal{A}$. The maximum weighted clique $C$ indicates the overlapping structure between $\mathcal{M}$ and $\mathcal{G}$.

In our case, given two structures $\mathcal{M}$ and $\mathcal{G}$, we need to enforce the constraints of the structures when creating the association graph. Algorithm 7 creates a weighted association graph, in which each edge and node has a weight represented by a real number in $[0, 1]$. These weights are created based on two functions NodeSimilarity and EdgeSimilarity.
NodeSimilarity(u, v) signifies the similarity between two nodes. Note the association graph is created between \( \mathcal{M} \) and \( \mathcal{G} \), therefore \( u \in N(\mathcal{M}) \) and \( v \in N(\mathcal{G}) \). First Roadview creates two centroid nodes \( u' \) and \( v' \) for \( u \) and \( v \), respectively in \( n \)-dimensional feature space where \( n \) is number of features used by LM. Some example features are color of the vehicles, aspect ratio, kinematic signatures. The centroid node \( u' \) is created by using the mean of the feature values of the VIDs in \( V_u \) and the EIDs in \( E_u \). \( v' \) is created in the similar way based on \( V_v \) and \( E_v \). If \( \mathcal{F} \) is the set of features used by Roadview, for \( i^{th} \) feature \( f_i \in \mathcal{F} \), \( u' \)'s value of feature \( f_i \) is \( u'[i] = \text{mean}(f_i, f_i \in \{V_u \cup E_u\}) \), and \( v' \)'s value of feature \( f_i \) is \( v'[i] = \text{mean}(f_i, f_i \in \{V_v \cup E_v\}) \). Then NodeSimilarity uses the AdaptiveWeight (AW) algorithm [111] to compute the similarity between \( u' \) and \( v' \). Note AW algorithm fuses different features by allocating weights based on distinguishability of the feature. For example, if color feature is more distinguishable, AW allocates more weight to the color feature for computing similarity between nodes.

EdgeSimilarity((\( u_1, u_2 \)), (\( v_1, v_2 \))) is implemented by computing the difference of the feature values of the mean nodes. We first create the centroid nodes \( u'_1, u'_2, v'_1 \) and \( v'_2 \) based on \( u_1, u_2, v_1 \) and \( v_2 \), respectively. Then we create feature difference vector \( w = (u'_1 - u'_2) - (v'_1 - v'_2) \), where the minus sign means subtracting corresponding feature values of the nodes. Then EdgeSimilarity = \( 1.0 - \min \left( 1.0, \sqrt{\frac{\sum_{f \in \mathcal{F}} w_f^2}{\sum_{f \in \mathcal{F}} \sigma_f^2}} \right) \), where \( \sigma_f \) is the standard deviation of feature \( f \). This heuristic captures the similarity between two edges in the association graph. If the distance between features is greater than the variance of standard deviation of the feature then the EdgeSimilarity is 0, signifying different edges. Our simulation shows that the similarities between the nodes and edges in the structures are well-captured by this heuristic approach.
Constrains on association graph: Different from the commonly used method of creating the association graph, Algorithm 7 imposes the following three extra constraints:

- **Reporters are not merged:** Different reporters represent different vehicles. We do not create an association node when both the corresponding pair of nodes are reporter nodes (Line 7). In this way, we exclude the case that two different reporter nodes are merged into the same node.

- **Threshold on NodeSimilarity:** We create an association node only if the NodeSimilarity is larger than a threshold $\tau_1$ (Line 7). It excludes matching nodes that are completely different.

- **Threshold on EdgeSimilarity:** We do not create an association node only if the EdgeSimilarity is larger than a threshold $\tau_2$ (Line 7). It indicates that to merge $u_1$ with $v_1$ and $u_2$ with $v_2$, edge $(u_1, u_2)$ and edge $(v_1, v_2)$ should have similar orientation and length.

These constraints significantly reduce the number of nodes and edges in the created association graph, which directly reduces the computational complexity of finding the maximum weighted clique. Therefore, NodeSimilarity and EdgeSimilarity can affect the computing time of the algorithm. Figure 5.3 shows one example of creating the association graph based on $\mathcal{M}$ and $\mathcal{G}$. Assuming Line 7 in Algorithm 7 allows matching $C$ with $g_2$, $c_1$ with $g_4$ or $g_5$ and $c_2$ with $g_3$, the association graph $\mathcal{A}$ will only have four nodes and two three-node cliques. Note that the node pairs in each vertex of $\mathcal{A}$ indicate the matching options.
Algorithm 7 Create the Association Graph

**Input**: \(G, M\)

**Output**: Association Graph \(A\)

\[ A \leftarrow \emptyset \]

for each vertex-pair \((u, v)\), where \(u \in N(G), v \in N(M)\) do

if not (u and v are reporters, and they are different) then

\[ s \leftarrow \text{NodeSimilarity}(u, v) \]

if \(s > \tau_1\) then

\[ N(A) \leftarrow N(A) \cup \{(u, v)\} \]

weight\((\text{Node}(u, v))\) \(\leftarrow s\)

end

end

end

for \((u_1, v_1) \in N(A), (u_2, v_2) \in N(A)\) do

if \(u_1 \neq u_2\) and \(v_1 \neq v_2\) then

if \((u_1, u_2) \in E(G)\) and \((v_1, v_2) \in E(G)\) or \((u_1, u_2) \notin E(G)\) and \((v_1, v_2) \notin E(G)\) then

\[ s \leftarrow \text{EdgeSimilarity}((u_1, u_2), (v_1, v_2)) \]

if \(s > \tau_2\) then

\[ E(A) \leftarrow E(A) \cup \{(u_1, v_1), (u_2, v_2)\} \]

weight\((\text{Edge}((u_1, v_1), (u_2, v_2)))\) \(\leftarrow s\)

end

end

end

end
5.2.4 The Global Merging (GM) Algorithm

In this section, we show how GM merges the structures based on the concept of association graph. The GM algorithm is an incremental algorithm. Initially, $G$ is empty. When a detection result $M(C)$ from vehicle $C$ is received, GM will convert $M(C)$ into a structure $M$, and merge the structure with $G$ based on the overlaps between them. We denote the merged structure as $G'$, $C$ can request to receive $G'$ or part of $G'$ based on $C$’s interest.

Matching $G$ and $M$: Algorithm 8 shows the detailed procedure of the GM algorithm. In Algorithm 8, we first convert the detection result $M(C)$ into a structure $M$ (Line 8 in Algorithm 8). If $G$ is not empty, we use Algorithm 7 to create association graph $A$ based on $M$ and $G$ (Line 8). After creating the association graph, the problem is reduced to finding the maximum weighted clique in graph $A$. In GM, the maximum weighted clique is defined as the clique in $A$ that maximizes the total weight of the nodes and the edges. Finding the maximum weighted clique in an arbitrary graph is an NP-hard problem [58, 128]. Any maximum weighted clique detection algorithm can be applied in Line 8. In our simulation, we implemented the pivoting version of the Bron-Kerbosch Algorithm [61] due to the simplicity in implementing it. The time complexity of this algorithm is $O(3^{n/3})$ [63].

Fusing matching results of $G$ and $M$: After finding the matched node pairs based on the clique, we save the VIDs and EIDs associated with the nodes in $M$ to the corresponding nodes in $G$ (Line 8-8). In this way, we combine the matching result in $M(C)$ with the matching results merged into $G$ previously. For each node in $M$ that has a matched node in $G$, we find the representative EID of the corresponding node in $G$ by combining all the related matching results reported to $G$. This representative EID is assigned to match with the only VID in $M$’s node (remember that there is at most one VID in each of $M$’s node). In the algorithm, we use the VoteForEID procedure to detect the representative EID of a node.
Algorithm 8 Merge \( \mathcal{G} \) with detection result \( M(C) \)

\[
\begin{align*}
\text{Input} & : \mathcal{G}, M(C) \\
\text{Output} & : \mathcal{G}' \\
\mathcal{M} & \leftarrow \text{CreateStructure}(M(C)) \\
\text{if} & \ G = \Phi \ 	ext{then} \\
\mathcal{G}' & = \mathcal{M} \\
\text{else} & \\
\mathcal{A} & \leftarrow \text{CreateAssociationGraph}(\mathcal{G}, \mathcal{M}) \\
\mathcal{C} & \leftarrow \text{FindMaximumWeightedClique}(\mathcal{A}) \\
\text{matchedNodes} & \leftarrow \{ \} \\
\text{for} \ (u,v) \in \mathcal{C}, \text{ where } u \in N(\mathcal{G}) \text{ and } v \in N(\mathcal{M}) \ 	ext{do} \\
\quad & \text{// Add the VID and EID associated with } v \text{ to the VID set and EID set of } u. \\
\quad V_u & \leftarrow V_u \cup V_v \\
\quad E_u & \leftarrow E_u \cup E_v \\
\quad & \text{// Update the EID set of node } v \text{ in } \mathcal{M} \\
\quad E_v & \leftarrow \text{VoteForEID}(u) \\
\quad \text{matchedNodes} & \leftarrow \text{matchedNodes} \cup \{v\} \\
\text{end} \\
\text{// Add } N(\mathcal{G}) \text{ and the un-matched nodes to } \mathcal{G}' \\
N(\mathcal{G}') & \leftarrow N(\mathcal{G}) \cup \{N(\mathcal{M}) \setminus \text{matchedNodes}\} \\
\text{// Add corresponding edges in } \mathcal{M} \text{ to } \mathcal{G}' \\
E(\mathcal{G}') & \leftarrow E(\mathcal{G}) \cup E(\mathcal{M}) \\
\end{align*}
\]

in \( \mathcal{G} \) (Line 8). A node \( u \in \mathcal{G} \) could contain multiple associated VIDs in \( V_u \) and multiple associated EIDs in \( E_u \). \text{VoteForEID} will find the EID for VIDs in \( V_u \) based on the following four rules.

1. If \( u \) has a reporter, then the VIDs in \( V_u \) are matched with \( u \). \text{VoteForEID} returns the reporter of \( u \). We will correct the EIDs \( e \in E_u \) if \( e \) is not the same as the reporter of \( u \).

2. If \( E_u = \Phi \), it means that there is no EID can match with the VIDs in \( V_u \). \text{VoteForEID} will return nothing. It indicates that \( u \) represents a legacy vehicle.
3. If $|E_u| = 1$, all the VIDs in $V_u$ are matched to the only EID in $E_u$, which is the output of VoteForEID.

4. If $|E_u| > 1$, we create a VID $v'$ that is the centroid of the VIDs in $V_u$. VoteForEID returns the EID that has the maximum similarity with $v'$.

**Improving LM matching result by GM:** In Items 1, 3 and 4, if any reporter or EID have been selected, we match the VID of the node in $\mathcal{M}$ with the selected EID. It can potentially improve the matching recall and precision of $\mathcal{M}$. The un-matched nodes in $\mathcal{M}$ are also added to $\mathcal{G}'$ (Line 8). In the future matching process, these un-matched nodes can potentially be matched with the nodes in the new structure. Line 8 adds the edges in structure $\mathcal{M}$ to $\mathcal{G}$. This is an important step as it creates connections between the nodes in $\mathcal{G}$ and the node newly added by $M(C)$, which can let the existing nodes learn the relative location of vehicles that does not exist in their list of VIDs. Therefore, after the matching, structure $\mathcal{G}'$ is also valuable for vehicles who have previously submitted their detection results before $C$. In Figure 5.3, we assume the maximum weighted clique $C$ is the clique with nodes $\{(c_1, g_5), (C, g_2), (c_2, g_3)\}$. $C$ indicates matching $c_1$ with $g_5$, $C$ with $g_2$ and $c_2$ with $g_3$. We merge these node pairs, and finally add the un-matched node $c_3$ into $\mathcal{G}$ to create the merged structure $\mathcal{G}'$. Note that the edge between $c_3$ and $g_2$ is one of the edges that do not exist in $\mathcal{G}$. It indicates that by merging the detection result $M(C)$, vehicles associated with node $g_2$ can discover the relative location with the vehicle associated with $c_3$. In our simulation, we examine the degrees of the nodes in $\mathcal{G}$ to indicate how GM helps the vehicles to discover extra neighbors.

The feature values of the VIDs and EIDs will continually keep on changing. The existing values saved in $\mathcal{G}$ need to be updated. To address this problem, the GM algorithm records the time-stamp when the VIDs and EIDs are merged into $\mathcal{G}$. GM uses time alignment
techniques [119] to update the state of the vehicles, based on the speed and the map of the road. Upon invocation, GM removes the VIDs and EIDs that are merged into $\mathcal{F}$ more than $\tau$ seconds ago.

5.3 Simulations

**Simulation set-up:** Evaluating Roadview with large scale real-world driving requires multiple drivers and vehicles, which makes it difficult to conduct in practice. Instead, we implemented high fidelity simulations using SUMO [102] and NS-3 [161]. SUMO is an open source simulator that can create customized 2D road network and vehicle traffic on demand. NS-3 is a network simulator commonly used to simulate communications between wireless devices. We record the driving traces of the vehicles in SUMO, and simulate each vehicle as a node that moves following the SUMO traces in NS-3. The nodes use 802.11b IBSS mode for communication. Since we mainly compare the performance of our work with ForeSight [111] and RoadMap [167] we use the same simulation parameters as mentioned in [111, 167]. Colors and GPS coordinates are selected as the two types of features used in LM for vehicle matching. The same configuration is used for the color detection error model and GPS receiver’s error model.

**Simulation details:** We first use SUMO to generate a road map that has a square shape. The length of each edge in the square is 2 kilometers, and the total length of the road on the map is 8 kilometers. There are five one-way lanes on each edge, and the speed limit is 50 km/h. Based on this map, SUMO simulates the traffic and logs the position of each vehicle at each time instant (every second). We used three representative traffic scenarios in the simulation: light traffic, medium traffic, and heavy traffic. The simulation period is 500 seconds. We skipped the first 200 seconds of the traces because the traffic condition
is unstable at the beginning of the traces. Table 5.1 summarizes the basic information at different traffic conditions. These traces are used as input to the NS-3 simulator to simulate the mobility of the vehicles in NS-3.

In NS-3, we install Roadview on a randomly chosen set of vehicles to simulate different adoption rates. Vehicles that have installed Roadview will periodically estimate their own GPS coordinates and detect vehicles in LoS. In the simulation, we temporarily set the period to 5 seconds. We modeled the geometric shapes of each vehicle to simulate the visual blockage. Each vehicle is modeled as a rectangle $(3.8m \times 1.75m)$. Cameras are installed in the front center of the vehicle. A vehicle $C$ can only see a vehicle in front of the camera if its rectangle has at least one complete edge visible from $C$’s camera position. The vehicle will broadcast its own GPS coordinates and color to its neighboring vehicles. After receiving new EIDs, vehicles with Roadview will match the VIDs and EIDs using the RoadMap algorithm. We randomly select a fraction of the adopted vehicle as the vehicles that have access to a global server. Such vehicles will send their detection results to the global server after executing the RoadMap algorithm.
5.3.1 Evaluation of the GM Component

In this section, we focus on evaluating the performance of the GM component and examine the properties of the global structure $G$. Although GM is implemented on top of RoadMap, we only label GM in the following figures because the legend space is limited. $G$ contains the relative locations and the IP addresses of vehicles that have adopted the Roadview system. The percentage of the reporter vehicles among the adopted vehicles is denoted by $r$. We select $r = 20\%$ and $r = 80\%$ as two representative cases in the simulation to show how it affects the performance of the GM algorithm.

**Enhanced sensing by GM:** Unlike LM, GM allows a reporter to discover the relative locations of vehicles that it may not be able to detect through its camera. The degree of a reporter node $C$ in $G$ indicates the number of visual neighbors of $C$, plus the number of vehicles that are added by the vehicles matched with $C$. The degree of a node represents the number of immediate neighboring vehicles that have known relative locations. Figure 5.4(a), 5.4(b) and 5.4(c) show the average degree of the reporter nodes in $G$ for different adoption rates. Note that the average number of VIDs only depends on the traffic condition, and does not change as the adoption rate increases. On the other hand, the average number of EIDs increases linearly with the adoption rate. As we have expected, as $r$ increases from 20% to 80%, the degree of the nodes increases. One interesting observation is that when the adoption rate is larger than 50%, the average degree stops increasing and stays close to $2 \times$ of the average number of VIDs. Figure 5.4(d) depicts enhancement to the sensing capability achieved by the GM algorithm for different traffic densities compared to LM algorithms. Figure 5.4(d) depicts this enhanced sensing of 1.8x times for light traffic, 1.6x for medium traffic 1.3x for heavy traffic scenarios. In the simulation, each adopted vehicle only has one camera facing front. $2 \times$ the average number of VIDs is roughly the average number of...
VIDs a vehicle could observe if it has one front-facing camera and one camera facing back. By collaboration, GM discovers neighboring vehicles not in the view of the cameras and significantly increases the number of neighboring vehicles with known relative locations. This is extremely useful for applications such as blind-spot detection. At the same time, GM maintains high matching precision, recall and F-score.

![Graph showing the number of vehicles sensed by different algorithms in light, medium, and heavy traffic conditions.](image)

Figure 5.4: The number of vehicles sensed by different algorithms (average degree of the reporter nodes in global structure \( G \)). GM improves sensing by a factor of 2.

**Computational Intensity of GM:** Finally, we study the size of the association graph \( \mathcal{A} \) and the clique \( \mathcal{C} \) in different traffic conditions. We assume the adoption rate is 100%, and all vehicles are reporters, which is the most compute-intensive setting. We use \( |N(\mathcal{A})| \) and \( |E(\mathcal{A})| \) to denote the number of nodes and edges in association graph \( \mathcal{A} \), and use \( |N(\mathcal{C})| \) to denote the number of nodes in clique \( \mathcal{C} \). Although the clique detection problem is an NP-hard problem, GM can significantly reduce the size of the problem and work efficiently. Heavy-traffic scenarios are the most compute-intensive. The average value of \( |N(\mathcal{A})| \) is
21.7 and the average value of $|E(\mathcal{A})|$ is 9.3. In medium traffic and light traffic scenarios, the size of the association graph is even smaller.

### 5.4 Related Work

Roadview enables vehicles to find the IP addresses of their neighboring vehicles, and it can combine the matching results into a global view of the vehicles on road. There are related works in matching information in other domains and graph matching.

**Matching Information in Different Domains**: Roadview uses the Adaptive weight algorithm for fusing the $\text{NodeSimilarity}$ which signifies the similarity between two nodes used by GM in fusing a reporter node with the existing GM structure. The adaptive weighted algorithm is employed by on-vehicular matching systems such as Foresight [111] and RoadMap [167]. Similarly, adaptive weighted algorithms are employed by [163, 165, 166] for vehicular to infrastructure (V2I) pairing the vehicles observed over camera (VIDs) with their respective EIDs. In contrast, Roadview also uses novel metric $\text{EdgeSimilarity}$ which signifies the similarity between edges in Global-map $\mathcal{G}$ and new detection result $M(C)$. The metric explores the rigidity of the vehicular map structure to improve matching results and minimizing the errors in combining new detection results.

**Graph Matching**: The GM algorithm merges the matching results of individual vehicles. Related works include graph matching and jigsaw puzzle matching problem. The graph matching problem can be reduced to the maximum clique problem by creating the association graph, which is an NP-hard problem [58]. [128] and [58] have detailed survey of the maximum clique problem. In 2001, [137] designed an algorithm that can find the maximum clique with time complexity $O(2^n/4)$. This is currently the best known result. Roadview has the freedom to employ any maximum weighted clique detection algorithm. We enforce
three restrictions when creating the association graph, which significantly reduces the size of the association graph and computational complexity.

GM combines different pieces of information to create the global structure, therefore, our problem has similarity with the image stitching problem and the jigsaw puzzle problem. The image stitching problem [127] needs to discover the correspondence relationships among images with varying degrees of overlap. It is used in video stabilization and the creation of panoramic images. In our problem, we need to identify the EID of the vehicles. Besides the jigsaw puzzle games, the jigsaw puzzle problem is also applied in document and archaeological artifact reconstruction [99]. Solutions of the jigsaw puzzle includes matching the share, edges, patterns or colors of the non-overlapping pieces to reconstruct the global picture. In our case, the detection results cannot be represented by non-overlapping pieces. In GM, we created a star-like structure for the detection results.

5.5 Conclusion

Roadview is a system that builds a live map of the surrounding vehicles by collaboratively fusing local maps created by vehicles. Roadview layers on top of local matching algorithms such as Foresight [111] or RoadMap [167] and improves sensing capabilities of vehicles by a factor of 1.8x. Roadview can work even at low adoption rates and can also map the legacy vehicles. The extended sensing range can benefit collaborative vehicular applications related to traffic statistics, safety by accident prediction and prevention, and energy efficient route planning.
Chapter 6: Soft-Swipe: Enabling High-Accuracy Pairing of Vehicles to Lanes using COTS Technology

Smartphone-based payments are becoming the new normal as evidenced by the ubiquitous nature of mobile payment systems such as Google Wallet and Apple Pay [37, 42]. Banks such as Mastercard and Visa are already working closely with a number of handset developers to make it widely available [45]. These solutions work for a few centimeters of range [72], which provides a level of security to the transaction. But, the ability to communicate over longer distances can lead to reduced service time and it can open up opportunities for many new applications.

In this chapter we explore applications in which interactions originate from within a vehicle. Transacting from within a vehicle can lead to shorter wait times and higher system throughput. Further, in many situations, the user would be thankful for reduced exposure to inclement weather conditions. The applications can be broadly categorized as follows. **Class-I (Temporary infrastructure):** Parking payments for temporary events such as football games, concerts, fairs etc. are usually processed manually (both payer and payee) and easily lead to heavy backlog in traffic whose effect can extend for several miles. **Class-II (Small-scale infrastructure):** Application scenarios where the infrastructure is owned by small players can be categorized as follows: 1) **Vehicle-specific services:** Payment for services such as car-wash, automated fueling, automated swapping of car batteries for
Electric Vehicles (EVs), automated battery charging centers for EVs, and parking charges can be made from within the vehicle. In an automotive manufacturing plant, a vehicle arriving at a manufacturing station needs to be identified correctly so that the appropriate set of tests can be conducted and the appropriate actions can be taken by the assembly line robots or humans. 2) User-specific services: Payment for drive-thru services such as fast-food or DVD rental can be supported by such a system. A bank customer can perform automatic verification from inside the vehicle before reaching the ATM machine. Today for such applications, usually the payer stops the vehicle to use a machine to make the payment. Class-III (Large-scale infrastructure): Highway toll collection systems can afford to deploy various types of expensive equipment such as directional RFID readers, laser sensors, and inductive loops. Widely used examples of such systems include E-Z Pass [38], Fastrack [41] and I-PASS [43]. Advanced systems on many US highways do not even require the vehicles to slow down when passing through such checkpoints.

Although for Class III applications a number of solutions are already in place, there are few solutions available for the other two classes. In some cases, Class II applications have resorted to using expensive Class III solutions (e.g., JFK airport parking payment lanes offer an option for using E-Z Pass). This chapter presents a first vehicle to infrastructure (V2I) pairing system targeting Class I and Class II applications by achieving design goals of low-cost and high-accuracy. Vehicles that are not paired are to be processed via manual intervention, incidences of which must also be kept to a minimum.

Low cost and limited instrumentation of infrastructure are the desired criteria for Class I and Class II applications. The existing solutions for Class III applications, such as E-Z Pass, Fastrack, and I-PASS, are not readily usable by the other two classes of applications due to the following limitations. (i) Tag identity database access: For performing an electronic
transaction or authenticating by reading a tag’s identity, the system needs access to a database holding the association information with user’s identity and banking information. In addition, there may be multiple such databases because there are a variety of available toll payment tags [38, 41, 43]. (ii) Hardware requirement on user end: The vehicle needs to have a device or sticker placed near the windshield or dashboard. Such placements are prone to mounting errors [39] and the involvement of an additional device at the user end limits its flexibility, because deployment is a custom effort and upgrading the hardware is cumbersome. (iii) Limited accuracy: Due to the transmission range of the tags, in scenarios with narrow lanes, the signal can be picked up by multiple tollbooths leading to inaccurate charges and unhappy customers [40]. Additionally, the use of such tags for general purpose applications can raise privacy concerns [52].

Although knowledge of location obtained from the GPS on our smartphone can be used to address the challenges, its accuracy ranges from a few meters to tens of meters [110]. It may perform poorly near large buildings and concrete structures. Thus, it is not well suited for our needs. Optical Character Recognition (OCR)-based number plate systems can be used to detect and identify a particular vehicle. But such a technique requires a dedicated infrared (IR)-capable expensive camera aiming for a number plate. Additionally, a number plate can be occluded by other vehicles in dense class-I and class-II applications.

The necessity of additional hardware can be addressed by developing the smartness as part of a smartphone-based application. But the challenge in performing interactions using a longer-range WiFi (or similar) technology is the accurate identification of the specific device to pair with, from a large number of in-range devices. In particular, financial transactions are location-aimed in order to charge the vehicle in a particular lane and position for the provided services. An up-to-date map of all the vehicles can be used to solve the
problem. However, the accuracy necessary calls for techniques that require major hardware upgrades in both the access points (APs) and the smartphones, making it difficult to deploy in practice [95, 104, 143]. In this chapter we exploit a distinct property of Class I and Class II applications: slow and time-varying speed of the vehicles. We refer to the recent time series of velocities of a vehicle as its motion profile or motion signature. Our solution uses self-generated natural signatures (specifically, motion signatures) reported by the target object matched with the same signature detected by simple instrumentation of the environment (a video camera and/or an inexpensive sensor array), layered on commodity, general-purpose communication, and sensing technology (smartphone or other similar device with low-cost inertial sensors) to identify a specific vehicle at a given location (e.g., vehicle A is in lane-4 and next to gate). Our system is comprised of three components: (i) a smartphone connected to the vehicle system using a Bluetooth or an OBD-II link or 802.11p link so that it can access the motion profile of the vehicle; (ii) a camera, which might be already deployed for security purposes; (and/or) (iii) a sensor array deployed for pairing with the vehicles in the lane.

The advantages of our system are many. Unlike range-based pairing technologies such as Near Field Communication (NFC), our system can use any long-range radio-based communication technologies. Soft-Swipe needs infrastructure areas to be instrumented with commodity products and vehicles equipped with smartphones. Therefore, the overall cost of deployment is much lower. Finally, because the device in the vehicle (smartphone) can be programmed, we have the ability to personalize the interactions, such as by allowing the driver to provide additional input, providing status updates to the driver, and so on, as well as to instantly deploy the application and updates.
Soft-Swipe makes the following contributions to the field: (a) presents automatic calibration techniques for infrastructure sensors such as camera and sensor array exploiting V2I links; (b) presents sensor fusion techniques by studying individual sensor characteristics; (c) presents a configurable matching system with precision touching 100% for reliable financial transactions; and (d) shows results from extensive evaluation using real-world experiments. Note that our system primarily targets the first two class of applications and can work using a camera alone, or commodity depth sensors alone.

### 6.1 System Overview

This section first presents the overview of the Soft-Swipe system, then the challenges in enabling V2I pairing based on position are described.

Soft-Swipe enables position-based V2I pairing of vehicles entering into a multi-lane service station. This is performed by matching motion signatures generated from two types of sources. First, Soft-Swipe needs a signature from the vehicle being serviced and tagged with the vehicle’s identity. This signature is received by the infrastructure using V2I links from a device such as a smartphone. The smartphone can fetch the motion profile from the vehicle system by using OBD-II port or by wireless links. Next, Soft-Swipe needs signatures for the same vehicle generated by external, *location-aimed* devices, that is, devices that are targeted at the locus of interaction, such as a video camera whose field of view covers the multi-lane service station. Note that these signatures are not tagged with the vehicle’s identity because the external devices only know that there is a vehicle in their field of view, but do not know which vehicle it is. Finally, note that multiple sensors may be used to provide complementary or additive information. For external *location-aimed* sensing, cameras, ultra-sonic range sensors, or passive IR sensors [49] may be used. In
addition, LIDAR, RADAR, and microwave technologies that do motion estimation by measuring Doppler shifts can be used as well. Finally, electromagnetic sensing devices such as inductive coils [98] may be used to detect the presence of metallic bodies, and potentially their velocity.

Figure 6.1 depicts the architecture of Soft-Swipe where the internal signature is generated by a service device in the vehicle. The external, location-aimed signatures are sensed from two sources (sensing): (a) a video camera aimed at the service lane and (b) an array of depth sensors above the service lane and parallel to it. Soft-Swipe uses the two types of signatures in two important ways. First, during system initialization, the motion signatures received from V2I links are used to calibrate the external sensing components. This allows these devices to properly convert the phenomena they detect (such as a series of images, or the distance between where the sensor array is mounted and a planar surface of the automobile) into motion signatures. When the system is in operation, the generated signatures from vision and sensors are combined adaptively (collaboration) for obtaining an accurate motion signature. The accurate motion signature thus obtained is sent to a centralized server-side
signature-matching module. Here, the external motion signatures are matched to the internal motion signature \((\text{matching})\) that contains the identity of the object. When proper matching occurs, Soft-Swipe can identify the moving object in the sensing field of view, and by definition in the systems proximal locus of interaction.

### 6.2 Soft-Swipe Components

This section presents details of the \(\text{sensing, collaboration, and matching}\) algorithms introduced in the previous section. As we mentioned earlier, our implementation uses two types of uncalibrated, commodity sensors to generate the external \(\text{location-aimed}\) motion signatures, namely, a camera or an array of ultrasonic range sensors, and a built-in device (smartphone) to generate the \(\text{object-tagged}\) internal motion signature.

#### 6.2.1 Automatic Calibration of Camera Exploring V2I links

This section presents a technique for automatic camera calibration exploiting the V2I links. Because a simple camera does not measure the depth of objects in its field of view, the camera needs to be calibrated in order to convert from the rate at which objects move in the camera plane (which is referred as \(\text{optical-flow} \ [46, 74]\)), and measured in pixels per second, to the actual velocity of the object being observed. Camera calibration involves finding its height and orientation with respect to the ground plane. Finding exact orientation and height of the infrastructure cameras is a tedious process. Therefore, prior works have resorted to \(\text{automatic camera calibration}\) techniques which are mainly studied in context of traffic camera installations. Most of these techniques are vanishing point-based approaches where they assume lane markers \([69, 175]\), vehicular size \([68]\), road lines \([152]\), straight line motion of vehicles \([75]\) to calibrate a camera. In contrast, Soft-Swipe explores V2I communication capability of vehicles to calibrate the infrastructure cameras.
Soft-Swipe exploits the *repetitive nature* of vehicles traversing through a station to extract precise scaling values to convert the optical-flow to vehicle’s velocity. Our work is similar to the vehicular speed estimation techniques exploiting lane markers [171], where the movement of vehicle across the known length (length of a marker) is used to derive the speed of the vehicle. Instead of assuming known geometries or markings on the ground, Soft-Swipe exploits V2I communication to know how much a vehicle has moved during initial calibrations. Essentially, given two pixel locations $L_1$ and $L_2$ along the trajectory of the vehicle, Soft-Swipe obtains distance between these pixel positions on the ground plane from the velocity of the vehicles obtained from V2I link. Similarly, for every pixel, Soft-Swipe identifies a scaling value for converting optical flow values to the actual speed on the ground. Vehicles can be detected and tracked across frames using haar based vehicles detectors [148], deep neural network (DNN) [133] approaches. In our implementation, we have created an *angular filter* exploring V2I communication to mitigate the effects of vehicle detection errors. This angular filter learns the direction of vehicular motion in the camera frame during calibration runs. During the system operation, Soft-Swipe projects the optical flow vectors along this direction. This helps in cases where the vehicles are not detected by the vehicular detectors. The auto-calibration then works as follows:

- After the camera is placed, a single test run is made by the vehicle.
- Soft-Swipe collects the pixel-based location-directed, external motion signature from the camera.
- Soft-Swipe collects the object-tagged motion signature from the device inside the car from V2I link.
• By comparing the two motion signatures, Soft-Swipe calibrates the camera by building a mapping function that translates from pixels/second to meters/second across the path of the moving object. Essentially, the mapping function is a location-dependent scaling multiplier that converts from optical speed to actual speed. Note this scaling values is dependent on the pixel position, therefore, a table of scaling values is created.

• Soft-Swipe studies the directions of the vehicular motion in the camera frame and creates angular filters for these directions.

The above technique is used to obtain scaling values and vehicular movement directions. Using these scaling values, for every vehicle entering into the station, Soft-Swipe converts optical-flow to fine-granular motion profile. In our implementation, we have used a camera with 28 frames per second (fps) which gives a velocity at a granularity of 1/28th of a second. For improving granularity of the motion profile cameras with high fps can be employed.

6.2.2 Sensor-Fence: Fine-Grained Motion Profiling with Array of Range Sensors

This section presents fine-granularity motion profile estimation using an array of commodity depth sensors. The array is hung from the ceiling and is parallel to the ground as shown in Figure 6.2 and each lane is equipped with one such sensor array that covers the entire vehicular service station. Inexpensive ultrasonic range sensors [44] that are typically used as robot-eyes [100] are used in our sensor array. Prior work has mainly used ultrasonic sensing and communication to decipher location information [73,117,130,151,186] and sensing shape of the objects [51,124]. In contrast, Soft-Swipe explores coarse ultrasonic-based shape sensing to first detect a vehicle’s existence and then smartly track it for estimating motion profile.
Prior work [185] exploited the roadside sensor deployments to track the vehicle’s time of travel between sensors for speed estimation. Let’s refer to this approach as *trigger speed*. As a vehicle enters the lane it triggers each sensor $i$ at a unique time $t_i$. The average speed between when a vehicle is detected by sensor $i$ and the next sensor $i+1$ is given by $rac{D}{t_{i+1}-t_i}$, where $D$ is the distance between consecutive sensors ($K$ in numbered). This method generates $K-1$ speed estimates.

By using a different approach, the same sensor array can be used to compute a finer granularity motion profile. Essentially, Soft-Swipe uses closely placed robot-eyes (ultrasonic sensors) to precisely estimate the shape of the vehicle at a given time, and tracks this shape with time across a chain of sensors. As a result, the shape of the vehicles (car, truck etc.) is a by-product that can be used by different toll applications. To begin with, shape estimation is performed by modeling a vehicle’s body as a set of planes \{P_1, P_2, P_3, ..., P_n\} with a corresponding set of slopes \{m_1, m_2, m_3, ..., m_n\}. Let us assume that consecutive sensors numbered $i$ and $i+1$ are pointing to the same plane $P_j$ and the vertically traveling signals from these sensors meet the plane at points A and B, respectively, as shown in Figure 6.3. The depths observed by these sensors are $h_i$ and $h_{i+1}$, respectively. Then the slope of plane $P_j$ is estimated as $m_j = \left(\frac{h_{i+1}-h_i}{D}\right)$. In time $\Delta t$, the vehicle moves ahead by $V\Delta t$, and the height reduces by $V\Delta tm_j$. Therefore, the speed of the vehicle at current instance can be estimated by observing the rate of change of depth and the above-computed slope measurement. Let us refer to this approach as *sensor-fence* because it uses the sensors as a fence to determine the speed of the vehicle. Because the sampling rate of these sensors is quite high (20 samples/sec), we can obtain a much finer-grained motion profile of the vehicle. For example, for a vehicle moving at 10 mph, with 20 sensors placed at a separation
of 2 feet, we can obtain more than 1,000 samples in contrast to 20 samples obtained using the trigger-speed approach.

Figure 6.2: Sensor fence design. It provides: 1) Highly accurate shape and speed estimation of vehicles; and, 2) Distinguishes very close-by vehicles.

\[
\theta = \tan^{-1}\left(\frac{h_{i+1} - h_i}{D}\right)
\]

a) Slope Measurement  

b) Speed Measurement

Figure 6.3: Speed calibration from sensors: a) Two points A,B on the vehicle close to each-other can be used to measure the slope of the plane. b) Speed of vehicle is measured using this slope and rate of change of depth observed by sensors.

To deploy a real system based on the above concept the following practical aspects need to be considered. (i) Measurement across different planes: If the points A and B are on different planes, we cannot use the above technique. For two points on the same plane, their
rate of change of depth must be the same, i.e., \( \frac{\Delta h_i}{\Delta t} = \frac{\Delta h_{i+1}}{\Delta t} \). If these rates are not the same, then the sensor reading pair must be discarded. (ii) **Number of sensors**: A larger number of sensors is needed to handle a wide range of speeds. (iii) **Sensor density**: As the sensor density increases, the inter-sensor distance decreases. If the sensors are very close, they will see similar heights, leading to noisy estimates of the speed. (iv) **Sampling time**: If the sampling rate is very high, the depth difference observed within a sample time will be small and affected by the noise floor. (v) **Noisy samples**: Some of the velocity samples estimated are prone to noise due to the flat shape of a plane on the vehicle. Only if the depth difference \( h_{i+1} - h_i \gg 2\sigma \), then the measurement must be used to estimate the speed. The sensor array-based system is inexpensive and can work even in dense vehicular environments with a wide range of speeds.

### 6.2.3 Collaboration: Adaptive Weighted Vision & Sensing

This section attempts to **combine the motion profiles obtained from a camera and sensor-array for obtaining a more accurate motion profile**. First, the properties of speed estimation using the sensor array and vision systems at a given time are studied, and then, an adaptive weighted scheme for an accurate motion profile is designed. In addition, this section automates the calibration and modeling of sensors required for adaptive fusing of motion profiles.

**Parameters impacting vision and sensing systems**: The experimental data depicts that the vision system performance varies with **Distance from camera**. As the distance between the vehicle and the camera increases, its observability in the frame decreases and eventually devolves into ambient noise beyond some point. Hence, the speed measurement accuracy decreases with increase in measurement distance. The sensor array motion profiling
performance depends on the *Angle of measurement* ($\theta$). Soft-Swipe estimates the velocity by measuring the slope of a plane (say, $\theta$). Figure 6.4 presents the velocity estimation accuracy for planes observed from a vehicle. The slope of these planes are measured by observing depth difference between consecutive sensors, which will be affected by the noise floor. Therefore, the slope measurement is not accurate for smaller angles. Notably, accuracy increases with the angle, but the chance of having higher-angle planes on vehicles with a horizontal spread of inter-sensor distance is low. The best angular plane observed by the sensor array is the windshield.

Figure 6.4: *Simulating sensor array with different angles. The higher the angle the better the accuracy of slope estimation.*

**Combining vision and sensor data:** Two major conclusions can be obtained from the previous discussion. First, the accuracies of the sensor array and the vision system depend on parameters independent of the other system, which changes with time. Second, these parameters need to be calibrated and studied for accuracy of measurement before using the system.
Prior approaches in sensor fusion fall into two categories: (1) *Dependent sensory measurements*, where multiple sensor measurements are dependent on each other. One example is widely used techniques for fusing data from inertial sensors such as Kalman filter [96], where different observations (such as accelerometer, GPS) are fused by exploring the relationship between these measurements; and (2) *Independent sensory measurements*, where different sensors sense for the same quantity using independent techniques. One example is EV-Loc [180], where location observations from two sensors (camera and Wi-Fi RSSI) are fused in an adaptive fashion. Similarly, Foresight [111] combines observations from different domains based on distinguishability (or reliability) in each domain. Soft-Swipe belongs to the second category, where independent measurements from the sensor array and camera are fused. However, in contrast to the above schemes, fusing the motion profiles in the context of Soft-Swipe has additional difficulties due to (1) *dependency on observable parameters*, in which errors are dependent on observable parameters such as distance from the camera and slope of the plane; and (2) *time variant errors*: in which the measurement errors depend on abovementioned parameters that change with time. Considering these observations, Soft-Swipe first creates an association table of observed parameters and error variance during the training phase. Using this association table, Soft-Swipe combines the vision and sensor motion profiles by *computing the weights for each sample* for accurate fine granular motion profile.

The collaboration between the camera and sensor array deployed in each lane is enabled by fusing their independent velocity measurements adaptively. Let the velocity measured by camera and sensor arrays be $\hat{v}_c[t]$ and $\hat{v}_s[t]$ respectively, at time $t$ in a given lane, then the velocity estimated by combining, $\hat{v}[t]$ will be

$$\hat{v}[t] = w_c[t]\hat{v}_c[t] + w_s[t]\hat{v}_s[t],$$

(6.1)
where $w_c[t]$ and $w_s[t]$ are the weights of camera and sensor array measurements, respectively. These parameters quantify the confidence or accuracy of individual measurements. The camera and sensor measurements can be modeled as $\hat{v}_c[t] = v_r[t] + e_c[t]$ and $\hat{v}_s[t] = v_r[t] + e_s[t]$, where $v_r[t]$ is the real velocity of the vehicle and $e_c[t], e_s[t]$ are measurement errors of the camera and the sensors, respectively. Therefore, $E(e_c[t]) = E(e_s[t]) = 0$. Let the variance of $e_c[t]$ and $e_s[t]$ be shown as $\sigma_c^2[t]$ and $\sigma_s^2[t]$, respectively. Also the weights must be normalized, therefore $w_s[t] = 1 - w_c[t]$. The error in combining is $e[t] = w_c[t]e_c[t] + w_s[t]e_s[t]$. Minimum mean square error (MMSE) estimation of velocity reduces to minimizing error variance $\sigma_e^2$ as shown below:

$$E(e^2[t]) = \sigma_e^2[t] = w_c[t]^2\sigma_c^2[t] + (1 - w_c[t])^2\sigma_s^2[t].$$  \hspace{1cm} (6.2)

This mean square error is minimized for

$$w_c[t] = \frac{\sigma_c^2[t]}{\sigma_c^2[t] + \sigma_s^2[t]}.$$  \hspace{1cm} (6.3)

Note the error variances of camera observation $\sigma_c^2[t]$ and sensor observation $\sigma_s^2[t]$ are functions of observable parameters such as angle of plane $\theta$ and pixel position $[x, y]$ which are function of time $t$. In order to estimate $w_c[t]$, the abovementioned error variances must be associated with parameters such as slope of plane etc. This involves modeling the sensor array and vision systems and manual calibration for system parameters such as height of camera placement, angle of camera tilt etc. Large sample sets are needed to estimate them accurately. Because modeling the system and observing large sample sets require considerable effort and manual intervention, we instead automate the system using a simple yet intelligent learning and estimation technique as described below.

**Learning Phase:** The training set is created and updated in two phases. First, during the training phase, for each lane, the user performs trial runs to create different possible ($[x, y], \theta$)
pairs and measures $\hat{v}_c[t]$ and $\hat{v}_s[t]$. Along with the estimated velocities, the training set contains associated real velocity $v_r$, which is obtained from the vehicle’s electronic messages. Second, during the test phase, if there is only one vehicle in the vehicle station, then the electronic transmissions of corresponding vehicle is used to train the system deployed in its lane. During this test phase, both vehicle transmissions and sensor observations are added to this set, providing a large training set whose size increases with time. Figure 6.5 presents these two phases and the table construction. With this continuous training set, the sample variances $\sigma^2_c[t], \sigma^2_s[t]$ are incrementally estimated and an association table is created for parameters $([x, y], \sigma^2_c[t]), (\theta, \sigma^2_s[t])$. Also, a smoothing function is applied on this table to average close observations, creating a continuous trend of variance. Figure 6.6 presents $\sigma^2_c[t]$ plotted as a function of distance from camera using the history table for 25 experiments. This distance from camera is mapped to pixel position using a fixed transformation function obtained during training.

*Estimating the velocity:* Often vehicles traveling in the same lane with similar build (e.g., car, truck etc.) have repetitive $(x, y, \theta)$ values. As a result of this, for repeating $(x, y, \theta)$, the variances can be obtained from the table. From the variance, the weight $\hat{w}_c[t]$ is estimated.

---

**Figure 6.5:** *Figure representing data-flow while estimating weights for MMSE estimation from history table.*

![Diagram](image)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\sigma^2_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\sigma^2_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 6.6: Camera speed estimation error variance is plotted with vehicle-position from camera frame over 25 experiments.

using Equation 6.3, which gives the velocity as \( \hat{v}[t] = \hat{w}_c[t] \hat{v}_c[t] + (1 - \hat{w}_c[t]) \hat{v}_s[t] \). The estimated velocity \( \hat{v} \) at each time \( t \) has different measurement errors that must be considered when computing the motion profile of a vehicle over a time-interval. This measurement error is quantified by the variance of measurement \( \hat{\sigma}^2[t] = \frac{\hat{\sigma}^2_c[t] \hat{\sigma}^2_s[t]}{\hat{\sigma}^2_c[t] + \hat{\sigma}^2_s[t]} \) which is derived using camera measurement error variance \( \hat{\sigma}^2_c[t] \) and sensor measurement error variance \( \hat{\sigma}^2_s[t] \) obtained from table look-up using Equation 6.3.

6.2.4 Weighted Matching of cross domain motion signatures

This section describes the matching component of the system that matches observations obtained from two domains, sensors in each lane and the motion profile from vehicles. In particular, lane observations are matched with electronic messages from vehicles. The accurate motion profile for each lane is obtained using the technique described in the previous section. Similarly, vehicles transmit their motion profile using electronic messages. Essentially if the observations get matched to a vehicle, then it is allowed gate access. If not, DashCalib calls for manual transaction. This section first describes the challenges
in cross domain matching and then presents a novel metric *Weighted Euclidean Distance* quantifying the closeness between cross domain motion profiles. Using this metric, the rest of the section presents matching and various decisions derived from it.

**Challenges in cross-domain matching:** As described above, the matching is performed between two domains (sets of data). First, the electronic identities (e.g., IP-addresses or MAC-addresses of smart-phones) is communicated to Soft-Swipe’s central server over the wireless medium. These electronic identities \( e_i \) are associated with their motion profile, which is received as a packet stream holding velocity and time. Also these electronic motion profiles are assumed to be highly accurate and sampled at a high sampling rate. Second, the observations \( o_j \) from the sensors in each lane are communicated over the wired infrastructure. These observations will be holding the lane identity and position (position in a lane), current time \( t \), observed velocity of the vehicle \( v_o^j[t] \), and the accuracy of observation \( \sigma_j^2[t] \). Note that the observed velocity is the adaptive weighted version of vision and sensor array and is the output of the algorithm described in previous section.

There are two critical challenges in matching electronic messages with observations. **Different accuracies of measurements:** The speed estimation accuracy obtained from an observation changes with time depending on different parameters described in \$6.2.3\$. If this effect is not considered, then noisy observations at one instant in time can render the accurate observations useless at other times. **Defective (or) tampered equipment:** There is no guarantee that the vehicles are transmitting their motion profiles. Lack of electronic messages from a vehicle can cause errors in matching.

These two challenges make the problem of matching motion profile distinct from the problems explored in the literature. Traditionally, Euclidean distance [132] and dynamic
time warping [179] are methods employed for finding the distance between two time series. But these methods cannot handle the noise or non-uniformity in the measurement errors. Longest common subsequence is proposed to handle possible noise that may appear in the data; however, it ignores the various time-gaps between similar subsequences, which leads to inaccuracies. Considering this, Soft-Swipe first defines a weighted version of Euclidean distance referred to as *Weighted Euclidean Distance* to compute the similarity between two time series that can handle noise. Then, Soft-Swipe uses the above metric to match vehicles with respective observations.

**Weighted Euclidean Distance:** Non-uniformity in measurement accuracies is addressed by giving weights to the observations based on accuracy. To derive weights based on accuracy (variance of observation), consider an observation \( o_j \) with motion profile spanning in a time window \([T_{oj}^o, T]\) containing \( M_j \) samples. This motion profile represents a point in \( M_j \) dimensional space. Let us define *Weighted Euclidean Distance* \( D = \sum_{t=T_{oj}^o}^{T} w_j[t]^2 (\hat{v}_j[t] - v_j[t])^2 \) between two motion profiles as the square of distance between two points in the multi-dimensional space, where each dimension is scaled by a weight. These weights \( (w_j[t]) \) are chosen such that the distance between motion profile of \( o_j \) and its accurate measurement (\( \hat{v}_j[t] \) obtained by electronic messages) must be minimum. In such a case, the distance \( D \) is same as mean square error due to measurement noise (discussed in §6.2.3) and can be formulated as given below

\[
E(\sum_{t=T_{oj}^o}^{T} w_j[t]^2 (\hat{v}_j[t] - v_j[t])^2) = \sum_{t=T_{oj}^o}^{T} w_j[t]^2 \sigma_j^2[t].
\] (6.4)

Also the weights must be normalized over time. Therefore the objective function \( D \), which can be formulated as:

\[
\text{minimize} \quad \sum_{t=T_{oj}^o}^{T} w_j[t]^2 \sigma_j^2[t] \\
\text{subject to} \quad \sum_{t=T_{oj}^o}^{T} w_j[t] = 1.
\]
From Cauchy-Schwarz Inequality,

\[
\sum_{t=T_j}^{T} w_j^2[t] \sigma_j^2[t] \geq \left( \sum_{t=T_j}^{T} w_j[t] \right)^2 = 1. \tag{6.5}
\]

Therefore,

\[
\sum_{t=T_j}^{T} w_j^2[t] \sigma_j^2[t] \geq \frac{1}{\sum_{t=T_j}^{T} \frac{1}{\sigma_j^2[t]}}. \tag{6.6}
\]

The above minimization function is optimized for \( w_j[t] \sigma_j^2[t] = K \forall t \in [0, T] \) where \( K \) is constant.

The optimal weights can be estimated from the variances of each observation as \( w_j[t] = \frac{1}{\sum_{t=T_j}^{T} \frac{1}{\sigma_j^2[t]}} \). The computed weights are based on accuracy of measurement as the weight is inversely related to the variance of the observation. Further, for a significantly large number of samples, the distribution of \( D \) can be approximated as a normal-distribution with mean of \( \mu_{Dj} = \frac{1}{\sum_{t=0}^{T} \frac{1}{\sigma_j^2[t]}} \), with variance of \( \sigma_{Dj}^2 = \frac{\sum_{t=T_j}^{T} \sigma_j^2[t]}{\sum_{t=T_j}^{T} \sigma_j^2[t]} \). This distribution of \( D \) for observation \( o_j \) is used to detect corresponding electronic match.

**Matching and Fault Detection:** Soft-Swipe considers observations that have crossed a threshold length for matching (15 to 20 seconds is found to be optimal in our experiments). With these data, matching happens in a time slotted fashion, and all the observations crossing this threshold in the current time slot are matched in the next time slot. Also time-slot length is chosen to be much larger than threshold length.

In order to perform matching, the user defines a parameter \( c \) (Match Confidence) lying between 0 and 1. Matching for an observation \( o_j \), is performed using the abovementioned weights and \( c \). Then Soft-Swipe computes Weighted Euclidean Distance \( D[i, j] \) for every observation \( o_j \) and electronic identity \( e_i \) to determine the following:

- If \( e_i \) is a correct match for \( o_j \), then the distance \( D[i, j] \) is the smallest \( \forall i \) and \( D[i, j] \) is in high confidence region of normal distribution. (Match)
• If $o_j$ has no correct match, then the distances $D[i,j] \forall i$ are not in high confidence region of normal distribution. (*Fault, blocked for manual processing.*)

• If $e_i$ has not matched with any $o_j \forall j$, then $e_i$ is carried over to the next time slot. (*Vehicles yet to enter the station.*)

### 6.3 Implementation

In this section we outline our system implemented in the vehicular manufacturing and testing station.

**Vision system:** Our vision system is implemented in C++ using OpenCV, which captures real-time video feed and finds good features in the frame that can be used to track a vehicle (described by Shi et al. [147]). These features typically include corners, boundaries of a vehicle etc. Once these features are extracted, the vision system checks how these features have moved across consecutive frames in order to measure their shift. These shifts are observed in terms of pixels per unit time and referred to as *optical flow vectors* in computer vision literature [46, 74]. The optical flow vectors from different feature points on the vehicle are aggregated to obtain the vehicle’s velocity in the camera plane.

Next, a noise filter is created to filter out the optical flow vectors that are below a threshold and not in the directions of vehicular movements. This threshold is determined during the initial calibration runs. Also, the pixels that do not correspond to any lane can be removed by using image segmentation (segmenting the image corresponding to the lane). Small changes in light-conditions, reflections from moving object on the ground and background human movements create optical flow vectors with much smaller magnitudes and in different directions compared to optical flow vectors of a moving vehicle and are filtered out.
The vision system was implemented using a commodity Logitech Quick-cam pro camera and was mounted 2 meters above ground level. Additionally, we have experimented with Belkin NetCam HD+ and other off-the-shelf digital cameras. The camera must be mounted at a significant height in order to ensure coverage and to approximate a vehicle’s motion to a straight line in the camera plane.

Figure 6.7: Optical-flow vectors of a moving vehicle created by observing the motion vectors of selected feature points.

The vision system assumes that a vehicle is a solid object and therefore, the system is not trained to look for specific visual features (such as shape of the car, car logo etc.). Feature-based vehicle detection and tracking mechanisms (where the vehicle can be classified as car, truck etc.) can certainly be layered on Soft-Swipe. Also, the visual features (described by Li et al. [111]) could be used for matching. However, these visual features cannot distinguish identical vehicles. Soft-Swipe, on the other hand, gives accurate matching without depending on vehicle-specific properties.
Sensor array: The sensor array is deployed using four ultrasonic sensors [44], which are controlled by an Arduino Yun [48] controller. The inter-sensor distance is 30 cm and covers only 90 cm of the vehicle service station. The sensor array measures the depth at a constant rate of 20 per second and these measurements are processed by Arduino to obtain parameters such as slope or velocity of a vehicle etc. First, the presence of a vehicle is detected by recording the number of sensors triggered at a given time instance. Other motions (such as caused by a walking person) will usually trigger a small set of sensors and can be ignored. Then the measured velocities along with the parameters are sent to the central server (implemented in a Laptop) using serial communication.

Motion profiles from vehicles are collected by connecting a smart device with OBD-II system. Adaptive weight and matching components are implemented in Matlab R2015a, where the data from the vision system, serial port communication (Arduino), and vehicle smart device are fetched and processed. The above implementation uses commodity sensors...
with an average cost of 250 USD per lane. Large-scale production of the system might cost much lower than presented costs.

6.4 Evaluation

This section evaluates the motion profile accuracy of the vision system, sensor array, and adaptive weight algorithm. Then, different metrics for evaluating Soft-Swipe are presented and evaluated with extensive real-world experiments.

Vision system performance: Our vision system is robust to background noise and estimated speed with an overall standard deviation of 2 kmph and less than 0.5 kmph with a large training set as shown in Figure 6.9 (a). In evaluating the vision-system we observed a variable accuracy achieved in speed sensing. This can be explained as follows, Soft-Swipe calibrates the pixel speed from raw frames and converts this pixel speed to true speed by multiplying with a scaling value. This scaling value is derived for each pixel position during

Figure 6.9: Speed estimation variance-plots of vision system (experiments) with average standard deviation 1.6 kmph, sensor system (simulation and experiments) with average variance of 2 kmph, and adaptive algorithm with average variance of 1 kmph from Indoor low speed experiments. The adaptive weight algorithm combines sensor-simulated results and vision experimental results for estimating the motion profile and reduces the error by more than 50%.
initial training runs. Each training run gives scaling values for a few pixels in the frame. However, during system usage, the closest pixel position with a known scaling value is used in that case.

Figure 6.10: Motion profile from vehicular Electronic messages, sensor-system, vision-system, and Adaptive-weight algorithm.

Sensor-fence performance: We have evaluated the 4-sensor array described in §6.3 by examining speed measurement accuracy. Figure 6.9(b) (blue bars) plots the speed measurement accuracy. We observe that the measurement error increases with the speed of measurement. To analyze the trend, we have simulated the sensor system by feeding traces containing dimensions of different vehicles and vehicle mobility traces. Figure 6.9(b) (red bars) plots the accuracy obtained from simulation. Simulation results showed significant performance for higher velocities. This is due to the higher number of sensors needed for capturing higher velocities. The sensor-fence performance depends mainly on the angle of plane as described in sensor fence section. With the limited number of sensors (4 were
used), the chance of capturing higher-slope planes is less as compared to a long chain of sensors (in simulations).

**Adaptive weight algorithm performance:** We evaluate the benefits of combining the motion profiles obtained from the vision and sensor systems by using the adaptive weight algorithm. Figure 6.10 plots the motion profile using vision, sensor array, and adaptive weight algorithm. The adaptive weight algorithm produces a less noisy and more accurate motion profile by combining information from both the vision and the sensor array components. We have also experimented with several naive smoothing algorithms to reduce noise in the process of combining information. But these algorithms miss the sharp peaks in the motion profile (sudden stops, acceleration, etc.) and therefore are not suitable for dynamic vehicular speeds. For a set of 30 experiments, the adaptive weight algorithm reduced error by 50% (i.e., nearly 1kmph) compared to vision system and 55% (i.e., nearly 1.2kmph), as shown in the Figure 6.9(c).

This section first presents the metrics involved in evaluating Soft-Swipe system. The experimental setup for evaluation is then presented, followed by a discussion on results and observations.

**Evaluation metrics:** To examine the benefits of the matching algorithm, we evaluated the system for following metrics. 1.) Precision \( p \), Recall \( r \) and F-Score \( f \): Precision gives the ratio of the number of correct matches to the total number of matches produced by the algorithm. Recall gives the ratio of the number of correct matches produced by the algorithm to the total number of correct matches (ground truth). F-score \( (F_1\text{-score}) \) is the commonly used statistical metric quantifying accuracy of matching, considering both \( p \) and \( r \). Precision, recall, and F-score are standard metrics defined for matching [126]. In addition, we define the following metrics from the user’s point of view, which are important
for different toll based applications. 2.) Identity-Swap: This is the probability of swapping identity between vehicles. It is the ratio of false positives to the total number of times an observation (user) participates in the matching. Note this is always less than $1 - p$, because $1 - p$ is the ratio of false-positives to total number of times an observation is matched. This metric quantifies the probability that a user pays someone else’s toll and still got the gate access. This metric is essential for drive-thru and other service based transactions as this metric quantifies the incidence of swapped transactions. 3.) False-stop: This is the probability of having a wrong match or no match for a given observation. This includes observations that are considered to have a wrong match (false negatives) as well as no matches and is therefore always greater than $1 - r$. 4.) Miss-Rate: This is the probability of detecting an observation without electronic transmissions (rogue vehicle). This metric quantifies the probability of having gate access without performing electronic pairing and therefore, is essential for toll-based applications.

**Experimental setup:** First a huge number of single-lane experiments are conducted with controlled variation of traffic pattern, just like typical class I and class II applications. Note these applications can often have multiple lane for reducing wait times. Because building the system for multiple lanes experimental setup is cumbersome, we have designed an emulator that simply replays different or the same experiments across different emulated lanes. Therefore, vehicles across different lanes can have the same motion profiles. Then multi-lane experiments are created with varying lane-count ranging from 1 to 5. Additionally, the system receives motion profiles from seven exterior electronic transmissions (vehicles yet to enter the station but transmitting the motion-profile). For all experiments, the user-defined parameter $c$ is set to 0.99. For evaluating the miss rate, out of the vehicles in the station,
one vehicle is assumed rogue, and does not transmit the motion profile. Then the system is evaluated for detecting this rogue vehicle.

**Results and observations:** Figure 6.11 depicts the results observed from the abovementioned experiments. From these results, we observe the following general trends: *Precision increased with number of lanes.* This trend in precision is mainly attributed to reduction in noise (noise-vehicle transmissions) per lane. Increase in precision rate also results in lower swapping rates. *Recall decreased with number of lanes and false stops increased linearly with number of lanes.* With an increase in the number of lanes, the fraction of noise vehicles (vehicles yet to enter the station) decreases, leading more vehicles to be considered a match. An increase in recall reduces precision. When recall is high, the lower precision will result in some vehicles being stopped for traditional processing (perhaps with manual intervention). We observed that miss rate can be reduced further by increasing the confidence \( c \) defined in §6.2.4, but this will reduce the recall, leading to valid pairs being eliminated as a miss.

Figure 6.11: *Weighted matching algorithm is evaluated for different metrics using vision-only, sensor-fence, and Adaptive weight(AW) algorithm.*
(rogue-vehicle). This implies the lower the miss rate, the higher the chance of valid vehicles being considered as a miss. Also, by reducing \( c \), recall can be increased, but this reduces the precision.

6.5 Related Work

DashCalib enables accurate pairing between a vehicle and the infrastructure by exploiting motion signatures of the vehicle at a particular location. Our work is primarily related to following three lines of research.

(i) **Motion signatures for identification:** Wang et al. [173] exploits visual and discrete motion sequences for identifying the human visually. These position sequences cannot be used to distinguish vehicles because all the vehicles move in the same direction and may have identical visual features. Li et al. [111] uses position and color to identify vehicles and enable unicast. However, GPS position cannot resolve the vehicle to its respective lane. Also, multiple vehicles can have the same color (e.g., very common in automobile manufacturing plants). RoadView [167, 168] uses motion signatures of vehicles observed by a vehicle using its on board sensors, such as camera and RADAR to identify neighboring vehicles. In order to enable pairing between vehicles, distinguishable signatures must be extracted with high accuracy, which cannot be achieved by the works described above.

(ii) **Location signatures:** Location-based signatures are widely explored in the context of NFC, wireless localization, and wireless security. The ambient sensors available on mobile phones, such as audio, light, GPS, Wi-Fi, Bluetooth, and thermal, are used to create location-specific signatures to authenticate [87, 88, 114, 170]. Wang et al. [174] defines motion signatures, which can be captured by inertial sensors on mobile phones to provide indoor localization service. Gao et al. [79] have presented techniques to track the
user exploring the motion signatures. Bao et al. [53] explores Wi-Fi and Bluetooth RSSI signatures to sense the context of a user. However, Wi-Fi-based signatures vary greatly in dynamic environments and are difficult to sense.

(iii) Vehicle speed sensing and Matching: Prior works have explored road-side camera [83,142]. Soft-Swipe uses a novel algorithm for dynamic speed estimation of a vehicle using both vision and depth-sensor array. The speed estimation algorithm from vision proposed in Soft-Swipe is similar to works on speed estimation from road-side cameras [83, 142]. Soft-Swipe first estimates the shape of a moving vehicle using a depth sensor array hung from the ceiling. Then, movement of this object across sensor-array length is used to estimate the vehicle speed. The problem of estimating the shape of a vehicle has similarities to the problem of object construction from 3D points [141], but Soft-Swipe exploits the 2D nature of the speed estimation problem and includes a novel lightweight algorithm for shape and speed estimations.

(iv) Sensor fusion: Prior works [111,180] have explored the weight adaptation algorithms by using variances of observations. However, we showed that these variances do not remain constant in the context of applications based on vehicular speed sensing. Realizing this non-uniformity in the variances, we have proposed a learning-based adaptive weight algorithm to combine motion signatures from multiple modalities by computing weights for each sample.
Chapter 7: Conclusion and Future Work

In this dissertation, I have studied three different calibration techniques intended for dashboard, traffic and infrastructure cameras and their applications. In particular, I have presented keypoint annotation-based calibration in AutoCalib, opportunistic calibration in DashCalib, and communication-based calibration in Soft-Swipe. Additionally, I have implemented vanishing point-based, MonoSLAM-based, and IMU sensor-based approaches for comparison purpose. These calibration services can be employed by different safety applications. The calibrated traffic cameras can ensure always-on speed enforcement services to make roads safer. The calibrated dashboard cameras derive the positions of neighboring vehicles, measure distances on the ground, distances from the curb, etc., enabling a wide range of safety applications. RoadView and RoadMap are collaborative vehicular applications to enhance the sensing range of vehicles. This extended sensing can enable different accident prediction and prevention applications. Soft-Swipe transforms a vehicle to an electronic card for financial transactions happening from the vehicle.

A variety of cameras are being installed on a vehicle’s body for enabling safety applications. Most of the autonomous vehicle (AV) designs employ camera and RADAR-based solutions in place of LIDAR because the former solutions are more cost efficient. The camera and RADAR sensor imagery are fused to derive the positions of neighboring vehicles, which helps an AV to plan its future trajectory. Due to the limited field of view (FOV) of
Figure 7.1: (a) 360 degree view generated by Mercedes AMG S 63 [19]; (b) Camera placements for generating stitched top-view [8].

For measuring the distances on the road and to derive other geometric measurements, the cameras must be calibrated. The 360-degree top view is an application that combines images from different cameras on a vehicular body to generate a top view. Figure 7.1(a) shows an example of a stitched 360-degree view generated by a Mercedes AMG S 63 vehicle. Figure 7.1(b) shows the typical camera placements for generating the stitched top view. In order to stitch different views, the cameras must be calibrated with respect to each other. Sensor fusion applications that employ data from multiple cameras, RADAR, and ultrasonic sensors must be calibrated to bring different sensors’ perceptions to a common frame of reference.
The calibration problems can be categorized as (a) cross camera calibration, which involves estimation of relative pose and translation between the cameras; and (b) cross sensor calibration, which involves estimation of relative pose and translation between the camera and the RADAR, LIDAR, or ultrasonic sensors. Each of the sensors can be calibrated in the common frame of reference, such as the vehicular coordinate system or they can be calibrated with respect to each other. For calibrating such camera installations, a calibration mat is used; it is employed in a calibration station. This calibration mat has known markers and is placed around the vehicle on the road plane.
7.1 **Calibration as a server-client application**

Different calibration modules can be abstracted to form a web service for providing calibration services. Figure 7.2 shows multiple modules for calibration and their dependency on the communication link, quality of the imagery, and the choice of the computation platform. The vanishing point-based approaches can employ lightweight feature-point tracking techniques or line detection techniques and therefore need good light conditions (as we observed during DashCalib evaluation). These techniques are computationally lightweight and can be employed on an edge node. Similarly, IMU, GPS based modules can be employed on the edge node.

Techniques such as keypoint annotation employ deep learning-based solutions, making them an ideal choice to be employed on the server. Also, such techniques require good communications links to upload the respective images. These services can be employed by DashCam-based applications to annotate the images from the scene in order to calibrate the DashCam. They can leverage lightweight techniques presented in DashCalib in the events of limited connectivity. Similarly, the keypoint annotation can be extended to the front view of the vehicles and the objects that appear on the roads such as stop signs and other information markers present on the road. The keypoints of the vehicles (or objects with known geometry) from two different camera views can be employed to derive the relative pose and translation between two camera views.

7.2 **Automatic LIDAR and camera cross sensor calibration**

Different sensor installations on the vehicle must be calibrated w.r.t each other for sensor fusion applications. The camera sensor is widely used for object detection and tracking. Contrary, LIDAR provides 3D point cloud of the objects which helps in motion planning
and collision avoidance applications. Fusing these sensor information helps in annotating the LIDAR point clouds to respective object identities. Cross sensor calibration brings the observations of different vehicles to a common frame of reference. A live and automatic calibration which estimates the relative orientation and translation between sensors on the fly can be designed for sensor fusion applications.

Road side markers such as stop signs can be exploited to derive the relative pose between the camera and LIDAR sensors automatically. LIDAR gives the 3D point cloud of the stop sign and the same sign can be detected by the camera. The peripheral keypoints of the roadsign images can be matched with respective 3D point cloud segmentations from the LIDAR. Using this matching, a perspective-n-point problem can be solved to derive the relative orientation and translation of the camera w.r.t the LIDAR. By observing multiple such road signs, multiple calibration values can be derived. Statistical filters presented in AutoCalib can be exploited to improve the accuracy of the calibration.
Bibliography


