# End-to-end Vehicle Tracking and Counting in Traffic Videos

by

Yanzi Jin
B.A., Dalian University of Technology

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2019

Chicago, Illinois

Defense Committee:
Jakob Eriksson, Chair and Advisor
Xinhua Zhang
Brian Ziebart
Jie Yang, MSCS Department
Ahmet Enis Cetin, ECE Department

Dedicated to my parents and sister, for their unconditional love and support.

## ACKNOWLEDGMENT

I want to thank my advisor Dr. Jakob Eriksson throughout these years for his support and instructions. He is very visionary about this exciting and challenging project. I admire his academic attitude and the effort he spent to learn together with students. I appreciate the time that we went through the code line by line for a hidden bug or hours of discussion of algorithm details. Thanks to his generous freedom and support, I received comprehensive training and became independent and mature in research.

Also, I would like to express appreciation to my committee members, for their advice and inspiration. I took their classes and learned preliminary knowledge of different research areas, contributing to the building blocks of my research project. Specifically, I want to thank Dr. Xinhua Zhang, whose rigorous attitude toward research and passion of teaching greatly inspired me. Also, Dr. Jie Yang's rich experience gave me helpful guidance on multiple technical questions.

In addition, I want to thank my parents and sister. My father teaches me to be strong and tough; my mother supports me for all my big decisions; my sister takes care of my parents when I am far away. Finally, special thanks to Sihong Xie, for his company and encouragement along this tough journey.

YJ

## CONTRIBUTIONS OF AUTHORS

Chapter 2 represents a published manuscript, where I am the first author. My advisor Jakob Eriksson contributed to the writing of it. Other parts of this thesis consist of my own un-published work.

## TABLE OF CONTENTS


| CHAPTER | | | PAGE |
|---|---|---|---|
| **1** | **INTRODUCTION** | | 1 |
| | 1.1 | Camera system for traffic engineering | 1 |
| | 1.2 | Computer vision in transportation | 2 |
| | 1.3 | Practical challenges | 3 |
| **2** | **FULLY AUTOMATIC VEHICLE TRACKER** | | 5 |
| | 2.1 | Introduction | 5 |
| | 2.2 | Preliminary | 6 |
| | 2.2.1 | Background subtraction | 7 |
| | 2.2.2 | Object detection | 8 |
| | 2.2.3 | Optical flow | 10 |
| | 2.3 | Heuristic tracker by Kalman Filter | 12 |
| | 2.3.1 | Model definition | 12 |
| | 2.3.2 | Measurement acquisition | 16 |
| | 2.3.3 | Model update | 17 |
| | 2.4 | Fully automatic initialization and termination | 18 |
| | 2.4.1 | Object entry and exit | 18 |
| | 2.4.2 | Our method | 19 |
| | 2.5 | Evaluation | 21 |
| | 2.5.1 | Tracker configuration | 21 |
| | 2.5.2 | Evaluation metrics | 22 |
| | 2.5.3 | Object-level evaluation | 23 |
| | 2.5.4 | Pixel-level evaluation during entire lifetime | 24 |
| | 2.5.5 | Pixel-level evaluation during tracked period | 27 |
| | 2.5.6 | Throughput | 28 |
| **3** | **SCENE LEARNING** | | 32 |
| | 3.1 | Introduction | 32 |
| | 3.2 | Atomic motion extraction | 33 |
| | 3.2.1 | Non-parametric clustering via Hierarchical Dirichlet Process (HDP) | 34 |
| | 3.3 | Semantic knowledge learning | 38 |
| | 3.3.1 | Robust ridge climbing | 38 |
| | 3.3.2 | Topic extent learning | 42 |
| | 3.3.3 | Entry/exit hotspots extraction | 42 |
| | 3.4 | Semantic knowledge visualization | 44 |
| | 3.5 | Discussion | 47 |

## TABLE OF CONTENTS (Continued)

**CHAPTER** **PAGE**


**4 SEMANTIC TRACKER** . . . . . 51
4.1 Introduction . . . . . 51
4.2 Object motion assignment . . . . . 53
4.3 Tracking score . . . . . 54
4.3.1 Initialization . . . . . 55
4.3.2 Termination . . . . . 58
4.3.3 Tracking quality evaluation . . . . . 59
4.4 Semantic tracker . . . . . 61
4.4.1 Semantic knowledge update . . . . . 61
4.5 Evaluation . . . . . 63
4.5.1 Tracking accuracy . . . . . 63

**5 SCENE-SPECIFIC MOTION MODEL** . . . . . 66
5.1 Introduction . . . . . 66
5.2 Gaussian Process . . . . . 67
5.2.1 Multiple output Gaussian Process . . . . . 68
5.2.2 Online processing for streaming data . . . . . 68
5.3 Unscented Kalman Filter . . . . . 70
5.4 GP-UKF tracker . . . . . 72
5.4.1 Tracking . . . . . 72
5.4.2 Cold start . . . . . 74
5.5 Evaluation . . . . . 74
5.5.1 Comparison with linear Kalman Filter . . . . . 74
5.5.2 Case study . . . . . 76
5.6 Discussion . . . . . 78

**6 VEHICLE COUNTING SYSTEM** . . . . . 79
6.1 Introduction . . . . . 79
6.2 Vehicle counter . . . . . 80
6.2.1 Vehicle counter with human annotation . . . . . 81
6.2.2 Vehicle counter with semantic knowledge . . . . . 84
6.2.3 Web portal . . . . . 85

**7 DATASET** . . . . . 88
7.1 Introduction . . . . . 88
7.2 Dataset . . . . . 89

**8 RELATED WORK** . . . . . 91
8.1 Object tracking . . . . . 91
8.2 Tracking initialization and termination . . . . . 91
8.3 Vision in intelligent traffic system . . . . . 92
8.4 Scene understanding . . . . . 92

## LIST OF TABLES


| TABLE | | PAGE |
|---|---|---|
| I | Dataset overview. The second and the third columns show the resolution and object size range in pixels, followed by number of videos under each group. The rightmost four columns show the number of videos reflecting various challenging aspects (occlusion, shadow, distortion and pedestrian). . . . . | 90 |

## LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 1 | Common background subtraction failure cases. Pixel values change for many reasons other than motion. | 9 |
| 2 | Common problems in optical flow estimation. | 11 |
| 3 | Overview of proposed system. Separate Kalman filter state is initialized, maintained and terminated for each tracked object. The state is updated based on matching input from background subtraction, object detection and optical flow. | 12 |
| 4 | Tracked object count on low resolution, less complex scenes. | 24 |
| 5 | Tracked object count on high resolution, more complex scenes. | 25 |
| 6 | Overall overlap ratio on low resolution, less complex scenes. | 27 |
| 7 | Overall overlap ratio on high resolution, more complex scenes. | 28 |
| 8 | Success plot of low resolution, less complex scenes. | 29 |
| 9 | Success plots of high resolution, more complex scenes. | 30 |
| 10 | Tracker throughput on different resolution videos. | 31 |
| 11 | System overview. The left corresponds to scene learning module: frequent motions are extracted in an unsupervised fashion, then the active regions where most objects move are learned on top of the motion result. Next, the entry/exit hotspot and their direction are extracted, describing how most objects enter and exit the scene. On the right, the semantic knowledge is applied to a tracker. | 34 |
| 12 | Left column illustrates the spatial and temporary quantization of video frames. The right shows a visual word obtained by discretizing the optical flow direction. | 35 |
| 13 | Motions learned by HDP, colors indicate directions as the color wheel shows, the lightness indicate the magnitudes of the maximal probability values.. | 37 |
| 14 | 3D visualization of the distribution of the first topic in Figure 13 and ridge climbing illustration on it: along the dominant topic direction (a) and its perpendicular direction (b). Blue cross indicates the starting point, red and green point indicates the end point. | 39 |
| 15 | Ridge climbing illustration on the first topic in Figure 13: (a) is an example of the topic distribution on one grid $\mathtt{i}$, where the radius of each circle sector indicates $\boldsymbol{\phi}_{\mathtt{i}} = [\phi_{\mathtt{i}}^{1}, \phi_{\mathtt{i}}^{2}, \ldots, \phi_{\mathtt{i}}^{\mathtt{D}}]$. (b) shows the move-adjust step of each iteration. (c) is an extracted ridge starting from the highest density grid, where the blue star indicates the highest density grid, green and red line indicates ridges to the start/end point separately. | 39 |

## LIST OF FIGURES (Continued)


| FIGURE | | PAGE |
|---|---|---|
| 16 | Scene learning results on the first topic in Figure 13: (a) shows multiple ridges learned from local maximal grid. (b) is the perpendicular width, where lighter intensity indicates a larger width. (c) shows extracted entry/exit hotspots indicates by the green and red star. And the yellow arrow shows their direction. . . . . | 41 |
| 17 | Scene learning results at an intersection with one-way road. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 45 |
| 18 | Scene learning results at an intersection with one-way road. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 46 |
| 19 | Scene learning results at a multi-way intersection. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 46 |
| 20 | Scene learning results at a crowded multi-way intersection. Entry (green) and exit (red) with direction (yellow arrows). . . . . | 47 |
| 21 | Failure case with simultaneous opposite directions. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 48 |
| 22 | Failure case with simultaneous opposite directions. Entry (green) and exit (red) with direction (yellow arrows) at a crowded intersection. | 49 |
| 23 | Model training on sparse night video about 5 minutes. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 50 |
| 24 | Model training on sparse night video about 30 minutes. Entry (green) and exit (red) locations with direction (yellow arrows). . . . . | 50 |
| 25 | Two scenarios of object enters and exits: (a): objects enter with a tiny size and move out of image boundary; (b): objects enter from the image boundary and exit with a tiny size in within the frame. Green and red star are the obtained entry/exit hotspots; yellow arrows shows their direction. . . . . | 56 |
| 26 | All the solid pink arrows above indicate the moving direction $\boldsymbol{\nu}_{\mathbf{r}}$, the pink dot is the center of the rectangle, blue dots are the intersection points with the rectangle. (a): perpendicular width $W(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$ of the object's bounding box wrt. to direction $\boldsymbol{\nu}_{\mathbf{r}}$, shown as the blue dash line. b: last point $\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$ of the object's bounding box $\mathbf{r}$, where the dash arrow is $\mathsf{R}(\mathbf{r}, -\boldsymbol{\nu}_{\mathbf{r}})$. (c): distance between a hotspot with a rectangle's center point and last point $\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$, shown in pink and blue dash lines. | 57 |
| 27 | The number of true positive and false positve trackers. The black lines mark the ground truth, the striped and black bar are the true positive and false positive, separately. . . . . | 64 |
| 28 | Success rate plot. A larger area under the curve shows the better performance. . . . . | 65 |
| 29 | The training data count grows linearly with the training trajectory count. . . . . | 69 |

## LIST OF FIGURES (Continued)

**FIGURE** **PAGE**


30 The mean IoU of Gaussian Process (GP)s trained on the trajectories from the heuristic and semantic tracker, and the linear Kalman filter. Each topic has two GPs and a linear Kalman filter. . . . . 75
31 Tracking screenshots at frame 854, no trajectory for Gaussian Process. 76
32 Tracking screenshots at frame 914, 1-2 trajectories for Gaussian Process. . . . . 77
33 Tracking screenshots at frame 989, 5-8 trajectories for Gaussian Process. . . . . 77
34 Vehicle counter workflow with human annotation. . . . . 81
35 Motion annotation interface: the end with a dot indicates the starting point of a motion, the bottom of the image and the list on the right displays the counting results. . . . . 82
36 End-to-end vehicle counter workflow with scene understanding. . . . 85
37 Main interface of the web portal, cameras are displayed on the map. 86
38 Camera view with video list and summary. . . . . 87
39 Individual video view with counting information. . . . . 87
40 Snapshots of videos in our dataset, with various resolution, viewpoint, illumination, vehicle size and interactions. In particular, (c) shows shadows; (f) and (g) show severe distortion by fish-eye camera. We group these videos by their characteristics: (a) - (g) are simple low resolution videos (lowRes), and (h) - (l) are complex high resolution videos (highRes). 90

## LIST OF FIGURES (Continued)

**FIGURE** **PAGE**

## SUMMARY

With the reduced manufacturing cost of cameras and the progress in the computer vision field, intelligent transportation via computer vision has raised much attention. However, there remains a huge gap between academic computer vision research and application. There lacks enough attention to end-to-end computer vision system real-time processing speed. This thesis aims to bridge the gap between the state-of-the-art computer vision research and real-world application. We first address the critical problem of proper initialization and termination in object tracking algorithm and propose a heuristic method for automatic tracking initialization and termination in chapter 2. Then we work on learning the scene-specific semantic knowledge and apply them for other tasks such as vehicle tracking and counting in chapter 3 to 5. Chapter 7 describes our public dataset from real traffic cameras. Chapter 2, 6 and 7 consist of the work before the preliminary exam, which is a complete end-to-end vehicle tracking and counting system running in real time. We demonstrate the performance improvement by the heuristic method and further boost by the semantic knowledge.

# CHAPTER 1

# INTRODUCTION

## 1.1 Camera system for traffic engineering

Vehicles on the road generate vast amounts of data which, if effectively analyzed, provide huge potential for informing decision making. In traffic engineering, urban planners use traffic flow data to develop better traffic patterns. The standard method for collecting data involves manual counting people by a field agent. For vehicles on the road, equipment such as pressure tubes laid across the pavement, magnetic loops under the pavement (**?**; **?**) is used to collect data, but it is hard and expensive to set up and maintain.

On the other hand, the cost of cameras continues to decrease, therefore, deploying a camera system rather than traditional equipment becomes an obvious choice. The wide deployment of cameras leads to better coverage of the city, but also a massive amount of video data. The essential task for image-based data collection is extracting and efficient indexing of desired data. Currently, such tasks still primarily rely on eye-balling the interested object or event, which is inefficient. Fortunately, computer vision is becoming powerful and approaching human performance on some problems. People tend to seek more cost-efficient solution of traffic flow analysis via computer vision. With the computational advantage, computer vision techniques are expected to cut the cost and scale up those tasks.

## 1.2 Computer vision in transportation

With the progress in artificial intelligence and higher demand in its application, computer vision is becoming one of its hottest sub-domains. Computer vision enables computers to perform tasks that human is good at, such as visual detection and tracking. With the superior advantage of computational speed, these tasks can be applied in large scale.

When the fundamental theory of this field has become rather mature, researchers gradually shift their attention to more specific tasks and data. For example, pedestrian detection (**?**) and face recognition (**?**), gesture recognition (**?**) has become individual topics due to their huge application potential and high demand. On the other hand, transportation videos have also drawn much attention because of the various practical challenges and the huge impact of applications in the intelligent system field. Researchers are studying the computer vision problems specific to traffic videos, such as anomaly detection (**?**), tracking (**?**) and counting (**?**).

The videos from traffic cameras have a few unique features:

- They are recorded ceaselessly, therefore, in large quantity;
- They are usually of low-resolution quality due to the transmission and storage limits;
- They have a special composition (pedestrian and vehicle) and the objects in the view move in a regular motion pattern.
- Depending on the location, the video may contain a large number of objects with highly complex interactions.

Therefore, the complexity of the videos and the high-performance requirement raise challenges for robust intelligent transportation systems. Ideally, they are expected to perform the task with one-time setup and the minimal amount of human effort, with high accuracy and high speed.

## 1.3 Practical challenges

While people in both civil engineering and computer vision are working toward the same goal — building the real-world intelligent transportation system via computer vision, there is still a big gap between the state-of-the-art research and practice use. Researchers in academia need an in-depth study of a specific problem; therefore, they usually make simplified assumptions to isolate the problem and ignore the impact of other factors. However, the real-world application runs in a complex environment and deals with noisy data. Data cleaning and processing speed have to be taken into account. For example, researchers care more about the novelty and performance of the solution and assume the computation resources are unlimited. In this case, even though some algorithms outperform human, they are too slow to process a massive amount of data. Besides, academic problems usually have a standard benchmark with well-processed data, so that people can spend the minimal amount of time on evaluation, with a uniformly acknowledged standard. However, those data might be too small and ideal compared with the real-world data, and the evaluation metric may not be comprehensive for those complex and noisy data for different tasks.

We found out through experiment that noisy data is one major obstacle for processing real-world videos. Besides, minimal human input and real-time processing speed is the prerequisite

for large scale application. On the other hand, there lacks a standard benchmark and dataset for the researchers in the field to study the visual transportation problems and solve the bottleneck problem.

# CHAPTER 2

# FULLY AUTOMATIC VEHICLE TRACKER

(This chapter was partially published as "Fully Automatic, Real-Time Vehicle Tracking for Surveillance Video" in Computer and Robotic Vision 2017. DOI: 10.1109/CRV.2017.43)

## 2.1 Introduction

Vehicle tracking has important applications in traffic engineering. Over time, a number of vehicle counting methods have been developed, including specialized hand-held counting boards with buttons to push, pressure tubes laid across the pavement, magnetic loops under the pavement (**?**; **?**), video-based lane occupancy detectors, and more. Overall, the most powerful techniques rely on manual input and tend to be extremely labor intensive, whereas the mostly automatic techniques lack in accuracy and descriptiveness. In principle, computer vision provides the most scalable and economical alternative. Ideally, a fully automatic computer vision-based tracker follows each vehicle as it enters, traverses and exits the scene. However, current tracking algorithms such as (**?**; **?**; **?**; **?**) all require initialization as input, leading to semi-automatic tracking systems. To avoid manual input, these trackers rely on background subtraction and/or object detectors for initialization. Here, the primary challenge is robustness to variations in illumination condition, viewpoint, and video quality. Background subtraction model can fail with illumination change, while detectors are not appropriate for detecting a vehicle in the distance, or in a grainy low-resolution video, as our experiments demonstrate.

Additionally, the low throughput of most trackers prevents widely deployed visual applications, as VOT 2016 challenge reports that none of the top-ranked trackers run in real-time for even a single tracked object.

In this chapter, we propose a fully automatic algorithm for vehicle tracking that runs faster than real-time. With a sensor fusion approach, we combine background segmentation, object detection, and optical flow into a single, robust vehicle tracking system via Kalman filtering. Initialization uses the same three sources, to automatically identify moving objects in the scene. Finally, when an object exits the scene, its movements are analyzed to filter out unlikely object trajectories. To evaluate our algorithm as well as prior work, we create a hand-annotated dataset, consisting of 11 diverse, 5-minute videos collected from existing traffic cameras. For each frame, the location and extent of each moving object is provided, which enables accurate, quantitative evaluation.

We also compare the proposed algorithm against multiple state-of-the-art trackers, which rely on human input for initialization. On this dataset, we report considerably better performance than the state of the art with manual initialization, and substantial accuracy improvement when using our new automatic initialization method. Moreover, we demonstrate throughput 4× faster than real time and over 5× improvement compared to 5 out of 7 several baseline trackers, up to 47×.

## 2.2 Preliminary

The problem of vehicle tracking in existing traffic video presents some unique computer vision challenges, including scale changes, video quality (exposure control, automatic white

balance and compression), weather conditions, illumination changes, variations in perspective, and occlusion. Current work in object detection (**?**; **?**; **?**; **?**), tracking (**?**; **?**; **?**) and background subtraction (**?**; **?**) can deal with a subset of these conditions, but so far a generic system has been elusive. Below, we first introduce the underlying methods used in our system, then describe our vehicle tracking framework in detail.

### 2.2.1 Background subtraction

Background subtraction generates a binary foreground mask given a sequence of frames. Connected areas in the foreground mask can be treated as moving objects, although this technique can be error prone. We use ViBe (**?**), for its balance of speed, robustness and accuracy. However, other methods can be substituted with acceptable results in many cases.

Figure 1 summarizes four common failure cases of the background subtraction with foreground bounding boxes on the original frame on the left and the foreground on the right. Automatic exposure (Figure 1a) is performed by the camera during recording, whereas illumination variation (Figure 1b) is due to external light sources, such as the sun and vehicle headlights. Both automatic exposure and illumination variation cause rapid and widespread changes in pixel values, which most background subtraction methods struggle with. Occlusion (Figure 1c) creates a single connected foreground area out of two or more moving objects, or sometimes multiple foreground areas for a single moving object, breaking any assumption of a one-to-one mapping between foreground areas and moving object. Ghosting (Figure 1d) usually happens when a foreground object remains stationary for a long time, during which time it is gradually assimilated into the background model. When the object begins to move, what

appears from behind the object is inaccurately marked as foreground, until the background model has had time to adjust.

In summary, background subtraction provides the ability to capture small movements without manual setup beforehand. However, it is error-prone and must be compensated by other methods to create a robust vehicle tracking system.

### 2.2.2 Object detection

Object detectors (**?**; **?**) work on individual frames, scanning the image for areas that appear similar to offline training samples. Compared to background subtraction, object detection method tends to be more robust to illumination change and occlusion. However, the cost of object detection is remarkable, as it often involves an exhaustive search throughout the image, both in location and object size. Cascaded classifiers partially address this by discarding background regions (**?**). More recently, deep neural networks (**?**; **?**) have emerged as a promising approach to object detection. We use a state-of-the-art detector called faster-RCNN (**?**). Running on a high-end graphics processing unit (GPU), the time required for detection on one image drops from 2 seconds (**?**) to 198 ms on the PASCAL 2007 dataset, making the real-time detection in video feasible. However, like other detectors, faster-RCNN still has missing and false detections. In our measurements, it has a missing rate in excess of 65% and 86% on high- and low-resolution videos, respectively. Thus, given the high miss rate, especially on poor quality images, object detection alone is not suffice for a robust vehicle tracking system.

(a) Camera auto-exposure.

(b) Illumination change.

(c) Occlusion. Note also the shadow, due to illumination change.

(d) Ghosting. Vehicles stopped at red light have become part of the background model.

Figure 1: Common background subtraction failure cases. Pixel values change for many reasons other than motion.

### 2.2.3 Optical flow

Optical flow is an estimate of the movement of pixels between two images: in our case, two consecutive video frames. Optical flow provides a low-level description of motion in images and can offer useful evidence for tracking applications. Estimating optical flow is a research area in its own right, but we use the seminal Lucas-Kanade algorithm (**?**) in our system, as it runs fast on GPUs, and provides useful results while making minimal assumptions about the underlying scene and image. Figure 2 illustrates two optical flow problems that may affect tracking accuracy. The left column shows the direction and magnitude of the optical flow vectors, while the right column is the color code visualization of the optical flow results, with the color wheel at bottom right corner of Figure 2b indicating the corresponding direction. Figure 2a illustrates the so-called aperture problem, where the center of the truck has no reported optical flow, due to its large and uniformly colored surface. Figure 2b illustrates the "turbulent", error-prone flow that occurs where objects traveling in opposite directions meet.

Thus, while *accurate* optical flow estimates offer valuable information about movement in the scene, it is neither complete (due to the aperture problem), nor free of severe estimation errors, in particular near occlusion boundaries.

(a) Aperture problem.

(b) Occlusion “turbulence”.

Figure 2: Common problems in optical flow estimation.

## 2.3 Heuristic tracker by Kalman Filter

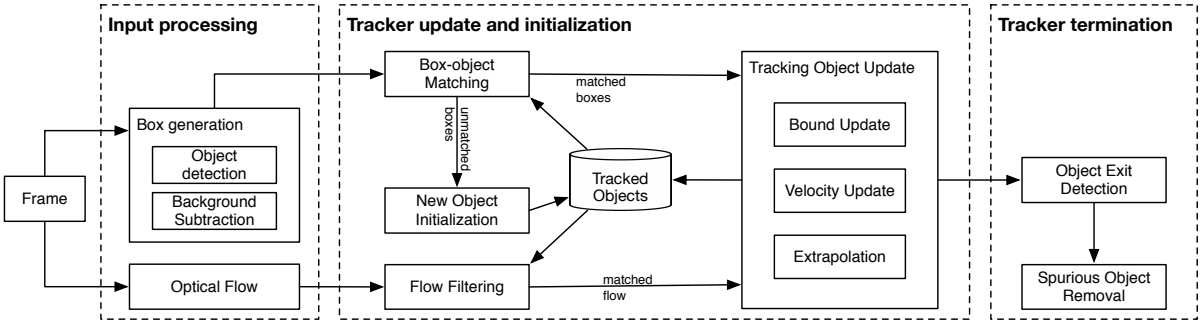



Figure 3: Overview of proposed system. Separate Kalman filter state is initialized, maintained and terminated for each tracked object. The state is updated based on matching input from background subtraction, object detection and optical flow.

Figure 3 describes the workflow of our automatic tracking application. We first apply background subtraction and object detector on each frame. This generates two sets of candidate boxes, which are used to initialize and update trackers. Each tracker is represented by an individual Kalman filter (**?**). Optical flow is also computed. Any flow that matches a tracked object both by location and velocity, is used for tracker update. Tracking is terminated based on object location, velocity and time; short-lived or otherwise spurious objects are filtered out.

### 2.3.1 Model definition

We use Kalman filter to smooth out the noise in the observed measurements by the aforementioned components and more importantly, to integrate the strengths and compensate the

weakness of each component. The *prediction* by its linear model is *corrected* with measurements observed over time, therefore the generated estimation is much smoother despite the noise in measurement input. To model the state change, a discrete-time controlled process is modeled as linear stochastic difference equation (**?**), state at time $k$ is a linear combination of its previous state $\mathbf{x_{t-1}}$ at time $t-1$ and the process noise $\mathbf{w}_{t-1}$:

$$\mathbf{x}_k = A\mathbf{x}_{t-1} + \mathbf{w}_{t-1}, \tag{2.1}$$

where $A$ is the state transition model. In our case, A models how object move on the frame along time, presumably satisfies constant acceleration movement. There is also measurement $\mathbf{z} \in \mathbb{R}^m$ formulated as

$$\mathbf{z}_t = H\mathbf{x}_t + \mathbf{v}_t. \tag{2.2}$$

Here $H$ is the measure model, which observation to correct prediction from previous step. $\mathbf{w}_t$ in Equation 2.1 and $\mathbf{v}_t$ in Equation 2.2 are process and measurement noise respectively. They are assumed independent of each other and zero-mean Gaussian, with $Q$ and $R$ as process noise covariance and measurement noise covariance, respectively.

$$\begin{aligned} p(\mathbf{w}) &\sim \mathcal{N}(0, Q) \\ p(\mathbf{v}) &\sim \mathcal{N}(0, R) \end{aligned} \tag{2.3}$$

The recursive process of Kalman filter contains two steps: time update (prediction) and measurement update (correction). Both steps are applied at time $k$, indicated by subscripts below.

For such a continuous system, we define a time unit $dt$, which is the time interval we perform an update, in our case, the time between two consecutive frames. Each variable has its own value at a certain time step $t$, indicated by the subscript. The prediction is performed as follows:

$$\begin{aligned} \hat{x}_t^- &= A\hat{x}_{t-1} \\ P_t^- &= AP_{t-1}A^T + Q \end{aligned} \tag{2.4}$$

Here $\hat{x}_{t-1}$ and $\hat{x}_t^-$ are internal states before and after prediction at time $t$. $P_{t-1}$ and $P_t^-$ are prior and post error covariances, and $Q$ is the process noise covariance. In our case, we define the internal state a 10-dimensional vector:

$$\mathbf{x} = [x, y, w, h, x', y', w', h', x'', y''],$$

corresponding to the object's top left location $(x, y)$, size $(w, h)$, as well as the velocity $(x', y')$, rate of growth $(w', h')$ and acceleration $(x'', y'')$, respectively. The dynamic model $A$ is defined based on the physics equation of displacement with velocity and acceleration. With the assumption that the object has constant acceleration within $dt$, we have:

$$\begin{aligned} x_t &= x_{t-1} + x'_{t-1} \cdot dt + \tfrac{1}{2} \cdot x''_{t-1} \cdot dt^2 \\ y_t &= y_{t-1} + y'_{t-1} \cdot dt + \tfrac{1}{2} \cdot y''_{t-1} \cdot dt^2 \\ x'_t &= x'_{t-1} + x''_{t-1} \cdot dt \\ y'_t &= y'_{t-1} + y''_{t-1} \cdot dt. \end{aligned} \tag{2.5}$$

For width and height, we instead assume constant growth rate within $\mathtt{dt}$, thus

$$\begin{aligned} w_t &= w_{t-1} + w'_{t-1} \cdot \mathtt{dt} \\ h_t &= h_{t-1} + h'_{t-1} \cdot \mathtt{dt} \end{aligned} \tag{2.6}$$

After prediction, the estimated state $\hat{\mathbf{x}}_{\mathbf{t}}^{-}$ is corrected by an observed measurement $\mathbf{z}$ at each time step by the steps below:

$$\begin{aligned} K_t &= P_t^{-} H^T (H P_t^{-} H^T + R)^{-1} \\ \hat{\mathbf{x}}_{\mathbf{t}} &= \hat{\mathbf{x}}_{\mathbf{t}}^{-} + K_t(\mathbf{z_t} - H\hat{\mathbf{x}}_{\mathbf{t}}^{-}) \\ P_t &= (I - K_t H) P_t^{-} \end{aligned} \tag{2.7}$$

In our case, we have a 10-dimensional measurement vector:

$$\mathbf{z} = [x^{bg}, y^{bg}, w^{bg}, h^{bg}, x^{det}, y^{det}, w^{det}, h^{det}, v_x, v_y],$$

which represents the top left coordinates $(x, y)$, width $(w)$, height $(h)$, reported by background subtraction $\mathtt{bg}$ and detector $\mathtt{det}$, separately, as well as the velocity $(v_x, v_y)$ reported by optical flow estimation. The measurement model $H$ is a 10-by-10 matrix of zeros except for ones at $(0,0)$, $(1,1)$, $(2,2)$, $(3,3)$, $(0,4)$, $(1,5)$, $(2,6)$, $(3,7)$, $(4,8)$, $(5,9)$, signifying that the background and detector boxes directly measure $x$, $y$, $w$, and $h$, and that optical flow directly measures $x'$ and $y'$. In other words, there are no direct measurements of $w'$, $h'$, $x''$ or $y''$ since they are not observable. $R$ is the measurement noise covariance, indicating the noisiness of measurement. By

manipulating the measurement noise covariance $R$, we compute a Kalman gain $K_t$, indicating the weight of the measurement to update the corresponding prediction.

### 2.3.2 Measurement acquisition

None of our input measurements: background subtraction, object detection and optical flow, correspond directly to a single tracked object. Instead, they generate boxes and flow indications for an entire frame. To produce input to an individual tracked object's filter, we first compute a matching of input data to tracked objects, then apply the matched boxes and flow to the corresponding object's Kalman filter state.

**Background subtraction and object detection:** Both background subtraction and object detection generate bounding boxes $\{x, y, w, h\}$. We only take those consistent with tracker's current state as the measurement. For each Kalman filter, the best matching box as measurement maximizes the total overlap between the predicted bounds of internal filter, and the bounds of the measured boxes. Boxes must overlap with the predicted state in order to be considered a match. Any remaining boxes are used to initialize new tracked objects.

**Optical flow:** Optical flow reports velocity on a pixel-by-pixel basis, with many erroneous flow vectors due to the aperture and turbulence problems described earlier. To produce a velocity measurement from the optical flow field for a single object, we first denoise the flow field by forward-backward error thresholding, as described in (**?**). We then filter the flow by location, velocity magnitude and direction. In particular, we only consider flow vectors that originate from the location of the object in the previous frame, follow the similar direction (within in 45°) of the previous state – to reduce the effect of turbulence problem, and only

keep those vectors that fall within twice the error covariance (available in $\mathsf{P}$, diagonal entries corresponding to the object velocity in $\hat{\mathbf{x}}$), using the simplifying assumption that flow errors follow a normal distribution. Finally we use the mean of the remaining flow vectors on horizontal and vertical direction, respectively, as the optical flow measurement for the object.

### 2.3.3 Model update

The measurements above are directly plugged into Equation 2.7, however, are of different quality, and oftentimes some of the measurements are missing entirely. For example, the invalid foreground is generated when occlusion happens; detector misses a small-sized object, or aperture problem gives zero movement of object. We address these problems by varying the measurement noise covariance accordingly, which in turn causes the Kalman filter to choose a gain $\mathsf{K}$ that maximizes the quality of the internal state. Recall in Equation 2.7, a large measurement covariance $\mathsf{Q}$ results in a smaller $\mathsf{K}$, therefore the measurement weights less in correction. When a measurement is missing, we use the value already in $\hat{\mathbf{x}}$ as a proxy, and set the covariance to $\infty$. Consequently, once none of the three measurements is available, the tracker merely relies on the internal prediction, which we call *extrapolation*. The tracker still generates tracking results by the internal linear model during extrapolation. The other two cases in Figure 3 are when at least one bounding box is available (bound update) or only optical flow is obtained (velocity update).

We also apply error gating to the background subtraction and object detection measurements. For example, foreground bounding box that is well overlapped with tracked objects (more than 30% overlap) has a higher confidence than those with other bounding boxes around.

Similarly, optical flow have a lower confidence with zero value on both directions, since we have no idea whether the object is still or the aperture problem happens. In addition, as described in §2.2, the three measurement types naturally have different error covariances. Although detector has a higher missing rate, the recall is also high. Therefore, measurement from detector has a smaller noise covariance value than those from background model.

## 2.4 Fully automatic initialization and termination

### 2.4.1 Object entry and exit

Currently available datasets usually have tracked objects in the center of the first frame, however, we have initialization more challenging when objects enter the scene in a variety of ways: approach from a distance, enter from the image boundary, appear from behind an occluding object — moving or stationary, and become visible due to changes in lighting or background conditions. Termination has similar challenges — vehicles may disappear temporarily behind obstructions or due to changing conditions, they may linger near the edge of the screen, exit the scene while behind a moving vehicle, or disappear slowly into the distance. It is usually natural to terminate tracking when sequence ends or object leaves the scene in short sequences, whereas in practice, it is hard to distinguish between exiting and temporal occlusion when the object is not visible in long-time videos.

As automatic initialization and termination are missing in the majority of current tracking literature, the generally accepted methods either heuristically manually label an entry/exit area beforehand, under the assumption that objects always enter or exit within a certain area, or rely on fully manual initialization. The first method is too simplistic for general purpose vehicle

tracking, and the second is impractical under constrained expense/time budget for large-scale use.

In the only available literature that explicitly addresses the effect of tracker initialization (**?**), the author concludes that slight temporal and spatial variation results in performance difference. However, unlike the currently available dataset, vehicles frequently encounter significant scale change while leaving or entering the scene in the traffic videos available to us. This presents a unique problem for initialization, as early initialization results in a small, poor-quality image, and late initialization results in missing of information due to the short trajectory. Thus, striking the right balance between tracking performance and lifetime is the key to automatic tracker initialization.

### 2.4.2 Our method

Our system initializes new trackers based on unmatched boxes from background subtraction and object detection. This supports the challenging cases described above, but can result in many spurious trackers due to the noisy nature of both background subtraction and optical flow. We use a two-pronged approach to limit such spurious results. First, initialization is limited to objects larger than 10×10 pixels. This reduces the number of trackers in flight, without significant negative effect: cars that are reasonably captured tend to be larger than that in our videos, except when approaching from or driving toward the vanishing point. Second, trackers are terminated when the object leaves the frame, after no direct observations (boxes or optical flow) has been made for 50 frames, in other words, in *extrapolation* mode. Any object has such 50 frames before exit. However, upon termination, tracked objects are validated based on the

number of observations and distance traveled. Spurious noise tends to be stationary and short-lived, whereas vehicles typically follow a continuous and long-lived path through the scene. To adapt to the variety of videos and objects, distance threshold is dynamically computed by object size and video resolution. One good consequence of such scheme is that early initialized noise is quickly discarded, since usually there is no consistent measurement available for them, while those tiny objects discarded as noise is soon available for future initialization, with better quality. Therefore, real small objects are able to be initialized at the earliest point and survive with a complete trajectory.

Algorithm 1 to 3gives the pseudo code for our method. Since it is uses heuristic criteria for initialization and termination, we call it the heuristic tracker.

**Algorithm 1** Heuristic tracker.

**for** each frame $t$ **do**
  Obtain foreground boxes $\mathbf{R}_{bg} = \{\mathbf{r}_{bg}\}$.
  Obtain detection boxes $\mathbf{R}_{det} = \{\mathbf{r}_{det}\}$.
  **for** each $i$ th object $O_i$ at $\mathbf{r}_{t-1}(i)$, $(i = 1, \ldots, N_t)$ **do**
    Find the most matched boxes $\mathbf{r}_{bg}(i)$ and $\mathbf{r}_{det}(i)$ with $\mathbf{r}_{t-1}(i)$.
    Compute mean velocity $\boldsymbol{\nu}(i)$ within $\mathbf{r}_{t-1}(i)$ from the optical flow.
    **if** $\boldsymbol{\nu}(i) = 0$ or neither $\mathbf{r}_{bg}(i)$ or $\mathbf{r}_{det}(i)$ exists **then** Enter extrapolation mode.
    **else**
      $\mathbf{R}_{bg} = \mathbf{R}_{bg} \setminus \mathbf{r}_{bg}(i), \mathbf{R}_{det} = \mathbf{R}_{det} \setminus \mathbf{r}_{det}(i)$.
      Make measurement for tracking update $z_t(i) = [\mathbf{r}_{bg}(i), \mathbf{r}_{det}(i), \boldsymbol{\nu}(i)]$.
      Set measurement covariance error $R$.
      Update object tracker with $z_t(i)$ and $R$, get result $\mathbf{r}_t(i)$.
    CheckExitHeuristic($O_i$)
  **for** each remaining box candidate $r \in \{\mathbf{R}_{bg} \bigcup \mathbf{R}_{det}\}$ **do**
    CheckEntryHeuristic($\mathbf{r}$).

**Algorithm 2** CheckExitHeuristic(O)

1: **if** Object O is in extrapolation mode more than 50 frames **then**
2: exit object O.

**Algorithm 3** CheckEntryHeuristic(**r**)

1: **if** r is larger than $10 \times 10$ **then**
2: initialize object.

## 2.5 Evaluation

### 2.5.1 Tracker configuration

To better evaluate how each component contributes to the tracker in videos under various conditions, we use three different configurations of our tracking system, based on the inputs used. BG uses background subtraction and optical flow, DET uses the object detector and optical flow, and BG+DET uses all three inputs. We also run several state-of-the-art trackers, including KCF (**?**), STRUCK (**?**), ASMS (**?**), DAT(**?**), staple (**?**), MEEM (**?**) and SRDCF (**?**) as baselines. Since all these trackers require manual initialization (namely human input), they are not able to be compared with our tracker directly. Instead, we can only partially evaluate

them by providing manual initialization, see §2.5.4. By carefully selected manual initialization, the comparison here is in favor of the baselines.

### 2.5.2 Evaluation metrics

Given a set of generated and ground truth object trajectories, usually represented by a sequence of rectangular bounding boxes, we desire one or more performance metrics that capture the accuracy of the proposed system. Aspects that need to be captured include 1) tracking duration—if a ground truth trajectory of an object contains N frames, for how many of these frames does the system track the object, 2) recall—how many of the ground truth trajectories were represented in the tracking result, 3) precision—what proportion of tracking results had a corresponding trajectory, as well as 4) the overlap between the tracking result and the ground-truth.

Note that our problem does not fit into common multi-object tracking setting, since moving objects are tracked individually, instead of being modeled globally. Common metrics such as MOTA (**?**) generates a meaningless number since we can have potentially more objects tracked than ground truth.

As the first step, we match each ground truth to a tracking result, by maximizing the accumulated overlap ratio $r$,

$$r = \frac{\sum_t (S_t^{gt} \cap S_t^{traj})}{\sum_t (S_t^{gt} \cup S_t^{traj})}, \tag{2.8}$$

where $S_t^{gt}$ and $S_t^{traj}$ are ground truth and trajectory box areas at frame $t$, respectively. Note that the sum is computed over the union of trajectory and ground truth lifetime. Equation 2.8 ensures the ground truth is matched to a longer trajectory with reasonable coverage. Additionally, to filter some spurious objects slightly touched the ground truth, we also require any match to have more than 30% overlap on at least one frame, where the value is set experimentally.

### 2.5.3 Object-level evaluation

Given the matched results, we can calculate the proportion of ground truth trajectories that were matched to the tracking result (recall), and the proportion of tracking results that had a matching trajectory (precision), for our three trackers and two types of videos. Figure 4 and Figure 5 show the results in absolute numbers, also to provide some perspectives on the scale of the evaluation. The true positives are those correctly tracked objects and false positives are noisy objects. Our best tracker configuration, BG+DET, achieves 81% recall and 87% precision on our low resolution, simple videos, and 65% recall, 57% precision on the high-resolution, complex scenes. Here, the majority of failures in the complex scenes were due to complicated interactions and heavy occlusion throughout the scene, where vehicles were only partially visible. Currently, we do not split such merged objects into their constituent parts; we leave this for future work.

Comparing the three configurations, we see that the detector performs quite poorly on the low-resolution scenes, due to small and poor quality imagery where it is sometimes difficult even for a person to correctly identify an object as a vehicle. By contrast, background subtraction does quite well on these uncomplicated scenes. For the more higher resolution complex scenes,

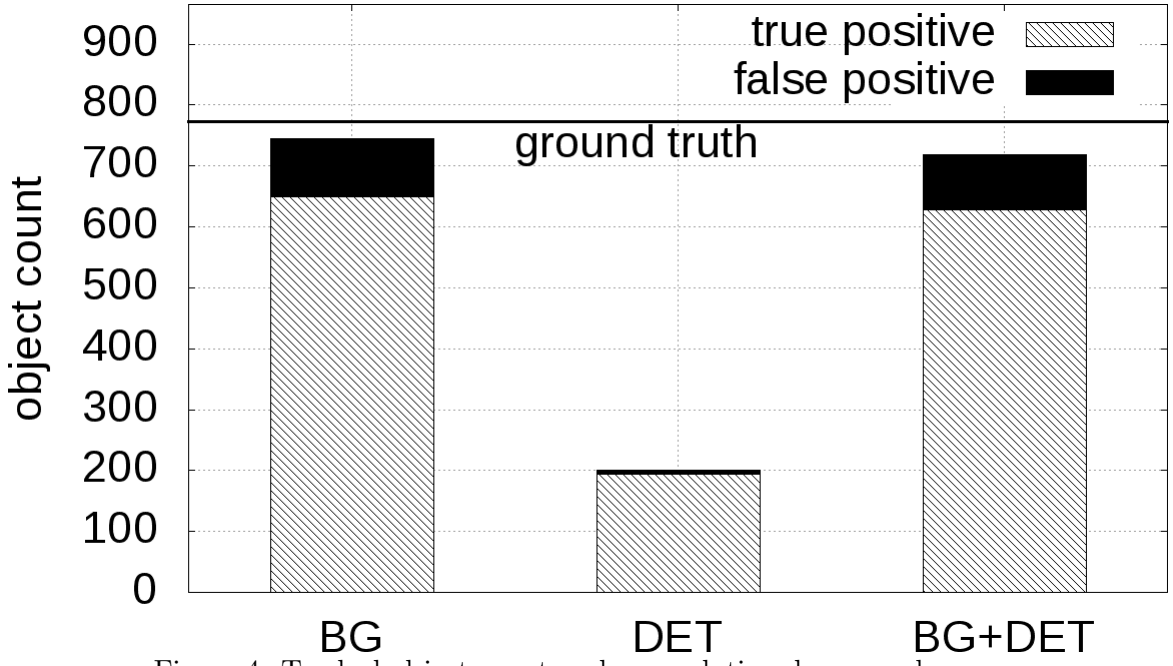



Figure 4: Tracked object count on low resolution, less complex scenes.

the detector performs well, while the background subtraction model struggles and largely introduces noise.

#### 2.5.4 Pixel-level evaluation during entire lifetime

We are also interested in evaluating how well the tracker follows objects in the scene. For example, as one of our motivating application, accurate tracking is necessary for correct turning movement classification, as Figure 40a illustrates. While there exists no similar work or benchmark that incorporates automatic initialization, we can only partially compare our system to prior work. Here, we modify our tracker to accept manual initialization. Both our modified trackers and the baseline trackers are initialized with boxes extracted from ground

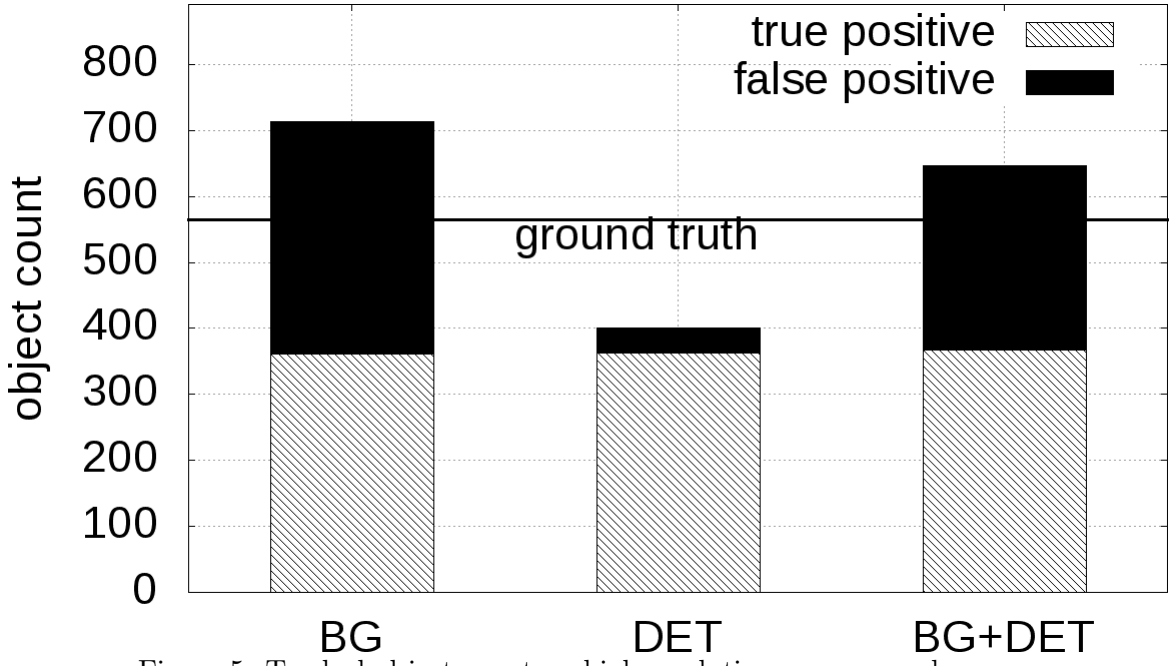



Figure 5: Tracked object count on high resolution, more complex scenes.

truth. These "initialized" trackers are then evaluated along with our automatically initialized trackers. Note that the experiment is designed in favor of those manually initialized trackers, since the initialization boxes are from ground truth and perfectly match the object in the scene.

To accept manual initialization in scenes with significant scale change, each tracker is initialized with the first box of ground truth larger than a threshold:

$$S_{\text{thresh}} = \left[\max(S^{gt}) - \min(S^{gt})\right] * s + \min(S^{gt}), \tag{2.9}$$

where $\max(S^{gt})$ and $\min(S^{gt})$ are the maximal and minimal ground truth areas along the sequence. $s \in \{0, 0.2, \ldots, 1.0\}$, corresponds to the minimum fraction of the maximum object

size at which initialization may occur. Note $s$ only affects objects that enter with an increasing size; while shrinking objects can be initialized upon appearance regardless of the threshold. Tracking is terminated at the last frame of ground truth. We arrived at this design as some trackers tend to perform poorly when initialized by small images, yet delaying initialization until the object grows large can result in arbitrarily shortened trajectory.

Figure 6 and Figure 7 compare the overlap ratio of our three tracker configurations both with manual and automatic initialization (horizontal lines), as well as the overlap ratio of several other trackers on our high- and low-resolution datasets. The x-axis is the initialization threshold $s$ in Equation 2.9, and the y-axis is the overlap ratio $r$ from Equation 2.8, averaged over all objects. This is computed from the first to the last frame of ground truth for the manually initialized and terminated trackers. While for our automatic trackers, it is computed over the union of tracking and ground truth lifetimes. This accounts for both tracking accuracy and lifetime, as we illustrated before.

The shape of the curves for the initialized trackers captures the trade-off between early-and-inaccurate, vs. late-but-accurate initialization. Overall, our BG-based trackers handily outperform other trackers when manual initialization and termination is provided. More importantly, the auto-initialized BG trackers essentially meet the performance of other trackers on the simple videos, and significantly outperform them on the complex videos. We hypothesize that the automatic initialization allows the tracker to better adapt to individual vehicles vs. the constant threshold set for manual initialization. Additionally, consistent with our conclusion

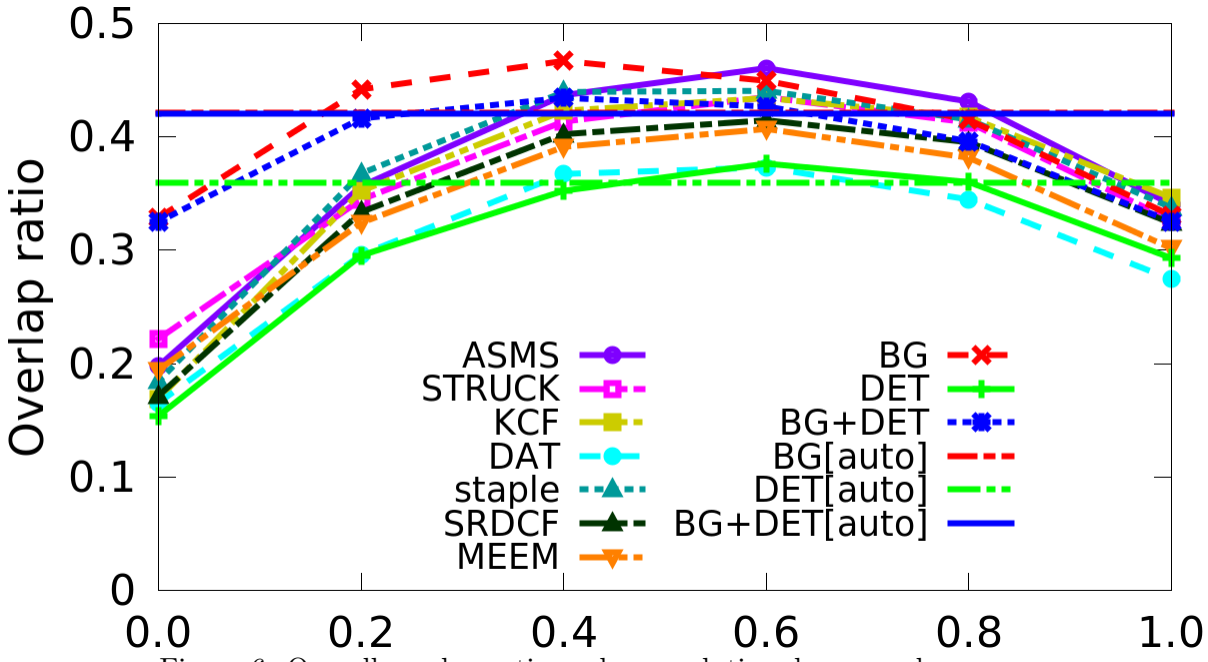



Figure 6: Overall overlap ratio on low resolution, less complex scenes.

in §2.5.3, detector dominates the tracking update on high resolution videos, while background subtraction plays its role on low resolution videos.

### 2.5.5 Pixel-level evaluation during tracked period

Then we focus on the performance during only the tracked period. Borrowed from standard single object tracking measurement, we show the success plot for trackers for two video groups, in Figure 8 and Figure 9. Different from above, we compute the success frame rate from the initialization frame, instead of entire ground truth sequence. The performance is measured by the area under the curve. By Figure 6 and Figure 7 we see overlap ratio between 0.2 and 0.6 is a good balance of tracking accuracy and trajectory completeness, we hereby show results

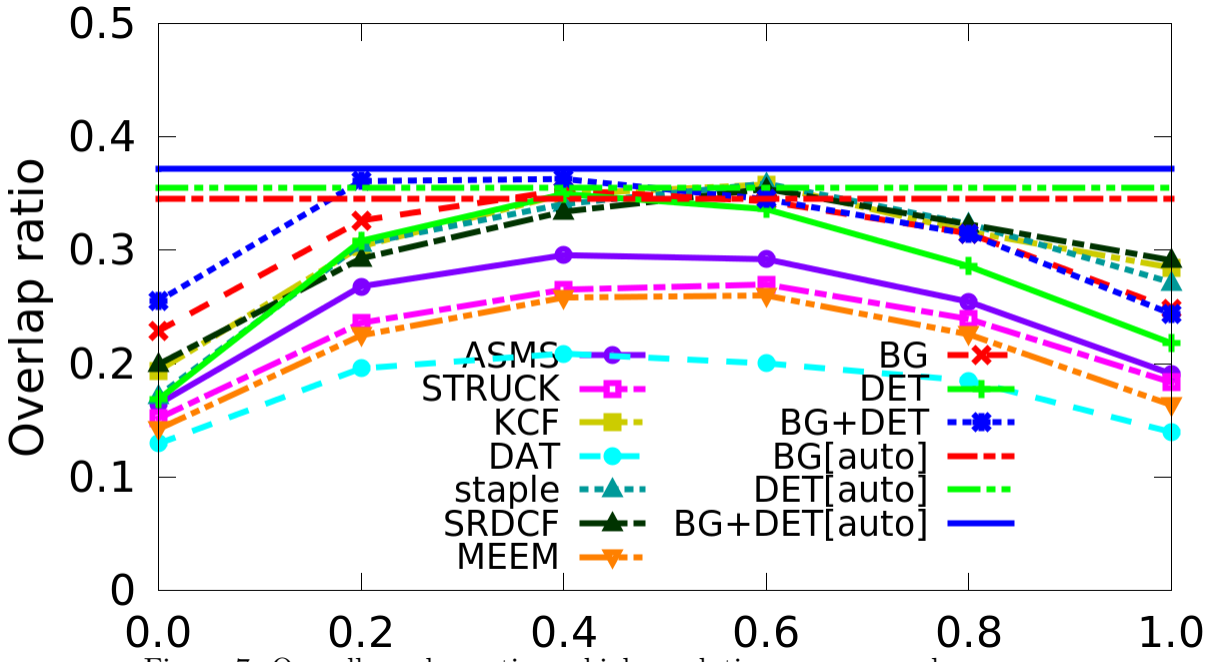



Figure 7: Overall overlap ratio on high resolution, more complex scenes.

of initialization threshold of 0.4 here. Our automatic trackers significantly outperform other trackers with manual initialization. This demonstrates that proper initialization is critical to the tracking performance and similar conclusion about the contribution of each tracking component can be drawn.

### 2.5.6 Throughput

Finally, we explore the throughput of trackers on different resolution videos, as shown in Figure 10. The evaluation was done on a Linux desktop with an Intel Core i7-3770 3.40GHz processor and a GeForce GTX TITAN X GPU. According to these results, BG significantly outperforms 6 out of 7 baseline trackers, with around 5× throughput improvement compared

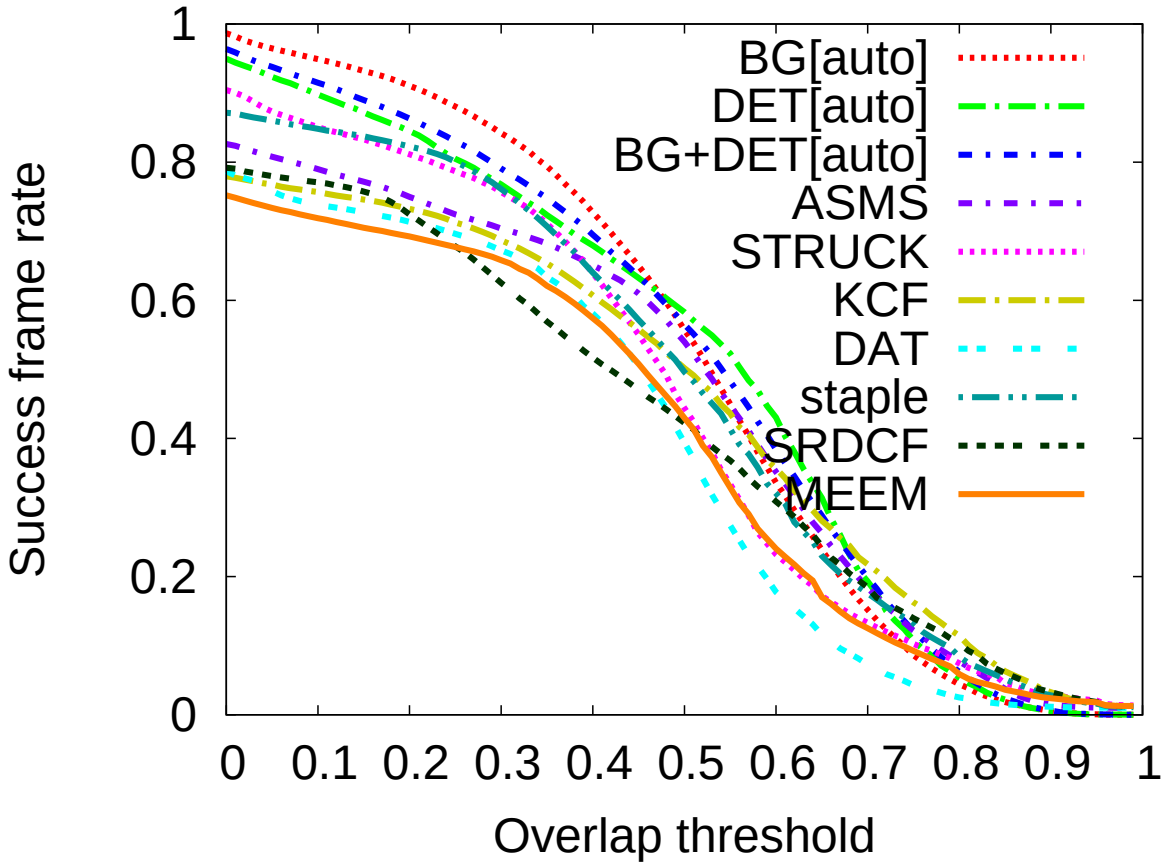



Figure 8: Success plot of low resolution, less complex scenes.

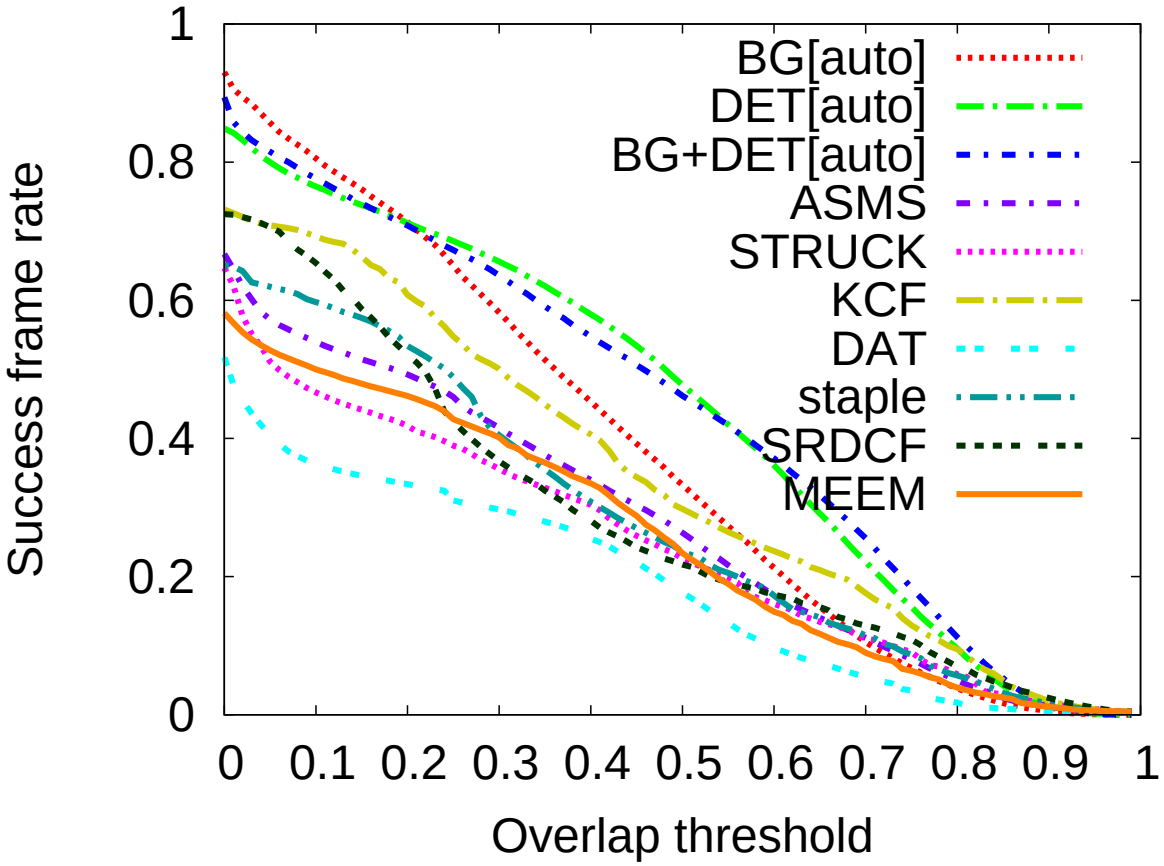



Figure 9: Success plots of high resolution, more complex scenes.

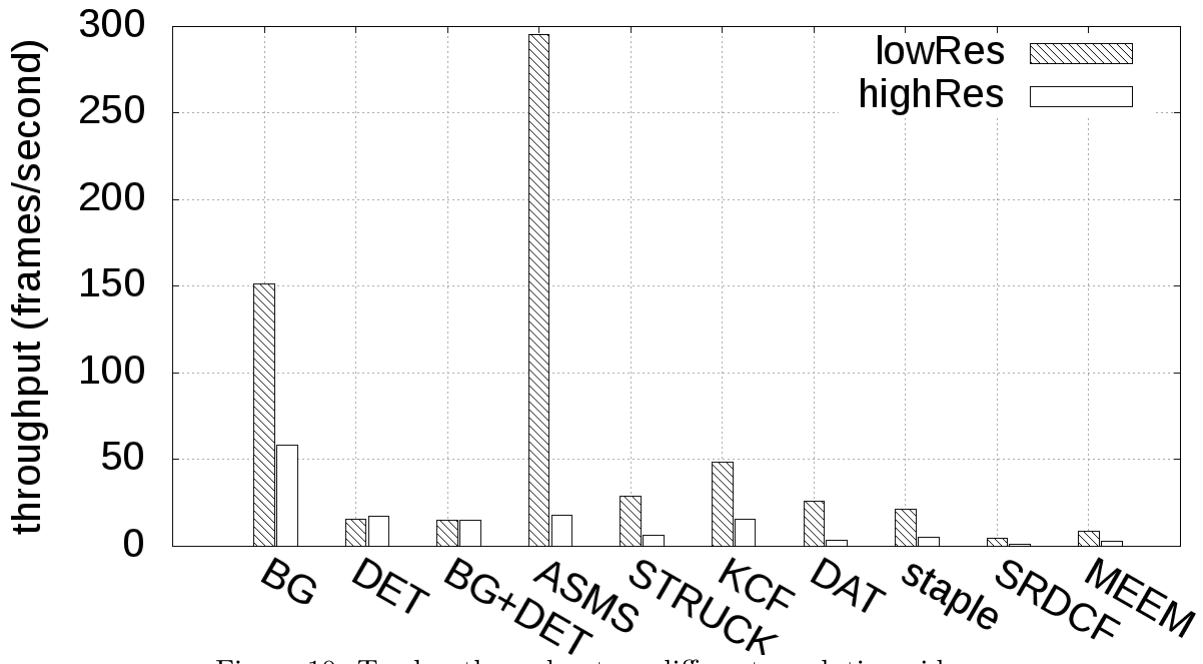



Figure 10: Tracker throughput on different resolution videos.

with real time (about 30 fps). Interestingly, ASMS outperforms our BG tracker on low resolution videos, however, it becomes 17× slower once it is applied on high resolution videos. Although the use of the object detector (DET, BG+DET) makes the tracker slower, they are the only ones that maintain a similar frame rate on both low- and high resolution videos near real time, showing good scalability of resolution.

# CHAPTER 3

# SCENE LEARNING

## 3.1 Introduction

For a traffic camera mounted at a static location, objects in the recorded videos move along the road geometry with a regular pattern. Those movement patterns can be useful for video analysis. For example, objects that do not follow the usual movement pattern are likely to be anomalous; therefore, they are worth special attention. Besides, statistics on different movements provide analysis on a finer granularity. For example, Illinois Department of Transportation (IDOT) keeps track of the vehicle counts in different moving directions, for traffic cameras all across Illinois.

It is usually intuitive for a human to identify the movement patterns after watching such a video for a while. However, human's interpretation lacks a mathematical representation and may vary among different people. Also, people are prone to making mistakes when dealing with complex videos; their performance may tend to decrease with longer working hours.

As a result, it is essential to automate this process for fast and large scale processing. Some researchers are working on clustering existing trajectories and obtain a semantic representation of the scene (**?**; **?**). However, such a method is usually sensitive to noise. More importantly, it is often hard to get clean trajectories via object tracking algorithms. An additional step of data cleaning is necessary to make the clustering method work. On the other hand, compared

with object-level trajectories across multiple frames, pixel movement on individual frames is a more robust input for the scene understanding problem. It excludes the interference of object interaction and forms a lower-level summary of the scene. In this chapter, we apply a non-parametric clustering method (**?**) to learn the vehicles' movement pattern in an unsupervised manner. The resulting clusters have a mathematical representation in the form of a set of distributions; therefore, they are easier to interpret in the subsequent processing.

## 3.2 Atomic motion extraction

Tracking performance can be improved with the help of scene semantics. For example, objects may have nonlinear changes in their sizes and speeds as they move due to camera perspectives, and characterization of such non-linearity can regulate and benefit the tracking of the objects. We propose an unsupervised framework, as shown in Figure 11, to capture and exploit these constraints. First, to capture global and long-range trajectories with rich semantics, local motion patterns of smaller units, such as fine-grained grids, are extracted (step 1). We capture the local motion directions, which can be different but spatially interdependent across the grids, via a non-parametric topic model called HDP (**?**; **?**). For any scene, the model flexibly captures any local motion directions, which are mixtures of base directions. Then, we synthesize global trajectories from the local patterns (step 2). Specifically, we adopt the trajectory finding algorithm from (**?**) but improve it by: (i) identifying multiple significant trajectories rather than a single one, and (ii) sampling multiple possible instances from the same underlying trajectory to capture richer geometric information rather than a single line. The sampled instances reveal key information for tracking: (i) hotspots where vehicles enter

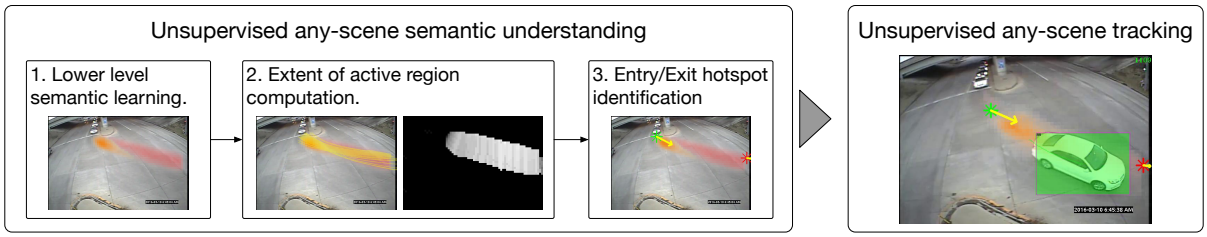



Figure 11: System overview. The left corresponds to scene learning module: frequent motions are extracted in an unsupervised fashion, then the active regions where most objects move are learned on top of the motion result. Next, the entry/exit hotspot and their direction are extracted, describing how most objects enter and exit the scene. On the right, the semantic knowledge is applied to a tracker.

and exit the scene; and (ii) the extent in the direction perpendicular to the principal direction (namely the direction from the starting to the ending point) that changes nonlinearly but smoothly along that trajectory. Lastly, we integrate the discovered hotspots and trajectories in a tracking model as constraints to significantly reduce the false positives with improved or comparable false negatives.

### 3.2.1 Non-parametric clustering via HDP

For the general purpose of scene learning, lower-level representation is a much more robust feature due to the difficulty of obtaining an accurate higher-level result, such as consistent object trajectories. A cluster of such low-level data as pixel movement is an informative summary. Inspired by the previous work (**?**; **?**), motion patterns can be learned by topic model, with an analogous bag-of-words representation to document classification.

Videos are processed as described in Figure 12: first every frame is divided into small grids — small enough to have consistent optical flow results. To distinguish different motions,

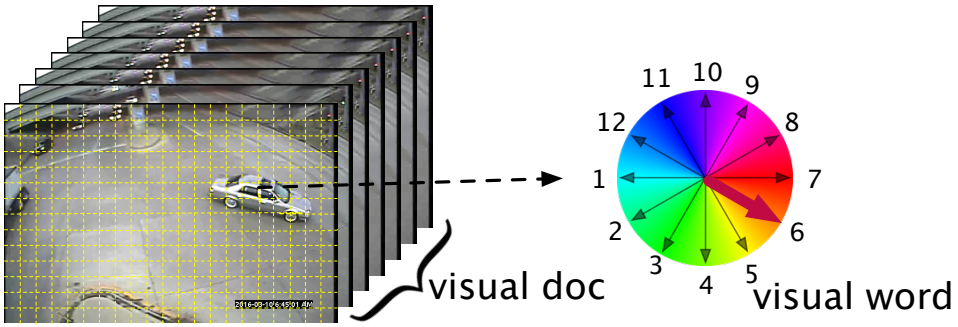



Figure 12: Left column illustrates the spatial and temporary quantization of video frames. The right shows a visual word obtained by discretizing the optical flow direction.

optical flows of consecutive frames are quantized on $D$ directions after some simple filtering (**?**). Making an analogy with to document clustering, on a certain frame, each optical flow at a grid coordinate $(x, y)$ on a quantized direction is a *visual word*. If the frame is divided into $w \times h$ grids, the vocabulary is $V = w \times h \times D$. Please refer to §3.4 for the values we use for the above parameters. As shown in the middle column of Figure 12, the videos are then split temporarily into short video clips. Each video clip has enough frames to contain a complete motion, at the same time, is not too long to have mixed motions. By counting the number of visual words on each grid and each quantized direction in every video clip, we have a bag-of-words representation of *visual documents*. Similarly, a cluster trained from the video is called *visual topic*.

Using the bag-of-words representation, motions can be learned by any existing topic model method. In most clustering methods, the number of clusters is required as an input, and the results usually vary accordingly. For complex videos, it is presumably hard for people to identify

the number of major motions. Therefore the non-parametric clustering method —HDP is used. It is a generalization of Dirichlet Process (DP), requiring no specification of the cluster number. In this model, visual words contain groups of observations, each exhibiting a mixed proportion of shared mixture components. The mixture components are learned in this model, also called "topics". More specifically, an HDP is a two-level DP with shared parameters. Each group $j$ is associated with a draw from a shared DP whose base distribution is also a draw from the top-level DP.

$$\begin{aligned} G_0 &\sim \mathrm{DP}(\gamma, H) \\ G_j|G_0 &\sim \mathrm{DP}(\alpha_0, G_0), \text{for each } j, \end{aligned} \tag{3.1}$$

Here $j$ is the group index. At the top-level, the distribution $G_0$ is drawn from a DP with concentration parameter $\gamma$ and base distribution $H$. The base distribution $H$ is a symmetric Dirichlet over the vocabulary simplex. Each atom is a distribution over the vocabulary, the atoms $\boldsymbol{\phi} = (\phi_k)_{k=1}^{\infty}$ are drawn independently $\phi_k \sim \mathrm{Dirichlet}(\eta)$. Since the numbers drawn from a Dirichlet distribution sum up to 1, they are usually used as the parameters of a multinomial distribution. Simply speaking, $G_0$ is a discrete distribution over the atoms $\boldsymbol{\phi}$. At the bottom level, $G_0$ is used as the base distribution to draw each group distribution $G_j$, each is another multinomial distribution over the atoms $\boldsymbol{\phi}$. By such hierarchical definition, the atoms are shared among $G_j$, which makes sense that different documents may belong to the same topic.

In our setting, a group is a visual document and the $i$th word of the $j$th document $x_{ji}$ is drawn as follows:

$$\theta_{ji} = \phi_{z_{ji}}, \quad \theta_{ji}|G_j \sim G_j, \quad x_{ji}|\theta_{ji} \sim \mathrm{Multi}(\theta_{ji}) \tag{3.2}$$

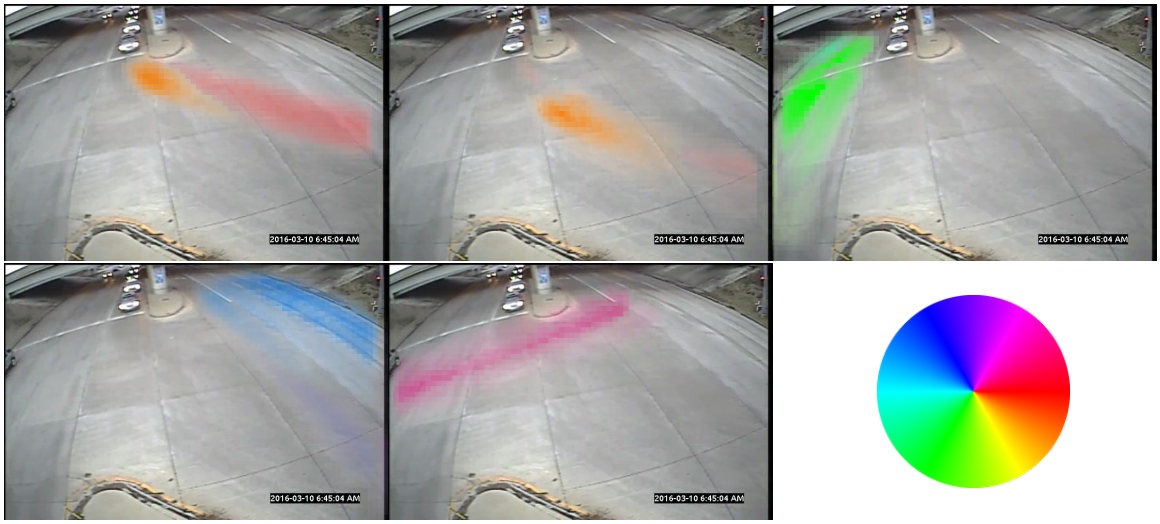



Figure 13: Motions learned by HDP, colors indicate directions as the color wheel shows, the lightness indicate the magnitudes of the maximal probability values..

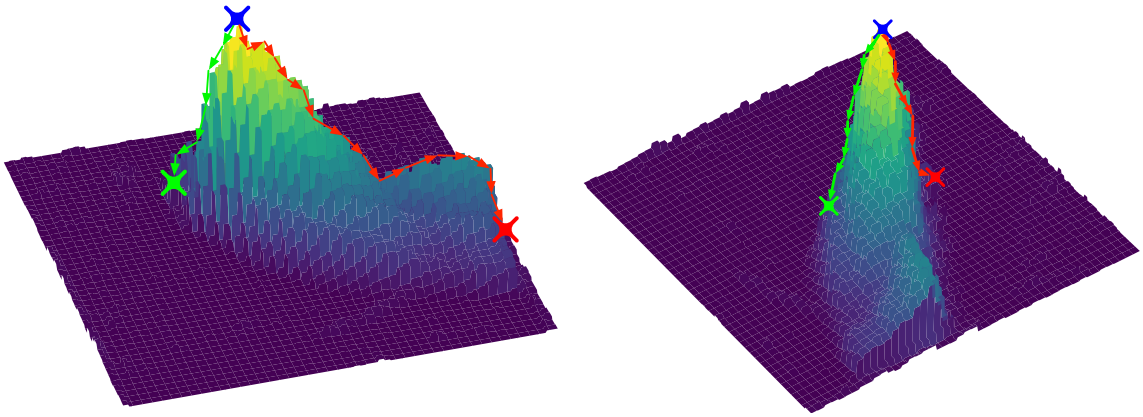
Here $z_{ji}$ is the topic assignment of $x_{ji}$, each word is associated with a multinomial distribution $\theta_{ji}$ according to its topic $z_{ji}$. We can potentially have an infinite number of atoms. However, only a finite number of atoms/topics are learned. After Gibbs sampling, there is a multinomial distribution for each topic over the vocabulary where each word in a document has a topic assignment. In other word, words in the same document may belong to *different* topics. Figure 13 visualizes those topics learned by HDP. The color indicates the direction shown in the color wheel on the right and the lightness indicates the value of the multinomial at the corresponding grid. The Figures show that HDP does an excellent job of summarizing the motions in the video, even without knowing any moving objects in the scene. For more details on the model and Gibbs sampling, please see (**?**).

## 3.3 Semantic knowledge learning

The results generated by HDP are several clusters of visual words, however, without any semantic meaning. This section describes some post-processing steps for extracting higher-level semantic knowledge of the camera scene.

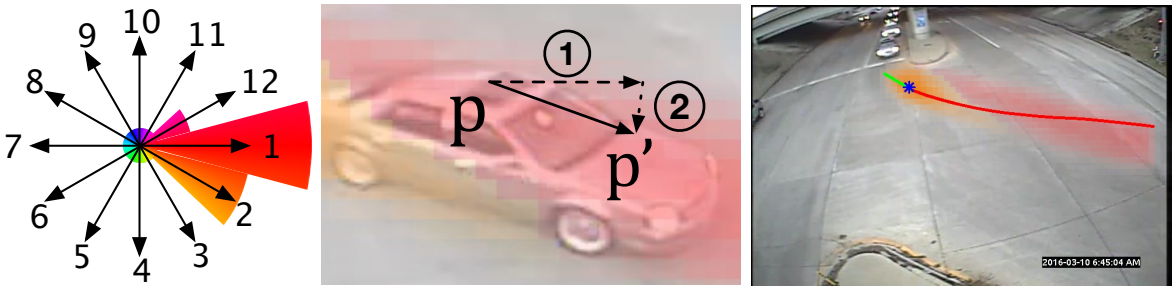
### 3.3.1 Robust ridge climbing

Just by looking at the topics in Figure 13, we can roughly infer how the vehicles move in the video. More importantly, the extent of the high-density grids gives useful insights into the size and moving region of the tracked objects. Let $\phi$ be a multinomial distribution of a topic learned by HDP. For any grid $i$, it has $D$ values, corresponding to $D$ evenly quantized directions, written as $\boldsymbol{\phi}_i = [\phi_i^1, \phi_i^2, \ldots, \phi_i^D]$. We define two terms: $d_i^* = \arg\max_d(\phi_i^d)$ is the *dominate direction* of grid $i$, which has the maximal value of distribution among all directions. We also define a terms summing over all the directions of each grid $\varphi_i = \sum_{d=1}^{D} \phi_i^d$. Figure 15a shows an example of the distribution of a particular topic on a grid, where the radius of each sector indicates the value of $\phi_i^d$. In this topic, the grid has the highest $\phi_i^d$ with $d = 1$, indicating that it is most likely to move along this direction. Figure 14 gives a 3D visualization of the first topic in Figure 13, where the height at each grid is $\varphi_i$. Overall they form a mountain-like surface. We aim to learn the shape and extent by extending the ridge climbing method in (**?**). The idea of this method is to start from a high-density grid, make one step iteratively until reaching the image boundary or the low-density area. The steps move along the desired direction and follow the shape of the topic distribution. For consistency, all the following examples are on the first topic in Figure 13.



(a) (b)

Figure 14: 3D visualization of the distribution of the first topic in Figure 13 and ridge climbing illustration on it: along the dominant topic direction (a) and its perpendicular direction (b). Blue cross indicates the starting point, red and green point indicates the end point.




(a) (b) (c)

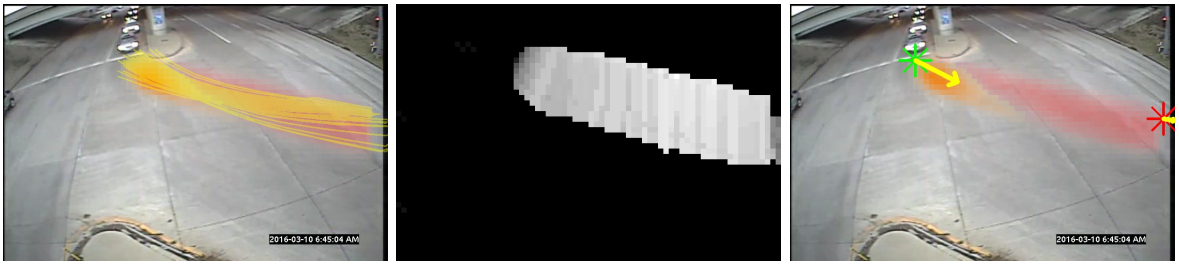
Figure 15: Ridge climbing illustration on the first topic in Figure 13: (a) is an example of the topic distribution on one grid $\mathfrak{i}$, where the radius of each circle sector indicates $\boldsymbol{\phi}_{\mathfrak{i}} = [\phi_{\mathfrak{i}}^1, \phi_{\mathfrak{i}}^2, \ldots, \phi_{\mathfrak{i}}^D]$. (b) shows the move-adjust step of each iteration. (c) is an extracted ridge starting from the highest density grid, where the blue star indicates the highest density grid, green and red line indicates ridges to the start/end point separately.

We propose a two-step movement in each iteration: first, we take a step, the step size is the normalized mean density on all the direction in its neighborhood; second, we adjust the step by moving toward higher density area. The above steps are illustrated in Figure 15b, represented in dash lines, the final step is in solid line. Mathematically, the two-step movement from $\mathbf{p}$ to $\mathbf{p^*}$ is represented as:

$$\mathbf{p}' = \mathbf{p} + \frac{\boldsymbol{\nu}}{||\boldsymbol{\nu}||_2}, \qquad \boldsymbol{\nu} = \frac{\sum_{i=1}^{n} f(\boldsymbol{\omega}) \cdot \boldsymbol{\phi}_i}{\sum_{i=1}^{n} \varphi_i}, \tag{3.3}$$

$$\mathbf{p}^* = \mathbf{p}' + \alpha \cdot \mathfrak{m}, \qquad \mathfrak{m} = \frac{\sum_{j=1}^{n} \varphi_j \times (\mathbf{p}_i - \mathbf{p}')}{\sum_{j=1}^{n} \varphi_j}. \tag{3.4}$$

We define a $2 \times D$ matrix $\boldsymbol{\omega} = [\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_D]$, each column $\boldsymbol{\omega}_d$ is the unit vector along the quantized direction $d$. $\boldsymbol{\nu}$ and $\mathfrak{m}$ correspond to the move and adjust step, separately. For the move step Equation 3.3, $f(\boldsymbol{\omega})$ defines movement of each $\boldsymbol{\omega}_d$ wrt. a desired moving direction, which is the same size with $\boldsymbol{\omega}$. Here we choose a neighborhood with $n$ grids around $\mathbf{p}$, and $\mathbf{p}_i$ is the coordinate of the $i$th grid in the neighborhood. Therefore, $f(\boldsymbol{\omega}) \cdot \boldsymbol{\phi}_i$ is the mean direction weighted by densities on each direction. However, the moving step may not follow the elongated shape of the topic due to the uncertainty introduced by the quantized direction, where an example is given in Figure 15b. To deal with it, we make an adjustment on the result of the move step Equation 3.4, once reaching a lower density area, $\mathfrak{m}$ pulls the point back to the higher-density area and make the final step (solid arrow in Figure 15b) better follow the shape of the topic, as illustrated in step 2 in Figure 15b. Note that in the adjust step, we compute a new neighborhood around the result of the move step. Intuitively, if we reach a grid



(a) (b) (c)

Figure 16: Scene learning results on the first topic in Figure 13: (a) shows multiple ridges learned from local maximal grid. (b) is the perpendicular width, where lighter intensity indicates a larger width. (c) shows extracted entry/exit hotspots indicates by the green and red star. And the yellow arrow shows their direction.

that has significantly different densities, we tend to take a smaller step in case the step size is too large to follow the density surface. On the contrary, if the first step reaches a grid with a roughly uniform distribution around, $\mathfrak{m}$ tends to be a zero vector and does not affect the final step. Due to the normalization term, $\boldsymbol{\nu}$ and $\mathfrak{m}$ has magnitude at most $1$ in (Equation 3.3) and (Equation 3.4). To make sure the adjust step does not cancel out the movement of step one, we normalize $\boldsymbol{\nu}$ into a unit vector. By setting $\alpha \in (0, 1)$, it has a fixed impact on the final step. We observe that $\alpha \in [0.1, 0.2]$ gives a reasonable result.

Figure 14a gives an visual illustration of the ridge climbing process along the dominant direction, and Figure 15c shows the resulting ridge. The process starts from the grid with the highest $\varphi$, indicated by the blue cross, along the dominant direction, in this case, direction 1. The red line is the path to the end point of the ridge, with $f(\boldsymbol{\omega}) = \boldsymbol{\omega}$; while the green line reaches the start point, obtained by setting $f(\boldsymbol{\omega}) = -\boldsymbol{\omega}$. So far the results are similar with those in (**?**).

### 3.3.2 Topic extent learning

Intuitively, the high-density grids indicate an active region when objects are more likely to move. A single ridge is not descriptive enough, especially for wide motion regions. We propose two variations of the above procedure. First, instead of starting from the grid with global maximal $\varphi$, we start with multiple grids with local maximal $\varphi$ in its neighborhood. The yellow lines in Figure 16a show the ridges extracted by the above procedure, where they roughly cover the high-density area. Second, we define a *perpendicular width* as the extent of the high density grids that perpendicular to the dominate direction. To obtain it, we climb the density surface along the direction perpendicular to the dominate direction $d_i^*$ of the current grid $i$, demonstrated in Figure 14b. $f(\boldsymbol{\omega})$ is obtained by rotating each unit vector $\boldsymbol{\omega}_d$ by 90 degrees clockwise ($f_1(\boldsymbol{\omega})$) and counterclockwise ($f_2(\boldsymbol{\omega})$). In other words,

$$f_1(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \boldsymbol{\omega}, \quad f_2(\boldsymbol{\omega}) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \boldsymbol{\omega}.$$

Figure 16b shows the learned perpendicular width of the same topic, where the lighter color indicates a larger width. The ridges and the perpendicular width together define the active region of a topic, giving size, location and direction of the belonging objects.

### 3.3.3 Entry/exit hotspots extraction

Compared with a single ridge, ridges start from multiple locations and better indicate the start and end of motions due to their larger coverage. The density of extracted ridges gives a nice interpretation of the area and it is unlikely for vehicles to go beyond the topic area. For

a set of ridges $\{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_m\}$, each of $\mathbf{l}_i$ is a sequence of grid coordinates $\mathbf{l}_i = \{\mathbf{p}_i^1, \mathbf{p}_i^2, \ldots, \mathbf{p}_i^{s_i}\}$, where $s_i$ is the length of ridge $\mathbf{l}_i$. We take the first and last point of each ridge, call them *candidates* of the entry/exit hotspot. Formally defined as

$$\mathbf{Z}_{\text{entry}} = \{\mathbf{p}_1^1, \mathbf{p}_2^1, \ldots, \mathbf{p}_m^1\}, \quad \mathbf{Z}_{\text{exit}} = \{\mathbf{p}_1^{s_1}, \mathbf{p}_1^{s_2}, \ldots, \mathbf{p}_1^{s_m}\}.$$

Similarly, each has its own direction:

$$\begin{aligned}\mathbf{V}_{\text{entry}} =& \{\mathbf{p}_1^2 - \mathbf{p}_1^1, \ldots, \mathbf{p}_m^2 - \mathbf{p}_m^1\},\\ \mathbf{V}_{\text{exit}} =& \{\mathbf{p}_1^{s_1} - \mathbf{p}_1^{s_1-1}, \ldots, \mathbf{p}_m^{s_m} - \mathbf{p}_m^{s_m-1}\}.\end{aligned}$$

Averaging over the coordinations and directions of candidates, we have a mean location and direction for each entry and exit hotspot:

$$\mathbf{p}_{\text{entry}} = \frac{\sum_{i=1}^m \mathbf{p}_i^1}{m}, \quad \boldsymbol{\nu}_{\text{entry}} = \frac{\sum_{i=1}^m \mathbf{p}_i^2 - \mathbf{p}_i^1}{m}, \tag{3.5}$$

$$\mathbf{p}_{\text{exit}} = \frac{\sum_{i=1}^m \mathbf{p}_i^{s_m}}{m}, \quad \boldsymbol{\nu}_{\text{exit}} = \frac{\sum_{i=1}^m \mathbf{p}_i^{s_m} - \mathbf{p}_i^{s_m-1}}{m}. \tag{3.6}$$

16c gives an visual result of the above process, where the green and red star corresponds to $\mathbf{p}_{\text{entry}}$ and $\mathbf{p}_{\text{exit}}$, the yellow arrow shoes their directions $\boldsymbol{\nu}_{\text{entry}}$ and $\boldsymbol{\nu}_{\text{exit}}$. Although entry/exit hotspots are also able to be obtained by fitting the start/end points of trajectories, they are less reliable than statistics learned from lower-level representation, since we are working on a tracking framework.

### 3.4 Semantic knowledge visualization

For topic model training, we make each video clip 90 frames (~ 3 sec); each frame is divided into $10 \times 10$ pixel grids. Different from $D = 4$ in (**?**; **?**), we make $D = 12$. This number considers motions more than horizontal and vertical, making subsequent post-processing more accurate. We extend a C++ implementation of HDP [1] and train the models on IDOT dataset (**?**), each video is around 5 minutes.

The post-processing results on HDP are high-level representation of the scene, which are subjective interpretation without numeric representation. Therefore, it is even not feasible to obtain ground truth for evaluation. Instead, we do extensive analysis and case study of the results on different videos. Complementing the partial results in previous sections, Figure 17, Figure 18, Figure 19 and Figure 20 provide complete results on several scenes, with the same representation as previous sections. Figure 18 summarizes a low-resolution video, where the four main motions are captured, while some turning motions are missed. This is likely because optical flow with small magnitude is filtered as error before it is fed to the topic model, or because such motions are rare or do not appear in the training video. For higher resolution videos, such as Figure 19 and Figure 20, optical flow results are more accurate and movement magnitudes are greater, when measured in pixels. Consequently, HDP can catch most visible motions, and even distinguish individual lanes. By visual inspection, the obtained movements,

---

[1] Chong Wang, David Blei: https://github.com/blei-lab/hdp.

Figure 17: Scene learning results at an intersection with one-way road. Entry (green) and exit (red) locations with direction (yellow arrows).

as well as the locations and directions of entry/exit points, are consistent with the subject scenes. We give results of more videos in the appendix.

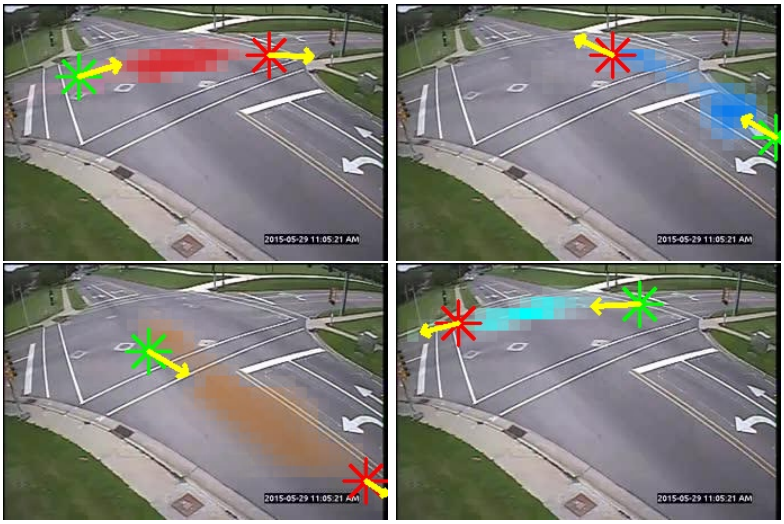


Figure 18: Scene learning results at an intersection with one-way road. Entry (green) and exit (red) locations with direction (yellow arrows).

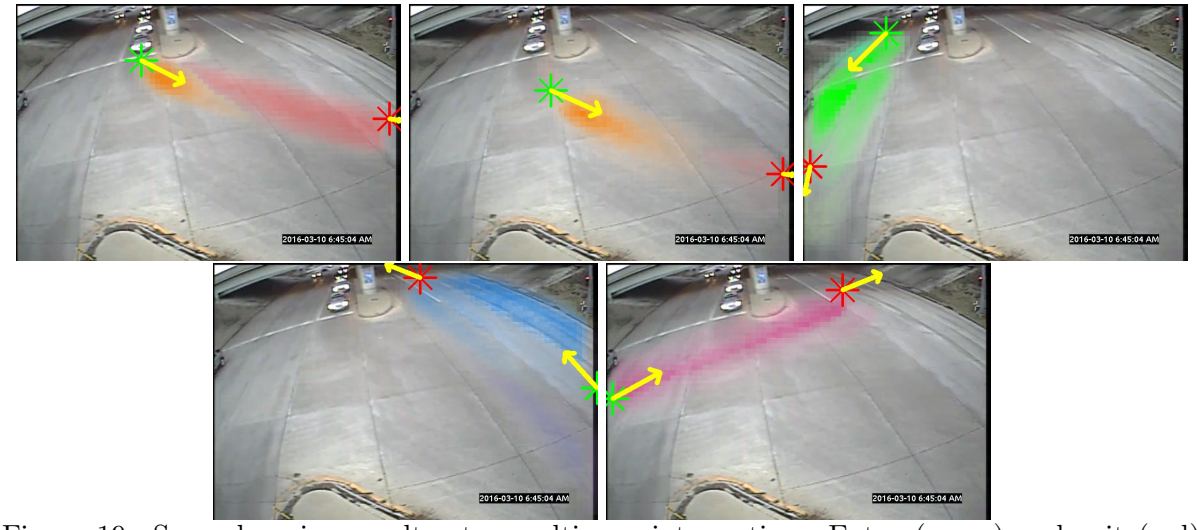


Figure 19: Scene learning results at a multi-way intersection. Entry (green) and exit (red) locations with direction (yellow arrows).

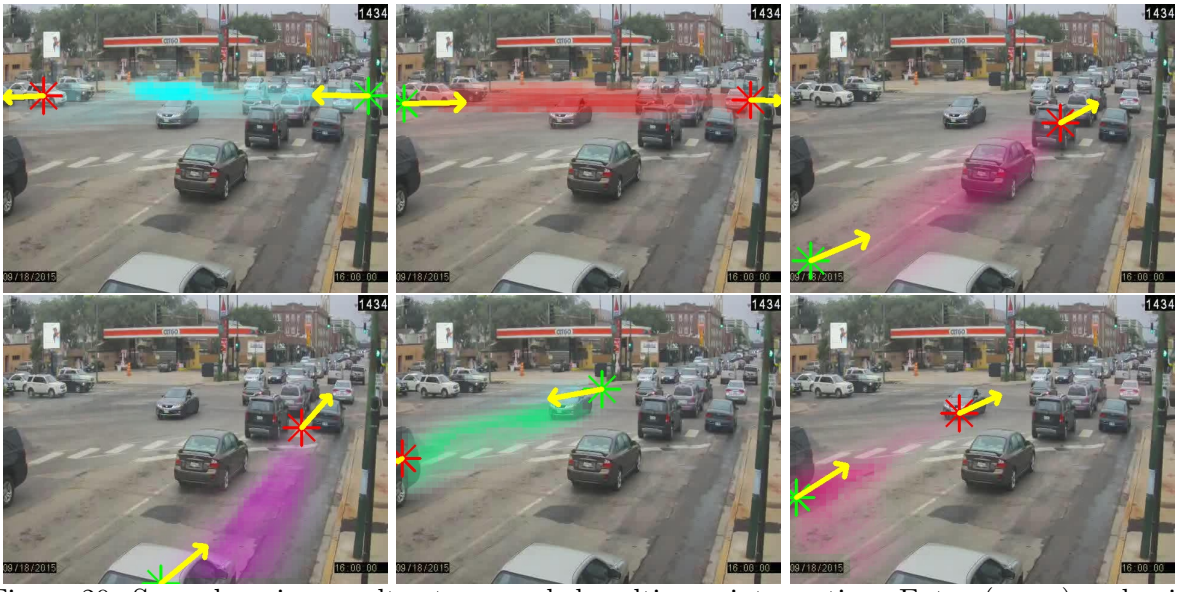


Figure 20: Scene learning results at a crowded multi-way intersection. Entry (green) and exit (red) with direction (yellow arrows).

## 3.5 Discussion

There remains some interesting aspects that can be potentially helpful for better scene understanding. First, it maybe be worth some effort of a better choice of hyper-parameters, such as grid size, number of quantized directions, video clip length, length training video, maximal sampling iterations. Grid size is related to the frame size. It is may not be necessary to have the same grid size for videos with different resolutions, since smaller grid size results in a larger vocabulary and a longer training time. For videos with only horizontal and vertical motions, the optical flow quantization can be sparser. The video clip length may also need to adjust depending on different scenarios: ideally, we expect a video clip to cover a complete lifetime of single motion; shorter clip may break a topic into different parts while a longer video

Figure 21: Failure case with simultaneous opposite directions. Entry (green) and exit (red) locations with direction (yellow arrows).

clip may generate a topic with mixed motions. Also, we have consider the traffic density of the scene to choose the size of the training data. A scene with little traffic needs a longer training video than a crowded one. However, even though the topic model is trained on a longer video, some rare motions may still be missing.

In addition, adding constraints among visual words might be necessary for some busy interactions. A "topic" is interpreted as a cluster containing "words" that always appear together, where the "words" are treated independently. For an intersection with bi-directional movements all the time, the movement on both direction is naturally clustered as a single "topic". However, under the context of a video, the "visual words" are spatially and temporarily connected. If a "visual topic" is defined with only a single motion, the "visual words" one the opposite direction should be mutually exclusive. Figure 21 and Figure 22 show two failure cases. One or more topics contain mixed motions, since those motion always happen simultaneously in the training data.

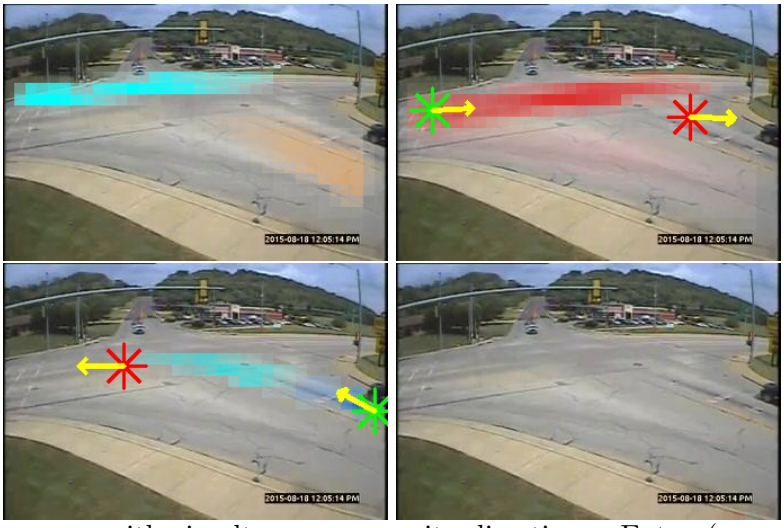


Figure 22: Failure case with simultaneous opposite directions. Entry (green) and exit (red) with direction (yellow arrows) at a crowded intersection.

However, if the training video is carefully chosen, this problem may be avoided. Figure 23 and Figure 24 are visualization of two models trained on a night video with sparse traffic. Figure 23 is trained on a 5-minutes video, and Figure 24 is trained on a 30-minutes video. Topics in Figure 24 have a better interpretation of the scene because the training video has sparse traffic at night. Since the video contain little traffic, there is less likely to have simultaneous movement in the opposite direction. A longer video has much more sufficient data for training.

However, it does not always work for tuning training videos. If a busy intersection always have the opposite movement at the same time, the learned topics tend to contain mixed movement. We considered introducing such exclusion into HDP via Hierarchical distance dependent Chinese Restaurant Process (ddCRP) (**?**). Due to the limited time, we do not completely finish this part.

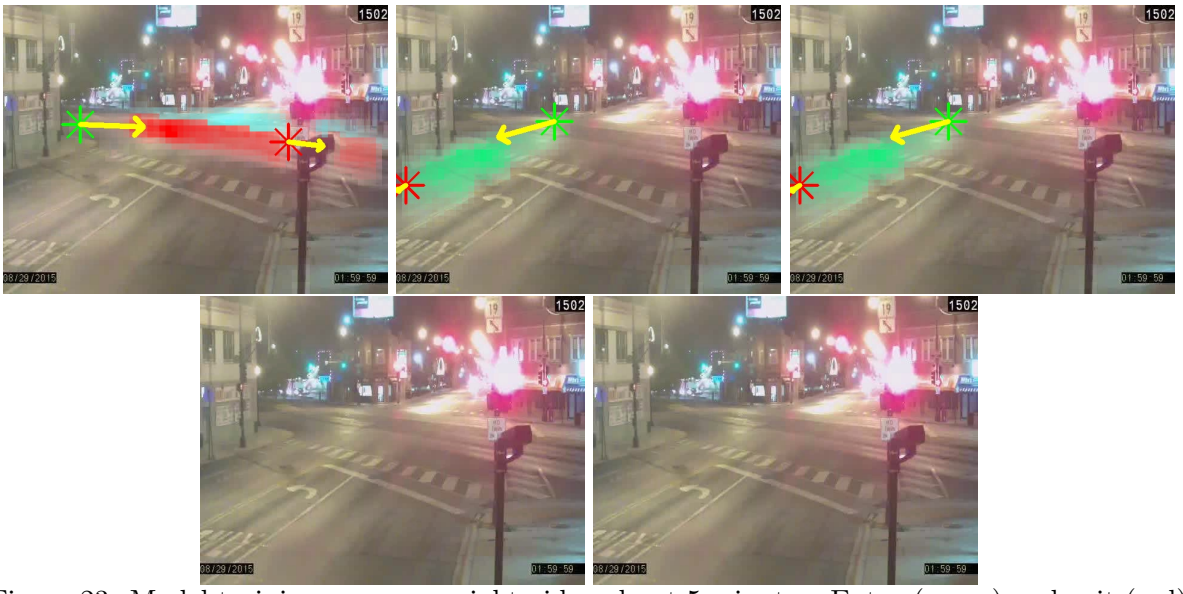


Figure 23: Model training on sparse night video about 5 minutes. Entry (green) and exit (red) locations with direction (yellow arrows).

Figure 24: Model training on sparse night video about 30 minutes. Entry (green) and exit (red) locations with direction (yellow arrows).

# CHAPTER 4

# SEMANTIC TRACKER

## 4.1 Introduction

Transportation videos are nowadays captured by millions of cameras installed at local and national highways and streets[1]. The analysis of such videos is critical for traffic volume monitoring, peak hour and congestion patterns discovery, tracing cars of criminals and stolen cars, highway toll management, among many others. For decades people have been trying to extract information from traffic videos. For example, one can hire a large number of human workers to watch and mark up any subregions, frames, or clips, that contain information of interest. This practice is labor-intensive and not scalable, as the workers need to pause from time to time and carefully analyze each object frame-wise. Even with high detection precision, recall can be low due to the bottleneck cheap human labor and processing bandwidth.

Recently, advanced solutions based on computer vision are proposed to mitigate the bottleneck but are still limited by human processing bandwidth and far away from fully automatic traffic analysis. First, while precise specifications of capturing camera or the starting and ending points play a large role in tracking accuracy (**?**; **?**; **?**; **?**; **?**; **?**), the videos are time-consuming to hand-labeled accurately with this level of details, while given a large number of cameras de-

[1]The Illinois Department of Transportation has a huge database of 24-hour traffic videos all across Illinois.

ployed nationwide, it is unwise to provide specifications for every camera. Second, any models trained for one scene cannot be deployed in other scenes, as the semantics of traffic scenes, such as entry/exit area (**?**; **?**) or road surface (**?**), can differ from scene to scene. Also, the same camera can undergo slight adjustments in their focus and angle, and a model trained for that camera can become inaccurate and incur further costly re-calibrations.

In this paper, we aim at fully automatic semantic knowledge extraction from transportation videos and significantly reduce human efforts in the vehicle tracking pipeline. A vehicle tracker can benefit from the more restricted motion patterns in traffic videos and we propose a video scene semantic learning method to discover atomic motions, the extent of active regions, entry/exit hotspots, etc.. We integrate the learned semantic knowledge into a tracker based on Kalman Filter, which can flexibly accommodate state-of-the-art object detectors like faster-RCNN (**?**). Experiments on 13 long traffic videos with diverse tracking scenes demonstrate the significantly improved tracking accuracy attained the proposed method, compared with other fully automatic approaches.

Our preliminary contributions are as follows:

- Explicitly addressed those critical yet ignored issues for practical transportation applications, such as automatic initialization and termination.
- Proposed a self-adaptive framework to where trackers are guided by the learned semantic scene knowledge, while in return the semantic knowledge is updated with tracking statistics.

- Provided a comprehensive evaluation of the proposed framework and demonstrate a significant improvement on object tracking.

## 4.2 Object motion assignment

The offline training phase generates models for the *visual topic*, where the inference on the videos from the same camera helps identify the topic assignment of a tracked object before any topic-specific constraints are applied. To do inference, first, we need to make the frame the same format as the *visual document*. We maintain a small sliding window of consecutive frames close to the current one, as well as its bag-of-words representation, same as in §3.2.1. The sliding window is across multiple frames; therefore, it takes the temporal consistency into account and is more robust than a single frame.

After inference, each visual word $x$ is assigned to the topic with the maximal probability. Then we reduce the results temporarily and spatially to the granularity of the grids on the frame. Assume $v_x$ is the index of $x$ in the vocabulary. Each word index may have multiple counts considering the bag-of-words representation being a histogram. We take the result of majority voting as the topic assignment $k_v$ of the word index $v$ as following:

$$k_v = \arg\max_{k} \sum_{x \in \{x : v_x = v\}} \mathbf{1}(k_x = k), \tag{4.1}$$

where $\mathbf{1}(\cdot)$ is the indicator function. On the other hand, each grid maps to D word index in the vocabulary, indicating D directions in Figure 12. Similarly, we can reduce the topic statistics to grids by majority voting.

$$k_g = \arg\max_k \sum_{v \in \{v: g_v = g\}} \mathbf{1}(k_v = k), \tag{4.2}$$

where $g_v$ is the grid that word index $v$ locates, and $k_g$ is the topic assignment of grid $g$ in that frame. When there comes a bounding box of interest, its topic assignment can be determined by the majority voting of topics of the covered grids. We omit the inference detail here; for details, we refer the reader to (**?**).

Here the underlying assumption of the reduction is that each grid belongs to only one movement, which is a reasonable case on a single frame. If the sliding window is small, all of the words on the same grid are likely to have the same topic assignment. Therefore, the majority voting rules out some unusual cases where some $v$ has little counts. Another benefit of these methods is that since the inference is done on words level, it captures the correct topic assignment even with mixed topics in the scene.

## 4.3 Tracking score

Based on the life span of a tracked object, there are three stages: initialization, tracking, and termination. Most literature focuses on tracking, assuming initialization and termination are properly handled. However, for practical use, each stage should be carefully dealt with

to ensure performance. For the first time, we introduce applying the scene-specific semantic knowledge to object tracking throughout the objects' lifetime.

### 4.3.1 Initialization

An object being "out of scene" actually refers to two scenarios: out of image boundary or too small to be recognized. Without loss of generation, this corresponds to both two cases when objects enter and exit the scene, as an example given in Figure 25. These two cases may be problematic for tracker initialization and termination if not handled correctly. An object may either be initialized when being partially visible or too small to be recognized. Different from the standard object tracking framework, tracking task in real-world use usually relies on external algorithms to find object candidates. Consequently, it is a critical step to the overall performance of any related task. In the following, we show how we apply the scene semantic knowledge into the tracking process and deal with the two scenarios above with a uniform affinity measurement.

In the computer vision field, for a rigid object, it is widely accepted to use a rectangle as the representation of an object: $\mathbf{r} = [\mathtt{x}, \mathtt{y}, w, \mathtt{h}]$, where $\mathbf{p}_\mathbf{r} = (\mathtt{x}, \mathtt{y})$ is the coordinates of the center point and $(w, \mathtt{h})$ is the width and height. A straight line through point $\mathbf{p}$, parallel to vector $\boldsymbol{\nu}$ is define as $\mathtt{L}(\mathbf{p}, \boldsymbol{\nu})$. Similarly, a ray from point $\mathbf{p}$ in the direction of vector $\boldsymbol{\nu}$ is define as $\mathtt{R}(\mathbf{p}, \boldsymbol{\nu})$. For a rectangle $\mathbf{r}$ with a center point $\mathbf{p}_\mathbf{r}$, given its moving direction $\boldsymbol{\nu}_\mathbf{r}$, we have the following terms for the rectangle, also visually illustrated in Figure 26a and Figure 26b:

- *Perpendicular width:* the distance between two intersection points of line $\mathtt{L}(\mathbf{p}_\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}\cdot\mathrm{perp}})$ and the rectangle $\mathbf{r}$, written as $W(\mathbf{r}, \boldsymbol{\nu}_\mathbf{r})$.

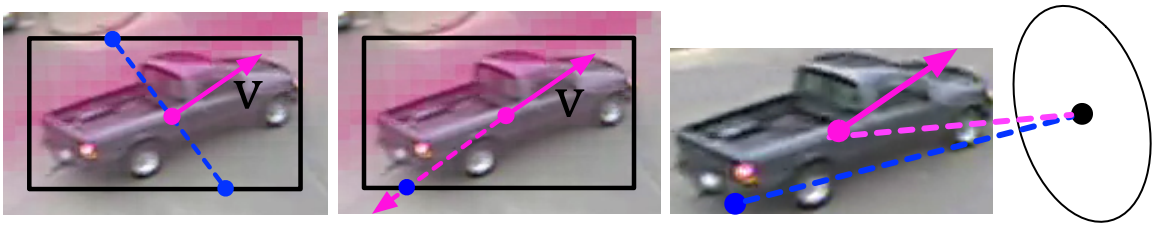
(a) (b)

Figure 25: Two scenarios of object enters and exits: (a): objects enter with a tiny size and move out of image boundary; (b): objects enter from the image boundary and exit with a tiny size in within the frame. Green and red star are the obtained entry/exit hotspots; yellow arrows shows their direction.

- *Last point:* the intersection point of ray $\mathsf{R}(\mathbf{p_r}, -\boldsymbol{\nu}_\mathbf{r})$ with the rectangle $\mathbf{r}$, written as $\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_\mathbf{r})$.

Here $\boldsymbol{\nu}_{\mathbf{r}\text{perp}}$ is the vector perpendicular to $\boldsymbol{\nu}_\mathbf{r}$, such that $\boldsymbol{\nu}_\mathbf{r} \cdot \boldsymbol{\nu}_{\mathbf{r}\text{perp}} = 0$. Since the object's aspect ratio may vary with the object type and view point of the camera, by defining the *perpendicular width*, both width and height are taken into account. As we show in the following, the definition of *last point* has a better interpretation of object's moving process, specifically for entering and exiting the scene.

In real-world applications, with no pre-determined object initialization available, applications usually rely on methods such as background subtraction and object detector, which usually tend to be error-prone. With the learned semantic knowledge, we only initialize objects that are around the entry area of the current available topics, with consistent direction and reasonable




(a) (b) (c)

Figure 26: All the solid pink arrows above indicate the moving direction $\boldsymbol{\nu}_{\mathbf{r}}$, the pink dot is the center of the rectangle, blue dots are the intersection points with the rectangle. (a): perpendicular width $W(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$ of the object's bounding box wrt. to direction $\boldsymbol{\nu}_{\mathrm{r}}$, shown as the blue dash line. b: last point $\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$ of the object's bounding box $\mathbf{r}$, where the dash arrow is $\mathsf{R}(\mathbf{r}, -\boldsymbol{\nu}_{\mathbf{r}})$. (c): distance between a hotspot with a rectangle's center point and last point $\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}})$, shown in pink and blue dash lines.

size. Consequently, noisy candidates with inconsistent motions or far away from the entry area are effectively eliminated.

To quantitatively represent the affinity of an object to a topic k's entry area, we define the following metric:

$$\begin{aligned} \mathsf{S}_{\text{entry}}(\mathbf{r}, \mathsf{k}) = \frac{1}{3} \times \Big[ & \Big(1 - \frac{\mathsf{D}(\mathsf{P}(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}}), \mathbf{p}_{\mathsf{k}}(\text{entry}))}{\mathsf{D}(\mathbf{p}_{\mathsf{k}}(\text{entry}), \mathbf{p}_{\mathsf{k}}(\text{exit}))}\Big) \\ & + \left(1 - \frac{|W(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}}) - W_{\mathsf{k}}(\mathbf{p}_{\mathbf{r}})|}{W(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}}) + W_{\mathsf{k}}(\mathbf{p}_{\mathbf{r}})}\right) \\ & + \cos\left(\boldsymbol{\nu}_{\mathbf{r}}, \boldsymbol{\nu}_{\mathsf{k}}(\text{entry})\right) \Big] \end{aligned} \tag{4.3}$$

Here $\mathsf{D}(\cdot)$ is the distance measurement of two arbitrary points, we simply use Euclidean distance here. $\mathbf{p}_{\mathsf{k}}(\text{entry})$ and $\mathbf{p}_{\mathsf{k}}(\text{exit})$ are the entry and exit point of topic $\mathsf{k}$, where the entry/exit points

are the Gaussian mean obtained in §3.3.3. $W_k(\mathbf{p_r})$ and $\boldsymbol{\nu}_k(\mathbf{p_r})$ are the perpendicular width and main direction of topic $k$ at location $\mathbf{p}_r$.

The first term in the square parenthesis is the distance of $P(\mathbf{r}, \boldsymbol{\nu_r})$ to the exit point, relative to the entry point. It produces a high value for bounding boxes around the entry area, taking the extent of the topic into account without placing any hard threshold. Note that instead of using the center of the rectangle $\mathbf{p}_r$, we use the last point of it. As illustrated in Figure 26c, when computing the distance of the object's bounding box and the hotspot, with the moving direction available, considering the last point along the direction measures the movement of the entire object. In other words, despite various object size, by ensuring the last point close to the entry/exit area, the entering/exiting process completes for the entire object. The second term is close to 1 if and only if $W(\mathbf{r}, \boldsymbol{\nu_r})$ and $W_k(\mathbf{p_r})$ are similar. This penalizes bounding boxes of size either too large or too small, making sure the qualified candidate has a reasonable size relative to the topic's perpendicular width.

To deal with objects close to the image boundary, we do not initialize a tracker until the last point of the bounding box $\mathbf{r}$ is apart from the image boundary at least a margin distance; for an object enters with a tiny size, initialization is not considered when the object movement $\|\boldsymbol{\nu_r}\|_2$ equals 0, since tiny size usually indicates tiny movement. Satisfying the above two cases, we initialize an object with a bounding box $\mathbf{r}$ once $S_{\text{entry}}(\mathbf{r}, k) > \sigma_1$ for a certain $k$.

### 4.3.2 Termination

When the tracked object manages to reach the exit area, it is quite likely to be well tracked, due to our checking scheme described later in §4.3.3. Therefore, we consider terminating a

tracker once it is close to its topic's exit area. Similarly, we define a distance measurement of a rectangle $\mathbf{r}$ with its exit area:

$$S_{\text{exit}}(\mathbf{r}, k) = 1 - \frac{D\left(P(\mathbf{r}, \boldsymbol{\nu}_{\mathbf{r}}), \mathbf{p}_k(\text{exit})\right)}{D\left(\mathbf{p}_k(\text{entry}), \mathbf{p}_k(\text{exit})\right)} \tag{4.4}$$

where we think the object passes through its exit area when $S_{\text{exit}}(\mathbf{r}, k) > \sigma_2$. However, in case the object is still able to be correctly tracked, we mark it and keep tracking until it gets lost, we call this phase *extended tracking*. The final trajectory is until the object gets lost. This is especially useful when the object exists with a tiny size. Due to the filtering of the topic model training process, tiny movements are filtered as noise. However, when the object can be tracked beyond the exit zone, it is better to keep a longer trajectory.

### 4.3.3 Tracking quality evaluation

An object's size, location, and motion are well constrained once we have a topic assignment of it. Once we initialize a tracker, we check the tracking quality on every frame. In general, we want objects to move smoothly and follow its visual topic. To quantitatively evaluate this, we define a smoothness term for an object's scale change:

$$S(\mathbf{r}_1, \mathbf{r}_2) = \frac{1}{2}\left[\left(1 - \frac{|w_1 - w_2|}{w_1 + w_2}\right) + \left(1 - \frac{|h_1 - h_2|}{h_1 + h_2}\right)\right], \tag{4.5}$$

here $w_i$ and $h_i$ are the width and height of rectangle $\mathbf{r}_i$, separately. $S(\mathbf{r}_1, \mathbf{r}_2)$ makes sure there is no abrupt scale change between two consecutive frames. Besides that, to evaluate how well the object is tracked, we have to consider regular tracking and extended tracking differently.

Before the object reaches its exit area, it is supposed to follow the corresponding visual topic tightly, possibly with occasional stops. So the tracking score is defined as:

$$S_{\text{track}}(\mathbf{r}, k, t) = \frac{1}{3}\Big[\Big(1 - \frac{|W(\mathbf{r}_t, \boldsymbol{\nu}_{\mathbf{r}_t}) - W_k(\mathbf{p}_{\mathbf{r}_t})|}{W(\mathbf{r}_t, \boldsymbol{\nu}_{\mathbf{r}_t}) + W_k(\mathbf{p}_{\mathbf{r}_t})}\Big) + S(\mathbf{r}_t, \mathbf{r}_{t-1}) + \cos(\boldsymbol{\nu}_{\mathbf{r}_t}, \boldsymbol{\nu}_k(\mathbf{p}_{\mathbf{r}_t}))\Big] \tag{4.6}$$

However, once the object passes through its exit area, it is expected to be terminated once it is not well tracked. Additionally, zero movements are not allowed. Since the object is not likely to pass through high-density grids in extended tracking phase, we compare the object's moving direction with the exit direction. A similar measurement is defined as:

$$S_{\text{extend}}(\mathbf{r}, k, t) = \frac{1}{2}\left[S(\mathbf{r}_t, \mathbf{r}_{t-1}) + \cos(\boldsymbol{\nu}_{\mathbf{r}}, \boldsymbol{\nu}_k(\text{exit}))\right] \tag{4.7}$$

In both measurement above, the notations are all similar with before, except that a subscript $t$ is added, indicating the corresponding variable at frame $t$. $\boldsymbol{\nu}_k(\mathbf{p}_{\mathbf{r}_t})$ and $\boldsymbol{\nu}_k(\text{exit})$ are the main direction of topic $k$ at $\mathbf{r}_t$'s center point $\mathbf{p}_{\mathbf{r}_t}$, and exit area. A valid tracking is defined as $S_{\text{track}}(\mathbf{r}, k, t) > \sigma_3$ or $S_{\text{extend}}(\mathbf{r}, k, t) > \sigma_4$, depending on whether it has passed through its exit area. Unqualified tracking is discarded as failures along the way. By checking the tracking quality in this way, it is guaranteed that objects manage to pass the exit area follows its topic well and has a smooth movement and scale change. Empirically, $\sigma_1 \sim \sigma_4$ between $0.7 \sim 0.8$ are a reasonable threshold.

## 4.4 Semantic tracker

### 4.4.1 Semantic knowledge update

With some successfully tracked objects, we may adaptively update the semantic scene. More specifically, we update the corresponding topics with well-tracked objects, which are those enter through the entry area, tightly follow its belonging topic, and successfully exit through exit areas. The scene is updated as follows: The first and last location, along with the direction of a high-quality trajectory is used to update the entry/exit candidates; the rectangle perpendicular width wrt. its topic is used to update the topic perpendicular within its extent, where the updated value remains the mean of all the updates. By such iterative updating, we have more representative scene knowledge regarded to object tracking. The entry/exit area may gradually shift to where most objects enter and exit, where the perpendicular width of each topic also better fits the object statistics. The entire process is described in Algorithm 4 to 6.

**Algorithm 4** Tracking with semantic knowledge.

**for** each frame $t$ **do**
  Topic model inference on the frame.
  Obtain foreground boxes $\mathbf{R}_{bg} = \{\mathbf{r}_{bg}\}$.
  Obtain detection boxes $\mathbf{R}_{det} = \{\mathbf{r}_{det}\}$.
  **for** each $i$ th object $O_i$ at $\mathbf{r}_{t-1}(i)$, $(i = 1, \ldots, N_t)$ **do**
    Get topic assignment $k_t(i)$.
    Find the most matched boxes $\mathbf{r}_{bg}(i)$ and $\mathbf{r}_{det}(i)$ with $\mathbf{r}_{t-1}(i)$.
    $\mathbf{R}_{bg} = \mathbf{R}_{bg} \setminus \mathbf{r}_{bg}(i), \mathbf{R}_{det} = \mathbf{R}_{det} \setminus \mathbf{r}_{det}(i)$.
    Compute mean velocity $\boldsymbol{\nu}(i)$ within $\mathbf{r}_{t-1}(i)$ from the optical flow.
    Make measurement for tracking update $\boldsymbol{z}_t(i) = [\mathbf{r}_{bg}(i), \mathbf{r}_{det}(i), \boldsymbol{\nu}(i)]$.
    Set measurement covariance error $R$.
    Update object tracker with $\boldsymbol{z}_t(i)$ and $R$, get result $\mathbf{r}_t(i)$.
    CheckExitSemantic$(O_i, \mathbf{r}_t(i), k_t(i), t)$
  **for** Each remaining box candidate $r \in \{\mathbf{R}_{bg} \bigcup \mathbf{R}_{det}\}$ **do**
    CheckEntrySemantic$(\mathbf{r})$.

**Algorithm 5** CheckExitSemantic$(O, \mathbf{r}, k, t)$

**if** $O$ is in extended tracking phase **then**
  **if** $S_{extend}(\mathbf{r}, k, t) <= \sigma_4$ **then**
    Exit object.
**else if** $S_{track}(\mathbf{r}, k, t) < \sigma_3$ **then**
  Discard object.
**if** $S_{exit}(\mathbf{r}, k) > \sigma_2$ **then**
  Enter extended tracking phase.

**Algorithm 6** CheckEntrySemantic$(\mathbf{r})$

Get the topic $k$ for $\mathbf{r}$ at this frame.
**if** $S_{entry}(\mathbf{r}, k) > \sigma_1$ **then**
  Initialize object.

## 4.5 Evaluation

### 4.5.1 Tracking accuracy

The extracted movements and entry/exit locations are hard to evaluate quantitatively, due to a lack of both ground-truth and accepted difference metrics. Instead, we extend a vehicle tracker with initialization- and update filters based on movements and entry/exit locations extracted from the scene, as described in the previous section.

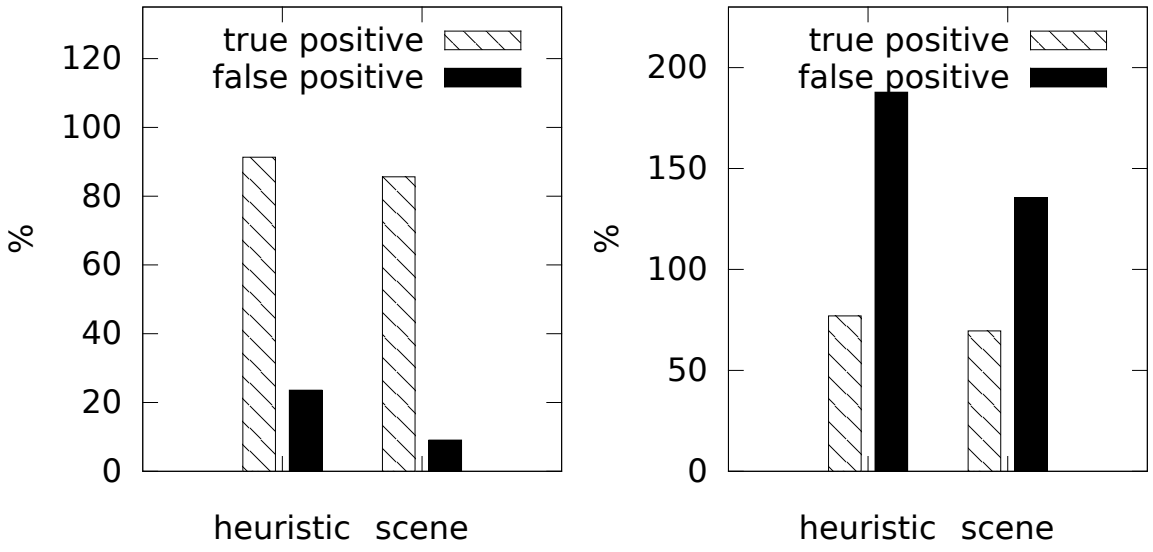
Figure 27 shows our main result. Here, we compare the tracker from (**?**), labeled *heuristic*, against our version extended with a scene learning filter, labeled *scene*. Although the *heuristic* tracker relies on several manually tuned parameters, this is one of the few benchmarks that quantitatively evaluate tracking including initialization and termination, which is critical for any realistic vehicle tracking application.

Here, true and false positives are based on matching tracked vehicles with ground truth trajectories. We use the *overlap* of two rectangles, defined by the intersection over union for a rectangle box $\mathbf{r}$ and a ground truth $\mathbf{r}_0$,

$$\text{Overlap}(\mathbf{r}, \mathbf{r}_0) = \frac{\text{A}(\mathbf{r} \cap \mathbf{r}_0)}{\text{A}(\mathbf{r} \cup \mathbf{r}_0)},$$

averaged over the frames in which the ground truth trajectory or the tracking result occurs.

As in (**?**), we match each ground truth trajectory to the tracking result with the greatest overlap and use overlap $> 0.3$ to indicate a true positive. Any ground truth trajectories without a true positive match are considered false negatives, and any tracking result without a matched



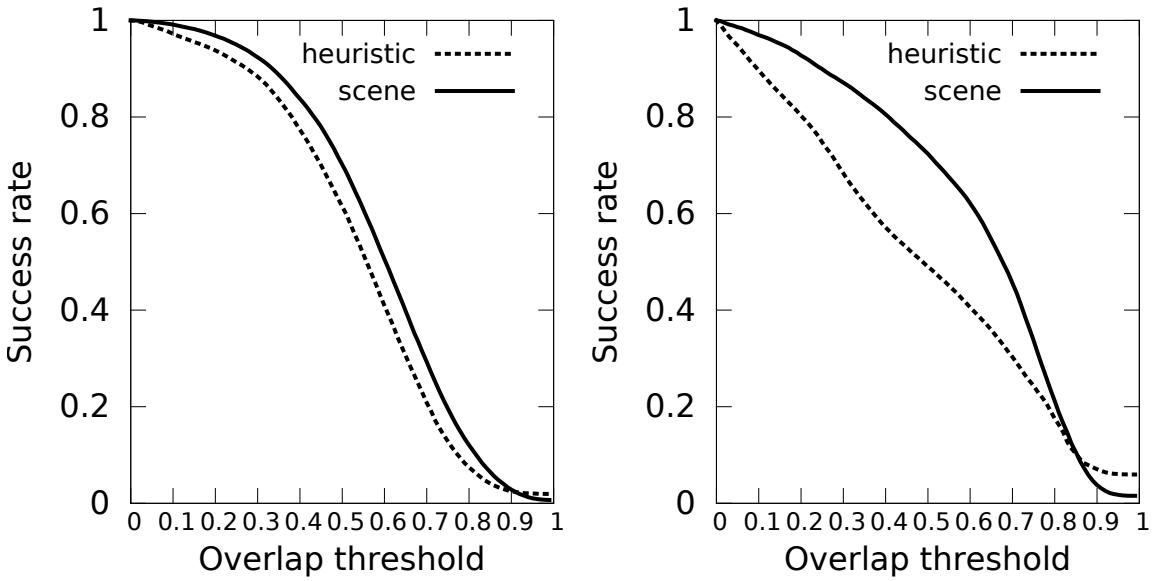

(a) Low resolution videos. (b) High resolution videos.

Figure 27: The number of true positive and false positve trackers. The black lines mark the ground truth, the striped and black bar are the true positive and false positive, separately.

ground-truth trajectory with overlap $> 0.3$ is considered a false positive. We find that our scene learning filter dramatically reduces false positives while keeping true positives essentially constant.

A large number of false positives remain for the high-resolution videos. Due to the complex vehicle interactions in high the resolution video (see Figure 20), even with the semantic knowledge, the naïve Kalman Filter may easily lose track, resulting in an increased false positive count.

Figure 28 shows the success plot of the two trackers for both simple and complex videos. Here, the *success rate* is defined as the fraction ground-truth trajectories that have a minimum




(a) Low resolution videos. (b) High resolution videos.

Figure 28: Success rate plot. A larger area under the curve shows the better performance.

overlap with its matched tracking output. Thus, the success rate measures how closely the matched trajectories actually match the ground truth. The larger area under the curves is, the better the tracking is. We see significant improvement in tracking accuracy with the help of scene learning, especially on the more complex videos.

In conclusion, we find that filtering tracker initialization and update using scene learning can significantly improve tracker performance, both in terms of precision and accuracy.

# CHAPTER 5

# SCENE-SPECIFIC MOTION MODEL

## 5.1 Introduction

The underlying assumption in §2.3 is that vehicles follow the constant acceleration model in a short period, like in the physical world. However, everything in the scene experiences distortion due to projection in most traffic videos. Therefore, vehicles' movement does not follow the linear model in Equation 2.1, and a linear state model is insufficient for the Kalman Filter to capture the movement pattern of the vehicles in the scene.

When good observations are available consistently, Kalman filter can follow the tracked vehicle, even though the linear model cannot accurately reflect the size and velocity change. With missing observation or occlusion, the tracker is expected to predict with its internal model in the *extrapolation* mode. However, with the linear model broken, the tracker can only generate reasonable results for a short period.

To tackle this problem, we proposed a Unscented Kalman Filter (UKF) tracker, which learns a non-linear model from the history trajectories by GP. The semantic knowledge is the prerequisite of the non-linear model, which ensures the trajectories of high quality. Under the constraints of semantic learning, every vehicle that survives to the exit hotspot is consistent with its motion topic in terms of entry/exit location, size, and velocity.

## 5.2 Gaussian Process

Gaussian Process is a non-parametric model that learns a distribution over functions $f \sim \mathcal{GP}$. Given a dataset $\mathcal{D} = \left\{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^d, y \in \mathbb{R}, i = 1, \ldots, N\right\}$, where $y_i = f(\mathbf{x}_i)$. A GP assumes a prior distribution that $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)$ jointly follows a Gaussian distribution $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mu, \mathbf{K})$, where $\mathbf{K}_{ij}$ is defined by a kernel function $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. When $N_*$ new data points come, by the definition of GP, the joint distribution forms another Gaussian distribution

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K^T}_* & \mathbf{K}_{**} \end{pmatrix}\right), \tag{5.1}$$

where $\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$ is $N \times N$, $\mathbf{K}_* = \kappa(\mathbf{X}, \mathbf{X}_*)$ is $N \times N*$, and $\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*)$ is $N_* \times N_*$.

By the conditioning Gaussian (**?**), the posterior is $p(\mathbf{f}_*|\mathbf{X}, \mathbf{X}_*, \mathbf{y}) = \mathcal{N}(\mathbf{f}_*|\mu_*, \Sigma_*)$, where

$$\mu_* = \mu(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1}(\mathbf{y} - \mu(\mathbf{X})) \tag{5.2}$$

$$\Sigma_* = \mathbf{K}_{**} - \mathbf{K}_*^{\mathbf{T}} \mathbf{K}^{-\mathbf{1}} \mathbf{K}_* \tag{5.3}$$

For the regression task, we can use $\mu_*$ as the prediction. Usually we assume $\mu(\mathbf{X}) = 0, \mu(\mathbf{X})_* = 0$. The Radial Basis Function (RBF) kernel is used here

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{\mathbf{x}_{id} - \mathbf{x}_{jd}}{l_d}\right)^2\right). \tag{5.4}$$

We use a different length scale for each dimension of $\mathbf{x}$. $\sigma_f$ and $l_d$ are hyper parameters of the GP learned through a optimization solver.

### 5.2.1 Multiple output Gaussian Process

§5.2 introduces the basic definition of GP, where $y$ is a real value. However, in our case, we learn a scene-specific motion model, which is a mapping of vehicle states between two consecutive frames. The history trajectories provide such mapping for training and capture the non-linear movement. Different from §2.3, we use a six-dimension input $\mathbf{x} = [x, y, w, h, x', y']$, which represents the location, size and velocity of vehicle. The output $\mathbf{y}$ has to have the same format with input $x$, as it becomes the input for the next frame. Therefore, the single-output GP extends to multiple output, where $\mathbf{y} \in \mathbb{R}^6$. The covariance matrix is shared among all dimensions, equivalent to several single-output GP with the same covariance matrix.

### 5.2.2 Online processing for streaming data

One of the major challenges for GP is the covariance matrix $\mathbf{K}$. $\mathbf{K}$ grows with the size of training data, while the cost of computing the inverse $\mathbf{K}^{-1}$ is $O(N^3)$. On the other hand, we have streaming training data as the tracking continues. $\mathbf{K}$ need to be updated every time new data comes and every iteration of the optimization for the hyper-parameter. $\mathbf{K}$ can become enormously large if we store all the data. Figure 29 shows the relationship between the trajectory count and the training data count. The training data count grows linearly with the training trajectory. With about 32 trajectories, the matrix $\mathbf{K}$ size reaches 2000.

Therefore, we have to maintain the $\mathbf{K}$ to a reasonable size and avoid computing $\mathbf{K}^{-1}$ too frequently to balance the computation overhead. Some filtering and forgetting policies are applied here. First, we only add the data that has enough change over the previously added data point. Here $L1$ distance is used to measure the difference. Similar trajectory record

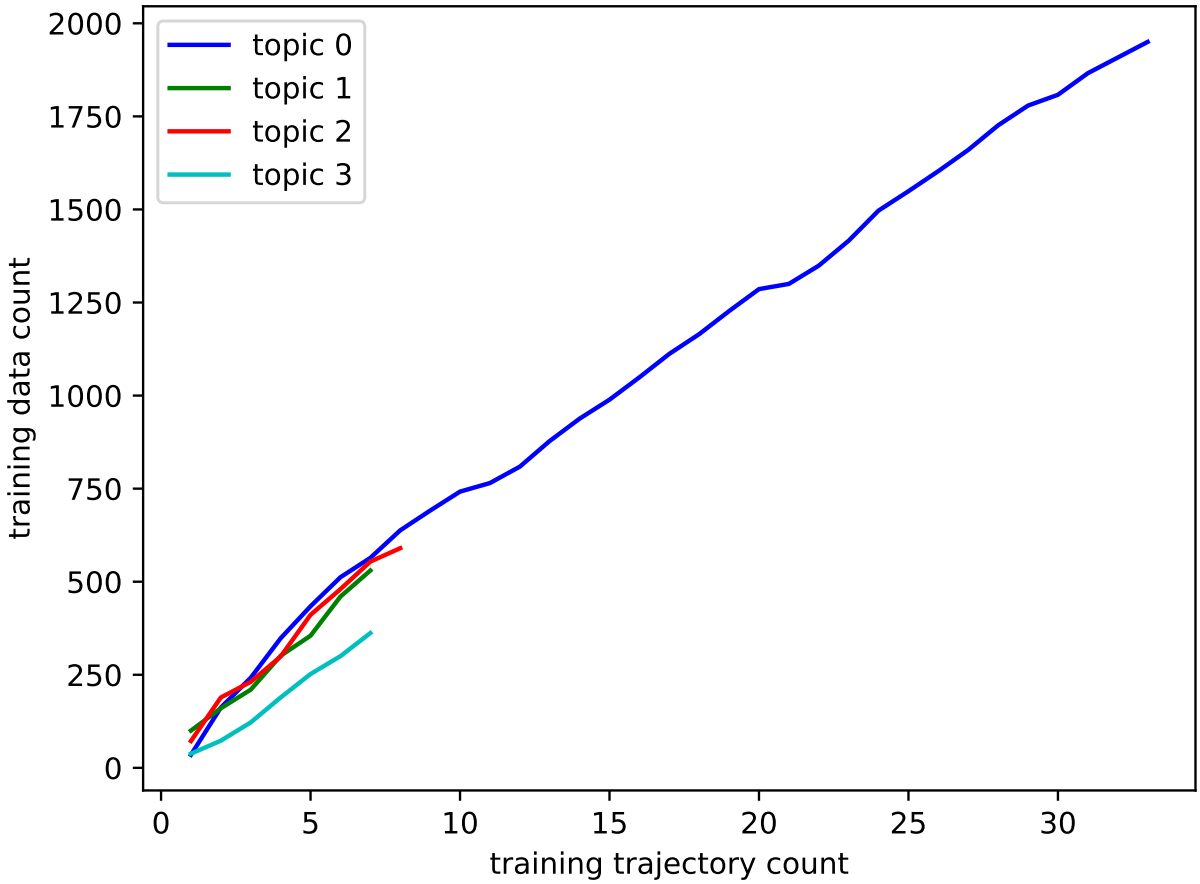



Figure 29: The training data count grows linearly with the training trajectory count.

between adjacent frames is often seen when the tracked object remains static in the scene or is too small to have a large change of size or velocity. Second, we run hyper-parameter optimization upon the first batch of training data arrive; later we run the optimization with a longer interval. Finally, after the training data has reached the maximal number, every time a new data point becomes available, it is added with a possibility of $p = 0.7$ and a random old data point is removed. Moreover, once the $\mathbf{K}^{-1}$ is computed, the prediction only involves simple matrix multiplication. We implement the algorithm on GPU to have fast matrix operation.

As the tracking continues, the covariance matrix $\mathbf{K}$ is maintained to a fixed size. We also keep a separate thread for computing $\mathbf{K}^{-1}$ and hyper-parameter optimization. The thread runs in the background without interrupting the tracker. The above methods ensure the tracker still run in real time.

## 5.3 Unscented Kalman Filter

In Kalman Filter, there are two important models,

- Internal state model: $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{w}_{t-1})$
- Observation model: $\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{v}_t)$

where $\mathbf{x}_t$ and $\mathbf{z}_t$ are the internal state and the observation, $\mathbf{w}_t$ and $\mathbf{v}_t$ are the process and measurement noise at time $t$, and both $f(\cdot)$ and $h(\cdot)$ are assumed linear. However, in many cases, at least one of the linear models does not hold. Therefore, methods for Kalman Filter with nonlinear model are proposed, including Extended Kalman Filter (EKF) (**?**) and UKF (**?**). EKF uses Taylor series expansion to approximate the nonlinearity, may introduce errors for the true mean and covariance of the state variables after the non-linear transformation. Additionally, in our

case, there is no close-formed nonlinear motion model under camera projection. Consequently, we use UKF, where its nonlinear state model $f(\cdot)$ is learned by the non-parametric model GP.

The key of UKF is unscented transformation (UT), which calculates the statistics of random variables under a nonlinear transformation, accurately capturing the mean and the covariance to the 3rd order Taylor series expansion for Gaussian inputs. For a random variable $\mathbf{x} \in \mathbb{R}^d$ with mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P}_\mathbf{x}$, to calculate the statistics of $\mathbf{z}$ under a nonlinear mapping $\mathbf{z} = g(\mathbf{x})$, first a set of sigma points are built around the $\bar{\mathbf{x}}$ (**?**),

$$\begin{aligned}
\mathcal{X}_0 &= \bar{\mathbf{x}} \\
\mathcal{X}_i &= \bar{\mathbf{x}} + (\sqrt{(d+\lambda)\mathbf{P}_\mathbf{x}})_i, \qquad i = 1, \ldots, d \\
\mathcal{X}_i &= \bar{\mathbf{x}} - (\sqrt{(d+\lambda)\mathbf{P}_\mathbf{x}})_i, \qquad i = d+1, \ldots, 2d \\
W_0^{(m)} &= \lambda/(d+\lambda) \\
W_0^{(c)} &= \lambda/(d+\lambda) + (1-\alpha^2+\beta) \\
W_i^{(m)} &= W_i^{(c)} = 1/\{2(d+\lambda)\} \qquad i = 1, \ldots, 2d
\end{aligned} \tag{5.5}$$

where $\alpha$ controls the spread of the sigma points around $\bar{\mathbf{x}}$, $\lambda = \alpha^2(d+\kappa) - d$ and $\kappa$ are the scaling parameters, $\beta$ describe the prior knowledge for the distribution of $\mathbf{x}$. $\alpha$ is usually set to a small value (*e.g.*$1e-3$), $\kappa$ is usually set to 0 and $\beta$ is optimally set to 2 for Gaussian distribution. $(\sqrt{(d+\lambda)\mathbf{P}_\mathbf{x}})_i$ is the $i$th row of the matrix square root. Each sigma point is propagated through the nonlinear function $g(\cdot)$,

$$\mathcal{Z}_i = g(\mathcal{X}_i) \qquad i = 0, 1, \ldots, 2d,$$

The mean and covariance for $\mathbf{z}$ are approximated by the weighted sample mean and covariance of the posterior sigma points,

$$\bar{\mathbf{z}} \approx \sum_{i=0}^{2d} W_i^{(m)} \mathcal{Z}_i \tag{5.6}$$

$$\mathbf{P}_{\mathbf{z}} \approx \sum_{i=0}^{2d} W_i^{(c)} (\mathcal{Z}_i - \bar{\mathbf{z}})(\mathcal{Z}_i - \bar{\mathbf{z}})^T \tag{5.7}$$

Therefore, with the non-linear state and measurement model, UKF naturally extends the unscented transformation by applying it alternately between the predict and correct step.

## 5.4 GP-UKF tracker

### 5.4.1 Tracking

In the tracking setting, the state model is a nonlinear GP model while the measurement model is still linear.

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) = \mathrm{GP}(\mathbf{x_{t-1}}, \mathbf{X}) \tag{5.8}$$

$$\mathbf{z}_t = h(\mathbf{x}_t) = H\mathbf{x}_t, \tag{5.9}$$

Similar with §2.3, the observation is still the bounding boxes generated by background subtraction model and vehicle detector, and filtered pixel movement from optical flow:

$$\mathbf{z} = [x^{bg}, y^{bg}, w^{bg}, h^{bg}, x^{det}, y^{det}, w^{det}, h^{det}, v_x, v_y].$$

Similarly, the measurement model $H$ is a $6 \times 10$ matrix with $1$ at $(0,0)$, $(1,1)$, $(2,2)$, $(3,3)$, $(0,4)$, $(1,5)$, $(2,6)$, $(3,7)$, $(4,8)$, $(5,9)$.

A complete step of the GP-UKF tracker is described in Algorithm 7, which can be adapted to the semantic tracker by substitution of the tracking step in Algorithm 4.

**Algorithm 7** GP-UKF.

**Require:** State transition mapping $\{\mathbf{x}_{t-1}, \mathbf{x}_t\}$ from history data.

1: Start with $\hat{\mathbf{x}}_0 = \mathbf{x}_0$.

2: Calculate the sigma points: $\mathcal{X}_{t-1} = \left[\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{x}}_{t-1} \pm \sqrt{(d+\lambda)P_{t-1}}\right]$.

3: Predict:

$$\hat{\mathcal{X}}_{i,t} = \mathrm{GP}_\mu(\mathcal{X}_{i,t-1}, \mathbf{X}), \quad i = 0, \ldots, 2d$$

$$\hat{\mathbf{x}}_t^- = \sum_{i=0}^{2d} W_i^{(m)} \hat{\mathcal{X}}_{i,t}$$

4:

$$P_t^- = \sum_{i=0}^{2d} W_i^{(c)} [\hat{\mathcal{X}}_{i,t} - \hat{\mathbf{x}}_t^-][\hat{\mathcal{X}}_{i,t} - \hat{\mathbf{x}}_t^-]^T$$

$$\hat{\mathcal{Z}}_t = \left[H\hat{\mathcal{X}}_{i,t}\right], \quad i = 0, \ldots, 2d$$

$$\hat{\mathbf{z}}_t^- = \sum_{i=0}^{2d} W_i^{(m)} \hat{\mathcal{Z}}_{i,t}$$

5: Get measurement covariance of $\mathbf{z}_t$: $R_t = \mathrm{GP}_\sigma(\mathbf{z}_t, \mathbf{X})$.

6: Correct with measurement:

$$P_{\hat{\mathbf{z}}_t, \hat{\mathbf{z}}_t} = \sum_{i=0}^{2d} W_i^{(c)} [\hat{\mathcal{Z}}_{i,t} - \hat{\mathbf{z}}_t^-][\hat{\mathcal{Z}}_{i,t} - \hat{\mathbf{z}}_t^-]^T + R_t$$

7:

$$P_{\hat{\mathbf{x}}_t, \hat{\mathbf{z}}_t} = \sum_{i=0}^{2d} W_i^{(c)} [\hat{\mathcal{X}}_{i,t} - \hat{\mathbf{x}}_t^-][\hat{\mathcal{Z}}_{i,t} - \hat{\mathbf{z}}_t^-]^T$$

$$K = P_{\hat{\mathbf{x}}_t, \hat{\mathbf{z}}_t} P_{\hat{\mathbf{z}}_t, \hat{\mathbf{z}}_t}^{-1}$$

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + K(\mathbf{z}_t - \hat{\mathbf{z}}_t^-)$$

$$P_t = P_t^- - K P_{\hat{\mathbf{z}}_t, \hat{\mathbf{z}}_t} K^T$$

### 5.4.2 Cold start

Before any trajectory is available, we track vehicles automatically with the Kalman Filter with the linear model in 2.3. With some accumulated trajectories, the GP can predict for objects with similar size and location. However, the prediction may fail at some unseen data; therefore, it is necessary to assess the quality of the prediction. We use the covariance of the data point in Equation 5.3, since it measures the certainty of the prediction. Upon initialization of an object, GP-UKF is used only when the obtained covariance is small ($< 0.1$); otherwise, the linear Kalman Filter is used.

## 5.5 Evaluation

### 5.5.1 Comparison with linear Kalman Filter

We train two sets of GPs on the trajectories from both the heuristic and semantic trackers, one for each topic. First, we split the ground truth trajectories into two sets for training and testing. Then we match both sets of trajectories with ground truth, as described in §2.5. For both trajectories that matched to the same ground truth, the training data are from the bounding boxes on the frames appear in both trajectories. Therefore, two sets of GPs have the same amount of training data. The GPs are trained incrementally, with a randomly sampled trajectory at a time. This simulates the process of our tracker setting. Apart from GPs trained on different datasets, we also compare with the Klaman filter with a linear dynamic model. A Kalman filter is built with the ground truth as measurement. It is the best that a linear Kalman filter can achieve since the measurement given is the perfect ground truth. Figure 30 shows change of the mean Intersection over Union (IoU) with the trajectory count. The GPs

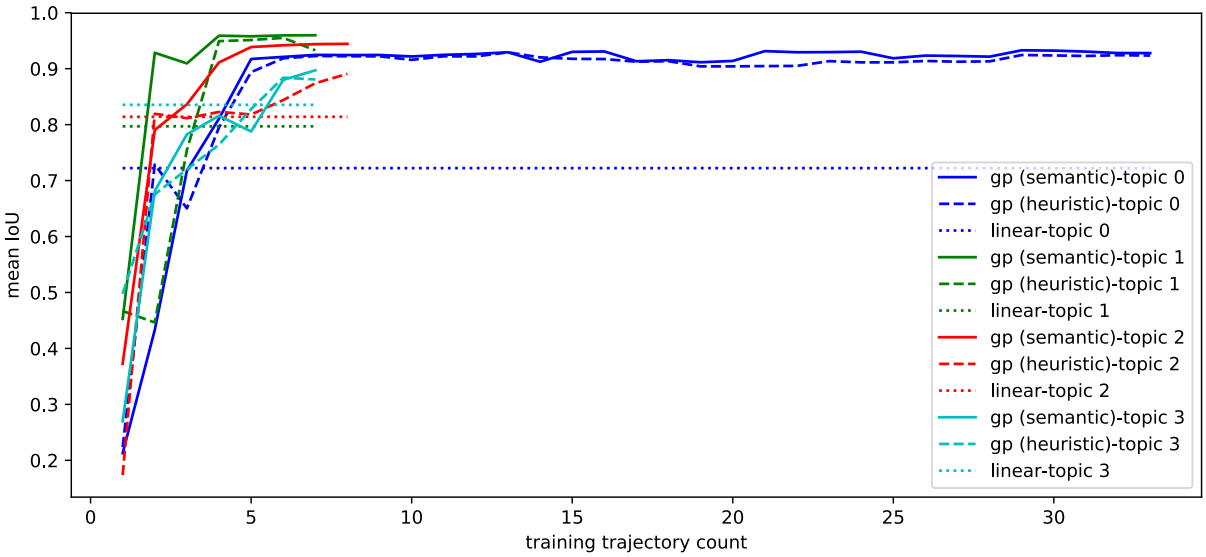



Figure 30: The mean IoU of GPs trained on the trajectories from the heuristic and semantic tracker, and the linear Kalman filter. Each topic has two GPs and a linear Kalman filter.

start with one training trajectories, where they make a poor prediction on the testing data. However, with a few more trajectories, the GPs have much better performance. The dash and solid lines are GPs from the heuristic and semantic trackers, separately. The GPs trained on the trajectories of the semantic tracker generally have better prediction over the ones from the heuristic tracker. By comparing with the linear Kalman filter, the GPs significantly perform better. The GPs, for topic 0 have the most significant gap with the baseline Kalman filter. From Figure 17, the vehicles from topic 0 moving with a increasing size and experience the nonlinearity the most; GP captures such non-linear change.

### 5.5.2 Case study

Figure 31, Figure 32, Figure 33 shows the comparison of the Kalman filter with linear and non-linear model, which we call KF (left) and GP-UKF (right). Under projection, vehicles moving towards the right of the frame has an increasing rate of size and velocity. In Figure 31, tracking just started. GP-UKF tracker does not have any training data, which works similarly with the KF tracker. In Figure 32, GP-UKF has accumulated 1-2 trajectories, showing a slightly better coverage of the actual vehicle. Finally, when GP-UKF has 5-8 trajectories as the training data, GP-UFK adapts to the scale and velocity significantly better than linear KF tracker in Figure 33.


(a) Kalman filter with Linear model. (b) Kalman filter with non-linear model.

Figure 31: Tracking screenshots at frame 854, no trajectory for Gaussian Process.


(a) Kalman filter with linear model. (b) Kalman filter with non-linear model.

Figure 32: Tracking screenshots at frame 914, 1-2 trajectories for Gaussian Process.


(a) Kalman filter with linear model. (b) Kalman filter with non-linear model.

Figure 33: Tracking screenshots at frame 989, 5-8 trajectories for Gaussian Process.

## 5.6 Discussion

Learning scene-specific non-linear movement models can be an interesting direction with huge potential. Although GP is a powerful formulation of this problem, there remains practical challenges for large streaming data processing. In GP area, algorithms are tested on a simple and small dataset, which is usually several hundred data points with only one dimension. Although there a few papers working on online GP, they are studying the incremental update of the covariance matrix, which avoids redundant computation of history data. However, the matrix cannot be infinitely large. Our filtering and forgetting strategy is heuristic methods for processing large streaming dataset. It is interesting to see some methods for data forgetting with a rigorous proof of improvement, rather than random forgetting.

# CHAPTER 6

# VEHICLE COUNTING SYSTEM

## 6.1 Introduction

People in civil engineering are devoted to building a better social environment. As one of its sub-discipline, traffic engineering focuses on efficient traffic flow. It aims to achieve safe and efficient movement of people and goods on existing transportation infrastructures. For example, by properly designing road geometry, the average travel time may be shortened; by modification of the traffic lights and signs, the crash rate at a specific location can decrease significantly.

To make the right decision, sufficient data should be acquired to support quantitative analysis. Under the theory of traffic engineering, the famous Lane flow equation (**?**) describes the relationship between traffic flow and speed:

$$Q = KV,$$

where Q is the number of vehicles per hour, $V$ is the mean speed, K is the vehicle density, usually can be changed by the speed limit, signals on the ramp entrance.

In general, we want our facilities to have maximal flow capacity, and Q is the quantity of interest. Therefore, the traffic flow Q needs to be obtained to evaluate the changes reflected by the decision made to the infrastructures. Apart from the traditional heavy equipment,

increasing attention and efforts have been put on analyzing the existing traffic videos. For example, IDOT maintains a huge database of videos recorded by traffic cameras across Illinois 24/7. People are hired to manually count the number of vehicles and generate a report for each one-hour video, which is expensive both in labor and time. To help facilitate the process, we build an end-to-end vehicle counting system on top of our fully automatic tracker. The system runs in real time, generates a similar report for each video. It significantly reduces the cost and time for this process.

In practice, the tracking and counting process requires immense computational resource; therefore, these tasks usually run on remote servers. To allow easy access to the data and results, we build additional GUI tools to provide the users interactive operation, such as video upload, result visualization, and report generation. With end-to-end workflow and interactive GUI interface, the system can be easily deployed in large scale.

## 6.2 Vehicle counter

After the tracking process finishes, the number of trajectories is regarded as vehicle counts. However, the current traffic statistics of IDOT contains specific vehicle counts in different directions. To match the format of the existing reports, not only we have to obtain the total number of vehicles, each vehicle has to be correctly classified to its corresponding motion. The accumulative counts of each motion are the desired results. Consequently, vehicle movements have to be available apart from the tracking results. With different tracking strategy, motions are obtained differently: with our heuristic tracker, we rely on human annotated input; however, with the semantic tracker, we learn the motion offline by unsupervised learning.

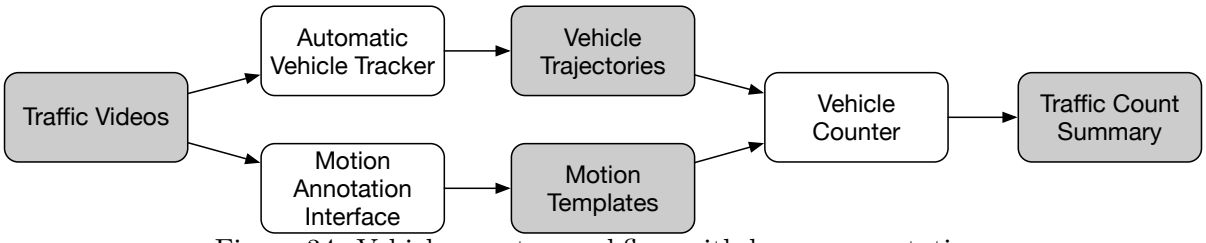



Figure 34: Vehicle counter workflow with human annotation.

### 6.2.1 Vehicle counter with human annotation

Initially, users upload videos via FTP and operated on our GUI interface by remote desktop. Our GUI interface and require the user to draw a few line segments as the templates of the vehicle motion, we call it *motion template*. Figure 35 is the screenshot of the interface, where the motion templates mostly align with the road surface. However, a road may have more than one motion due to potential multiple lanes. Figure 34 shows our first counter framework with the heuristic tracker. The tracker generates a set of vehicle trajectories; then each trajectory is assigned to a motion template that mostly matches its movement. By increasing the count of each vehicle's assigned motion template, we can obtain the final traffic count of each movement.

Suppose we have a set of $n$ motion templates $\mathbf{T} = \{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n\}$. Each template consists of segments $T_m = \{s_1, s_2, \ldots, s_{m_l}\}$ and $s_m$. is a line segment, $m_l$ is the number of the line segments of template $T_m$. There is also a set of trajectories of $N$ vehicles $\Omega$ =

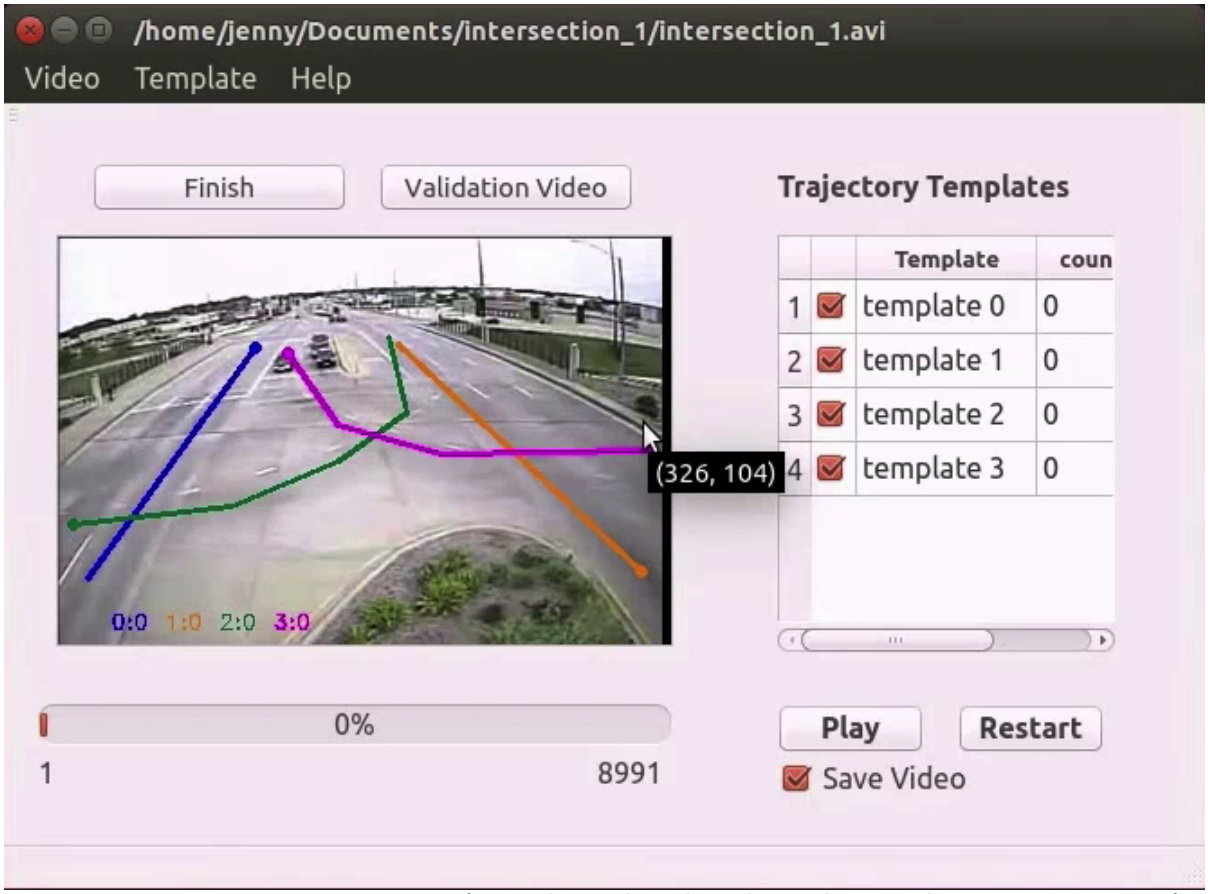



Figure 35: Motion annotation interface: the end with a dot indicates the starting point of a motion, the bottom of the image and the list on the right displays the counting results.

$\{O_1^1, O_1^2, \ldots, O_1^{t_1}, O_2^1, O_2^2, \ldots, O_2^{t_2}, \ldots, O_N^1, O_N^2, \ldots, O_N^{t_N}\}$, where $t_i$ is the lifetime of object $i$. Each object is matched to the template with the minimal maximal distance.

$$T_i^* = \arg\min_{T} \operatorname*{avg}_{t \in \{1,\ldots,t_i\}} D(O_i^t, T). \tag{6.1}$$

$d(O_i^t, T)$ is defined as the minimal distance between the trajectory point and motion template $T$, measuring the distance and direction consistency at the same time:

$$d(O_i^t, T_m) = \min_{s \in \{T_m\}} d(O_i^t, s) \cdot e^{-\alpha \cos(p(O_i^t, s))}.$$

$d(O_i^t, s)$ is the perpendicular distance of a point to the line segment $s$, while $p(O_i^t, s)$ is the angle between $s$ and the moving direction of object $O_i$ at time $t$. $\alpha$ is a constant, currently set as 1.

We develop the following template matching algorithm to ensure the trajectory is matched to the template that fit its motion most. We start matching from the most confident trajectory point and continue matching until reaching the source and sink point of the trajectory. Here the order of their closest segment was considered: all the trajectory points are not matched to the line segments of its previous trajectory point. $s^{it}$ is defined as the closest segment of $O_i^t$, which is the $t$th point of object $i$'s trajectory. The distance $D(O_i^t, T)$ of object $O_i$ to template $T_m$ was computed as follows:

- Find the time $t^*$ such that $t_* = \arg\min_{t \in \{1,\ldots,t_i\}} d(O_i^t, T_m)$ and its closest segment $s^{it^*}$.

- For the points before $O_i^{t*} : O_i^t \in \{O_i^1, O_i^2, \ldots, O_i^{t-1}\}$, the closest segment falls in the first line segment of $T_m$ to its successors closest segment:

$$D(O_i^t, T_m) = \min_{s \in \{s_1, s_2, \ldots, s^{it-1}\}} d(O_i^t, s) \cdot e^{-\alpha \cos(p(O_i^t, s))}.$$

- For the points after $O_i^{t*} : O_i^t \in \{O_i^{t^*+1}, O_i^{t^*+2}, \ldots, O_i^{t_i}\}$, the closest segment falls in its predecessors closest segment to the last line segment of $T_m$:

$$D(O_i^t, T_m) = \min_{s \in \{s^{it^*+1}, \ldots, s_{m_l}\}} d(O_i^t, s) \cdot e^{-\alpha \cos(p(O_i^t, s))}.$$

By defining the distance function $D(\cdot)$ with both spatial distance and direction consistency and keeping the closest segment order, we eliminate close segments with inconsistent directions. Finally, the number of trajectories belonging to each template is the desired vehicle count.

### 6.2.2 Vehicle counter with semantic knowledge

Figure 36 shows the workflow of the improved counter with semantic knowledge. After offline learning in §3.2, the distribution of visual topics are learned; and each of them is a parameterized model. In the tracking process §4.2, the fitting of each tracked vehicle with every model is examined by online inference. In other words, the motion of the tracked vehicle is determined by the most fitted motion model. By increasing the vehicle count of each vehicle's motion upon leaving, counting can be done along with tracking.

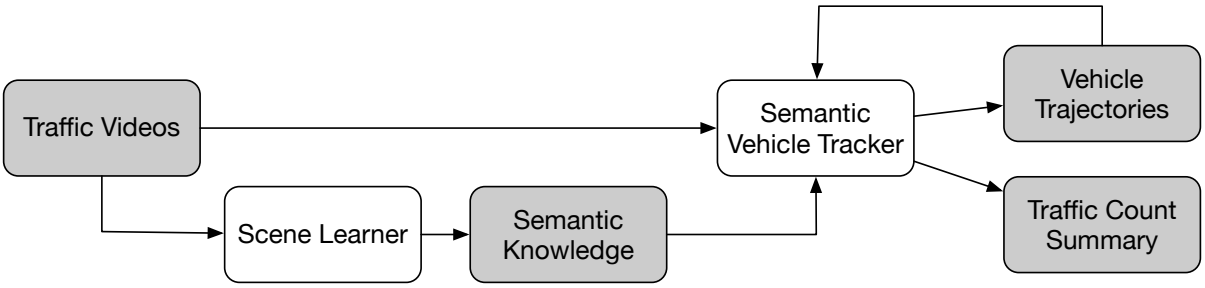



Figure 36: End-to-end vehicle counter workflow with scene understanding.

### 6.2.3 Web portal

Based on the feedbacks from IDOT about our previous GUI interface, we build a web portal of this system that integrates all the previous functions. Currently, the system only supports manual template annotation as in Figure 34, the scene understanding module has not been integrated. Figure 37 is the main interface of camera display: cameras are displayed on the map, with a list of their name on the left. Users may create or browse cameras by list or on the map, and upload videos for each camera. Similar to our previous GUI, once a new camera is created and the first video is uploaded, the user needs to draw the motion templates. Then the tracker and counter are executed sequentially in the background. Users may return later to check the progress and download the results. We allow at most four videos processed at the same time. Figure 38 and Figure 39 are the interface for camera and videos. Each camera may have multiple videos, all its videos, and their count summary is displayed on the left in the camera view. Clicking a video on the list leads to its individual video view. The video is played and detailed counts on each motion are displayed.

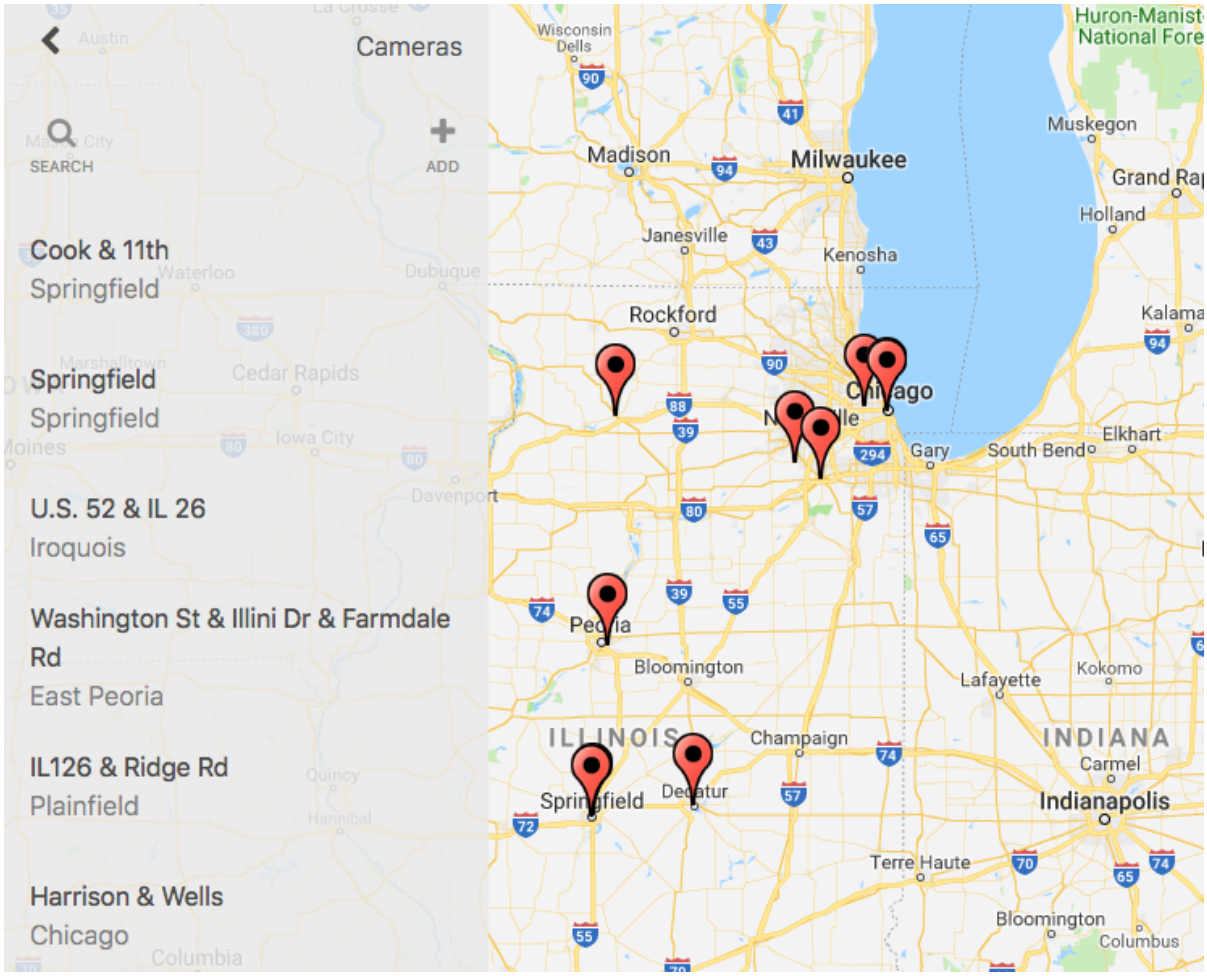



Figure 37: Main interface of the web portal, cameras are displayed on the map.

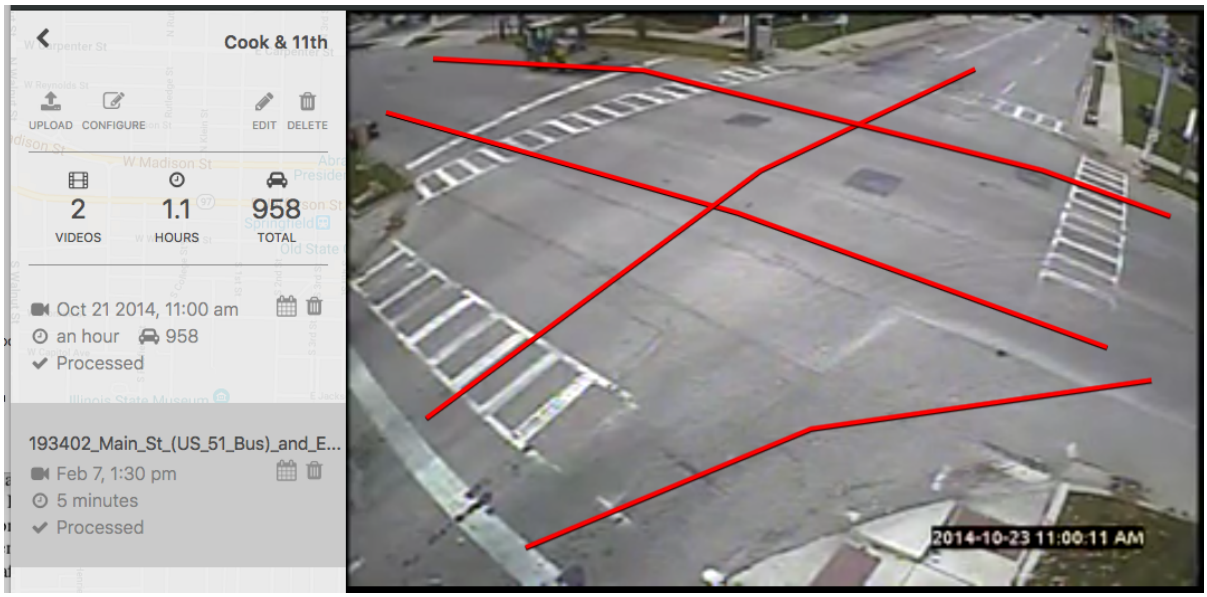



Figure 38: Camera view with video list and summary.

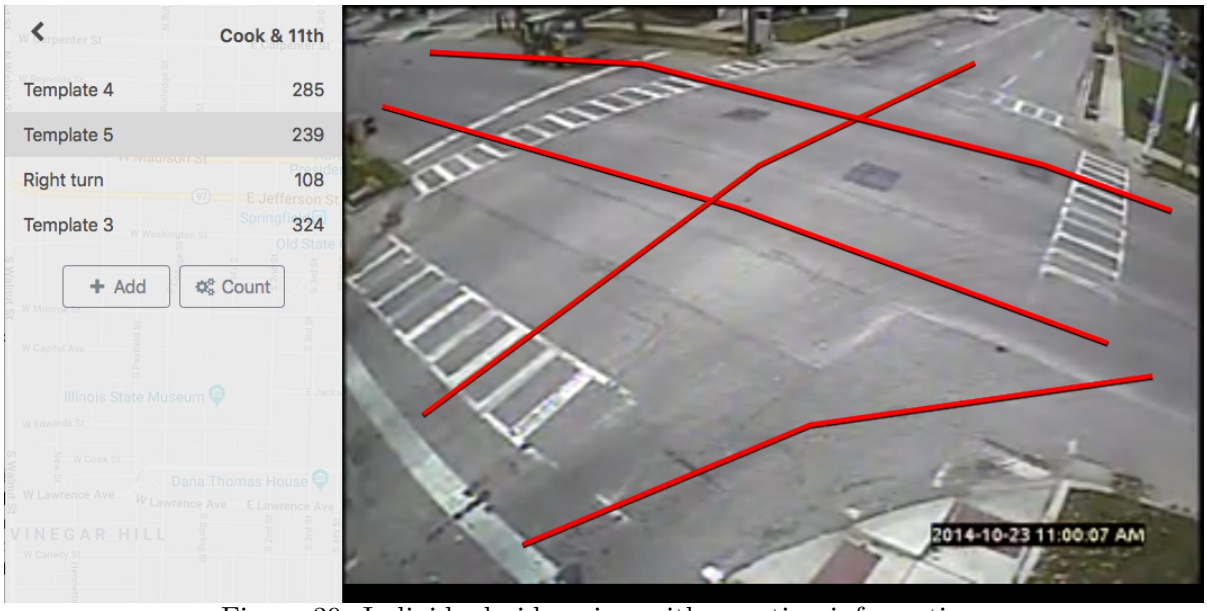



Figure 39: Individual video view with counting information.

# CHAPTER 7

# DATASET

## 7.1 Introduction

In the field that has such a massive amount of video data like intelligence transportation, computer vision has become a primary tool for information retrieval from those videos.

Usually, an agent has access to a specific kind of camera and aim to develop a system to perform a particular task. Therefore, such systems are usually restricted to some specific camera views or settings and hard to migrate to other tasks. Moreover, due to a large amount of videos, there is usually little annotation due to the expensive labor cost. It is hard to carry out a quantitative evaluation for those systems.

On the other hand, there lacks a comprehensive and widely accepted video dataset in the academic community. Although challenges like VOT (**?**) and MOT (**?**) are widely used in the object tracking community, the given data are relatively in small scale and lack real-world interactions, compared with what is directly acquired from the traffic camera. For example, the videos in VOT and MOT mostly have a few hundred frames, lasting no later than 1 minute; even though a few videos exceed 1000 frames, it is still too short compared with real-world data. Such a short sequence cannot raise enough attention to algorithm throughput. Despite the occlusion and consequent difficulties in the above dataset, those videos fail to address the critical challenges for intelligent transportation tasks. Most traffic cameras are mounted at a

high place with little view changes, while the above popular datasets do not contain videos with such overlooking views. Besides, objects tend to have intermittent stops and severe occlusion at intersections, which is also rarely available in the above datasets.

We release a comprehensive traffic video dataset with vehicle trajectory ground truth, containing various adversarial interactions. We aim to provide other researchers with real-world data and raise more attention in such problems.

## 7.2 Dataset

To the best of our knowledge, there exists no public traffic surveillance video dataset containing complex real world interactions and illumination variations. Existing vision datasets are either not applicable to our scenario with different viewpoint (driver's view) (**?**) or contain short clips with limited adversarial conditions, scale changes, and illumination variations (**?**; **?**). Even in the largest dataset collected (**?**), only 15 out of 98 videos exceeds 1000 frames (33 seconds).

We collected 13 representative traffic videos across our state, from the local department of transportation, and annotated these using VATIC (**?**). Each object has its location and extent annotated on every frame, which is used as our ground truth. The average length of each video is five minutes (around 9000 frames), sufficient to cover several traffic signal cycles with real-world vehicle interactions and movement patterns. We divide the videos into two groups: simple low resolution (lowRes) and complex high resolution (highRes). Figure 40 shows screen shots from this dataset, and Table I gives an overview of our dataset, where the rightmost

TABLE I: Dataset overview. The second and the third columns show the resolution and object size range in pixels, followed by number of videos under each group. The rightmost four columns show the number of videos reflecting various challenging aspects (occlusion, shadow, distortion and pedestrian).

| Group | Resolution | Object size | # | Occlusion | Shadow | Distortion | Pedestrian |
|---|---|---|---|---|---|---|---|
| lowRes | $342 \times 228$ | 32–44,814 | 5 | 3 | 1 | 3 | 0 |
| | $320 \times 240$ | 48–25,284 | 2 | 2 | 1 | 2 | 0 |
| highRes | $720 \times 576$ | 84–255,106 | 4 | 3 | 0 | 0 | 1 |

(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k) (l)

Figure 40: Snapshots of videos in our dataset, with various resolution, viewpoint, illumination, vehicle size and interactions. In particular, (c) shows shadows; (f) and (g) show severe distortion by fish-eye camera. We group these videos by their characteristics: (a) - (g) are simple low resolution videos (lowRes), and (h) - (l) are complex high resolution videos (highRes).

four columns indicate the number of videos reflecting various challenging aspects: occlusion, shadows, distortion and pedestrians.

# CHAPTER 8

# RELATED WORK

(This chapter was partially published as "Fully Automatic, Real-Time Vehicle Tracking for Surveillance Video" in Computer and Robotic Vision 2017. DOI: 10.1109/CRV.2017.43)

## 8.1 Object tracking

Object tracking algorithms fall into either multi-object or single object tracking category. Multi-object trackers deal primarily with the combinatorial task of associating sets of detections across frames, often modeled as a graph-based global optimization problem (**?**; **?**). Such methods are computationally expensive, and typically impractical in time-critical applications. Meanwhile, single object tracking has experienced a rapid progress with the help of robust feature representation (**?**) and better learning scheme such as SVM (**?**) and boosting (**?**). However, object tracking still remains a challenging problem due to the difficulty caused by changing appearance and occlusions. To cope with appearance change, better affinity measurement (**?**) and adaptive learning schemes (**?**) may help. For occluded objects, multi-object trackers tend to inherently model occlusion, for example, using augmented graph representation (**?**) while single object trackers maintain the memory of the object appearance (**?**).

## 8.2 Tracking initialization and termination

Trackers today are evaluated in an idealized setting, where manual initialization and termination is provided. Most multi-object trackers assume excellent detector performance, but do

handle object entry/exit. They either model it intrinsically by adding entry/exit nodes to the optimization graph (**?**), or manually by defining an entry and exit area (**?**). In (**?**), the only available literature we found related to tracker initialization, the authors evaluate single-object tracker initialization spatially and temporally, and conclude that spatial and temporal variation of initialization can affect the tracking performance. However, no conclusion on how to make proper initialization is made.

## 8.3 Vision in intelligent traffic system

Recently computer vision techniques are extensively applied in traffic analysis with the wide deployment of traffic surveillance camera, such as real-time vehicle tracking (**?**), vehicle counting (**?**), parking occupancy detection (**?**), anomalous event detection (**?**). Compared with core computer vision algorithms, such systems face more real-world challenges and restrictions. Therefore manual input is often added to achieve reasonable performance. For instance, image–real world coordinates mapping beforehand (**?**), lane width on image (**?**), and entry region for vehicle detection (**?**). Another observation is that those systems are only applicable to videos of a certain view. For example, (**?**) uses top-front view of high way videos, making vehicles roughly the same size. Therefore, the portability is significantly limited by manual input and task-specific applications.

## 8.4 Scene understanding

Most visual traffic analysis system heuristically relies on prior knowledge to guarantee the accuracy for practical use. Commonly used methods include a specific deployment of the cameras or some human input. For example, by calibration with known camera specifications,

(**?**; **?**) restores the real-world coordinates and infers the actual measurement of the tracked objects. Another type of work requires the area of interest for the tracked objects in advance, such as entry/exit area in (**?**; **?**; **?**), or a skeleton of the road surface in (**?**). Since all of the above methods require some camera-specific settings, none of them is easy to apply to a new camera.

To understand the scene in the videos, current work either learns the semantic areas or movement patterns. The former such as (**?**; **?**; **?**) learn the entry/exit areas from the available trajectories or on-the-fly. Such methods require clean trajectories, which may not always be available in practical use. On the contrary, the latter (**?**; **?**; **?**; **?**) statistically learn the motions from the quantized optical flow, without knowledge of the actual objects in the scene; therefore, they are more robust and flexible.

On the other hand, some researchers are working on the semantic aided tracking. (**?**; **?**) use motion information to constrain the movement of the tracked objects; however, they are only for crowded scenes, since single object-trackers prefer appearance-based features. There is also work exploring scene evidence from trajectories: either from existing trajectories (**?**) or hand-drawn artificial trajectories (**?**). However, reliable trajectories still remain an issue for real-world applications.

Despite the robust results in Bayesian motion learning, little work has extended them to tracking. Zhao etc. (**?**) applies (**?**) to vehicle counting. However, we still lack a general component for automatic initialization/termination to fit in our fully automatically tracking framework.