Extraction of Vehicle Trajectories from Online Video Streams

Xinhe Ren
David Wang
Michael Laskey
Ken Goldberg

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-44
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-44.html

May 10, 2018

Extraction of Vehicle Trajectories from Online Video Streams

by

Xinhe Ren

A technical report submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ken Goldberg, Chair
Professor Trevor Darrell

Spring 2018

The technical report of Xinhe Ren, titled Extraction of Vehicle Trajectories from Online Video Streams, is approved:

Chair _____ Date 5/7/18

_____ Date 5/7/18

_____ Date _____

University of California, Berkeley

Extraction of Vehicle Trajectories from Online Video Streams

Abstract

Extraction of Vehicle Trajectories from Online Video Streams

by

Xinhe Ren

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Ken Goldberg, Chair

To collect extensive data on realistic driving behavior for use in simulation, we propose a framework that uses online public traffic cam video streams to extract data of driving behavior. To tackle challenges like frame-skip, perspective, and low resolution, we implement a Traffic Camera Pipeline (TCP). TCP leverages recent advances in deep learning for object detection and tracking to extract trajectories from the video stream to corresponding locations in a bird's eye view traffic simulator. After collecting 2618 vehicle trajectories, we compare learned models from the extracted data with those from a simulator and find that a held-out set of trajectories is more likely to occur under the learned models at two levels of traffic behavior: high-level behaviors describing where vehicles enter and exit the intersection, as well as the specific sequences of points traversed. The learned models can be used to generate and simulate more plausible driving behaviors.

Dedicated to my parents and family

Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my research advisor Professor Ken Goldberg for the opportunity to work in the AUTOLAB. The fascinating research opportunities he provided me have greatly enriched my academic experience at UC Berkeley. Professor Goldberg does not compromise for mediocrity, and has constantly pushed me beyond my comfort zone towards more ambitious goals. His ideas and insights showed me what great research manifests.

I especially thank my mentor Michael Laskey for supporting me throughout my research experience here. He showed me what meticulous research look like. Michael's guidance and way of thinking compelled me to see the big picture where concepts spanning different aspects of mathematics, machine learning, and robotics are in fact closing linked. I am incredibly grateful for the opportunity to work with him, and his advises on my future careers.

I also thank my partner in this project, David Wang. He contributed greatly to this work, and we would not have achieved these results without his help. A perfectionist, David showed me what it means to be scrupulous in research. I wish him the best in his master's career next year.

I would also like to thank the other members of the AUTOLab for contributing to a great research environment. This lab is an incredible environment full of talents who are always willing to lend a helping hand. The researches coming out of AUTOLab are cutting edge as a result. I also thank Professor Trevor Darrell for his help on my masters thesis committee.

Finally, I would like to thank my family for their support throughout the last four years of my undergraduate and graduate career at the University of California, Berkeley.

Chapter 1

Introduction

Autonomous driving simulators offer the potential to rapidly prototype behavioral algorithms. However, a significant concern with developing a simulation is how accurately it reflects the physical world. Currently, autonomous driving companies collect real-world traffic data through countless road trials and expensive sensor and telemetry equipment, a luxury possessed only by a few. In recent years, live streaming has become a popular internet trend. The availability of cheap, live traffic camera streams provides us an economical alternative method of studying traffic interactions.

We examine how to leverage online video streams (e.g., a YouTube stream of a four-way traffic intersection at 7 Ave and Main St in Canmore, Alberta[†]) to learn high-level driving behaviors and trajectories capturing vehicle motions (Figure 6.1). Using online video streams for data collection has the potential to capture a significant amount of driver demonstration data, however, we must address the challenges of perspective, skipping, and low resolution.

We propose a Traffic Camera Pipeline (TCP), which applies recent deep learning advances in object detection [21] to detect vehicles in the video stream. We then use homography to register the located vehicles to the corresponding positions in the intersection. We also propose a filtering algorithm to assign observations of vehicles to their respective trajectories by maximizing the likelihood of an estimated distribution over observations. In addition, we explore deep learning based tracking [8], and compare the result to that from our likelihood estimator based method. The output of the system is a set of trajectories of vehicles.

We use the extracted trajectories from TCP to learn two levels of traffic behaviors. First, we consider the high-level behaviors of vehicles, which describe where they enter and exit the intersection. From this information, we can compute plausible distributions of where vehicles appear and what general movement actions (i.e., turning or moving forward) they take based on real data. For example, in the traffic feed, we detect a preference for Main St over 7 Ave. Next, we consider the specific positional trajectories traversed by the agents. We train a generative model on the trajectories extracted by TCP as a distribution over cubic-splines in the plane. We compare the learned models of traffic behaviors against those used in a simulator, and we find that the learned models better fit a held-out set of TCP-collected data, suggesting that TCP can produce models that simulate more plausible driving behaviors.

[†]https://canmorealberta.com/webcams/main-street

Chapter 2

Related Works

2.1 Automated Extraction of Traffic Data

Traffic data historically has been sourced from both traffic cameras and sensors as well as onboard sensor arrays. Research on traffic detection using traffic cameras historically used classical machine vision and digital signal processing techniques to produce real-time traffic scene analysis [14]. More recent work has investigated improving static intersection vehicle detection methods with existing video image vehicle detection systems [38] through improved sensor fusion with camera systems.

Current sources of traffic and vehicle detection data for autonomous vehicle research mostly focus on using vehicle sensors, such as onboard cameras or LIDAR, to detect and identify nearby objects [18]. Another approach is to equip vehicles with GPS sensors and then analyze the decision-making data afterward to examine traffic congestion in urban settings [2]. With the rising popularity of deep convolutional neural networks, it is interesting to explore collecting traffic behavior through fixed traffic cameras without the need for sophisticated vehicle telemetry.

2.2 Simulating Driving Behavior

Driving Simulators

There exist several open-source driving simulation platforms that have been used extensively in autonomous driving research. CARLA [6], an end-to-end simulation platform, provides photorealistic urban environments from a first-person perspective. These simulators leverage hand-tuned controlled agents and both vehicles and pedestrians are designed to follow specific rules such as staying in lanes and stopping at traffic lights. FLUIDS[‡] is another open-source light-weight Python-based traffic intersection simulator intended for easily customizable extensions. In our experiments, we apply TCP to FLUIDS. However, our method could potentially be extended to more complex simulators.

[‡]https://github.com/BerkeleyAutomation/Urban_Driving_Simulator

Data-Driven Simulation Improvement

Cha et al. [3] built an interactive driving simulator in a data-driven approach: they collect control inputs (steering, acceleration, braking) and dynamic motions (linear acceleration and angular velocity) from real road-driving samples to build a database of primitives. The simulator is then able to produce realistic motions in response to user inputs. Chu et al. [4] analyze traffic at a highway using inductive loop detectors and use the observations to build a microscopic traffic flow model. Ngan et al. [23] take traffic videos and use the data to develop a model for traffic behaviors such as vehicle speeds and queue lengths. We aim to fine-tune a simulator that can exhibit more plausible behavior using similar data-driven methods via video streams.

2.3 Trajectory Extraction

Object tracking and trajectory extraction are widely studied in the computer vision domain. Hu et al. [12] conducted a survey in 2004 on video surveillance of object motion and behaviors. Specifically, the survey examined region, active contour, feature, and model-based object tracking. One of the earlier studies in 1994 by Koller et al. [14] examines traffic flow on a highway by computing affine transformations of vehicles between sequential frames of the surveillance video. However, the affine transformation assumption breaks down when vehicles execute angle shifting motions such as turning. Vidal et al. [36] applied a Kalman filter to a feature based object tracking algorithm, greatly improving the performance. More recently, Li et al. [19] developed a model based pedestrian tracking system with human segmentation. The system applies a human model prior as a seed for segmentation. In our experiment, we first apply a simple likelihood-based method to classify time agnostic bounding boxes into different candidate trajectories.

Recently, we begin to see many deep learning based approaches to object tracking. In 2005, MD-Net [22] became the first deep learning method to win The Visual Object Tracking challenge (VOT) [15]. Leveraging large training datasets, many offline-trained neural networks can track objects with very high inference speed. Held et al. [10] proposed a convolutional neural network on tracking generic objects, which generalizes to novel objects and tracks at 100 fps. Bertinetto et al. [1] use a Siamese CNN network architecture to detect differences between video frames to track objects, also enabling the network to be trained offline. The Re3 (Real-Time Recurrent Regression Networks) by Gordon et al. [9], in addition to a CNN for vision processing, employs a recurrent neural network for temporal information handling. In our experiment, we test a pre-trained Re^3 network[§] as a potential deep learning based replacement for our likelihood estimator based tracking method.

2.4 Learning from Online Videos

There have been many instances of learning from online videos, which can be noisy and ill-constrained. Ulges et al. [35] utilized YouTube content for the autonomous training of a video tagger. Prest et al. [26] trained an object detector from weakly annotated videos on the internet

[§]https://gitlab.cs.washington.edu/xkcd/re3-tensorflow

using domain adaptation to improve the performance of the detector. In robotics, Yang et al. [39] explored learning robot manipulation tasks (grasping) by processing videos from the World Wide Web. The paper used a convolutional neural network for object recognition, and action grammar parse tree to interpret the videos' unconstrained semantic structures. Niebles et al. [24] developed a system to learn human motions from YouTube videos. Srivastava et al. [31] use LSTM neural networks to perform unsupervised learning on YouTube videos. Sorschag [30] conducted a survey of video annotation techniques by examining how to collect large amounts of video for machine learning algorithms. Another survey by Vishnu et al. [37] examined how the prevalence of web videos has enabled large-scale learning.

In this study, we use websites such as `earthcam.com` and YouTube, which feature more than hundreds of live traffic cam streams. We apply deep learning object detection networks to these videos to extract driving behaviors that can be used in simulations.

Chapter 3

System Overview

Figure 3.1 shows the video processing procedure of TCP. The pipeline has four main steps:

1. Collect traffic video.
2. Detect vehicles via convolutional neural network.
3. Extract valid trajectories.
4. Learn high-level statistics about trajectories of vehicle motions.

3.1 Capturing Video

TCP uses OpenCV as the interfacing framework to video streams. As a result, TCP is able to read common video file types (e.g. mp4, wmv, avi) and load online stream MIME types (e.g. ogg, m3u8).

Using TCP, we have created a dataset of 234 annotated, minute-long videos with 2618 labeled vehicle trajectories, and 8980 additional unannotated minute-long videos (6+ days).

3.2 Example Intersection

Figure 3.2 illustrates the four-way intersection in Canmore[†] in our experiment. It was chosen for its unobstructed view of the intersection. We downloaded footage from the traffic cam video stream for further processing. However, we also noticed frame skips in the collected videos.

3.3 Detection and Tracking Overview

The core of TCP is vehicle detection and tracking. We explore how we can leverage recent successes in deep learning to accurately track vehicles. Chapter 4 explores many pre-trained object detection

[†]https://canmorealberta.com/webcams/main-street

Figure 3.1: TCP system architecture (excluding learning and analysis). First, we capture a video stream of a traffic intersection and use SSD, a deep object detection network, to identify and label vehicles. Then, we manually label the first detection of each vehicle in the video stream. Finally, we map the identified vehicles to a bird's eye view using homography and run a probabilistic grouping algorithm to extract trajectories.

Figure 3.2: TCP captures a four-way intersection in Canmore, Alberta at different times of day. It features a variety of lighting conditions, weather, and road conditions. For the following experiments, we only labeled a small subset of the daytime videos.

networks and their performance on detecting cars. Once we obtain the bounding boxes of cars in each video frame, we assign these bounding boxes to trajectories of vehicles over time.

In this report, we explore two ways of tracking: likelihood estimator based and deep learning based. The formal method requires a human supervisor to label the bounding boxes in which vehicles first enter the scene. A likelihood-based estimator then assign any subsequent bounding boxes to the most likely trajectory candidate. The deep learning method, Re3, uses the bounding boxes output from the detection network as initial tracking labels and then outputs the tracked bounding boxes in each subsequent frame. We will discuss this in more detail in chapter 5.

Chapter 4

Vehicle Detection

Advances in deep learning over the last several years have significantly improved perception and object detection in images. Recently, deep neural networks such as Single Shot Detector (SSD) [21] and Faster-RCNN [27] have demonstrated surprising performance on many object detection datasets, including PASCAL VOC [7] and COCO [20], in which the goal is to classify objects and localize them using bounding boxes. These datasets include familiar objects, such as vehicles and pedestrians. After benchmarking several deep networks on a hand-labeled held-out dataset of collected images from TCP, we found that SSD has the highest performance. TCP utilizes the pre-trained SSD network to label vehicles in the collected data with real-time performance.

4.1 Comparing Different Object Detection Networks

There are many deep neural networks architectures for object detection, often trading off between speed and accuracy. Also, sharing pre-trained network weights has become increasingly common. Many network models are available online for immediate, off-the-shelf inference. In TCP, we are interested in building a fast pipeline for extracting trajectories of cars and people towards real-time data analysis and learning. Thus, we examine using $>$30 fps networks such as SSD. SSD has a simplified network architecture that combines a feature extractor with additional layers that perform object detection at various scales. We compare several feature extractors: VGGNet [29], InceptionNet [32], and MobileNet [11]. Google's open-source TensorFlow Object Detection API[‡‡] [13] provides implementations of Faster-RCNN, SSD-InceptionNet, and SSD-Mobilenet trained on the COCO dataset. We also examine an implementation of SSD-VGGNet[¶] trained on the PASCAL VOC dataset.

To benchmark these object detection networks for TCP, we hand-labeled held-out datasets of 100 daytime images collected by TCP. We used the BBox-Label-Tool[‖] to draw bounding boxes around objects. To evaluate the networks, we use the following procedure:

[‡‡]https://github.com/tensorflow/models/tree/master/research/object$_detection$

[¶]TensorFlow implementation of SSD-VGGNet: `https://github.com/balancap/SSD-Tensorflow`.

[‖]Bounding box labeling tool: `https://github.com/puzzledqs/BBox-Label-Tool`.

Class	Network	Precision	Recall	Average IoU
Car	SSD-VGGNet	**0.907**	0.354	**0.787**
	SSD-InceptionNet	0.427	0.273	0.706
	SSD-MobileNet	0.431	0.258	0.702
	Faster-RCNN	0.570	**0.553**	0.702

Table 4.1: Results of evaluating four object detection networks on the hand-labeled daytime dataset. We report the precision, accuracy, and average bounding box quality (IoU) of true positive detections for the car class labels. The dataset contains a total of 718 cars.

Class	Network	Precision	Recall	Average IoU
Car	SSD-VGGNet	**0.990**	0.758	**0.796**
	SSD-InceptionNet	0.854	0.320	0.718
	SSD-MobileNet	0.625	0.313	0.689
	Faster-RCNN	0.680	**0.797**	0.750

Table 4.2: Results of evaluating four object detection networks on the hand-labeled nighttime dataset. We report the precision, accuracy, and average bounding box quality (IoU) of true positive detections for the car class labels. The dataset contains a total of 128 cars.

- For a particular image, obtain the detection predictions from the network, which include a class label, confidence score, and a bounding box.
- Discard any predictions with a confidence score lower than 0.5, the default threshold defined in the balancap/SSD-Tensorflow repository.
- Compute the Intersection over Union (IoU) between each detected bounding box and each of the ground truth bounding boxes in that same image. If the best IoU match is greater than 0.5 for detection, it is considered a true positive, and we record the bounding box quality. If there are no matches of sufficiently high quality, the detection is considered a false positive.
- After processing all detections, any ground truth detection that was not matched by a predicted detection is a false negative.

We complete this procedure for each of the four networks on both hand-labeled datasets. We evaluate both detections of cars and people. Table 4.1 lists the resulting precision, accuracy, and average bounding box quality for the two object classes. We also show examples of the detections outputted by the four networks on an example image in Figure 4.1. We note that the recall values are quite low due to the difficulty of the hand-labeled test dataset. The images are of a busy intersection with many cars that are close together, often partially occluding each other. Also, there may be cars at a farther distance that are difficult for any network to detect.

Although we observe higher recall values with Faster-RCNN, we use SSD-VGGNet because of its higher precision and better bounding box quality on our dataset.

Figure 4.1: Example detections from each of the four networks on a sample daytime image: SSD-VGGNet (top left), SSD-InceptionNet (top right), SSD-MobileNet (bottom left), and Faster-RCNN (bottom right). Detections of cars are shown with blue bounding boxes, and detections of people are shown with green bounding boxes.

Chapter 5

Trajectory Extraction

5.1 Homography: From Camera to Bird's Eye Perspective

The bounding boxes generated by SSD gives the location of vehicles within the RGB image taken from the traffic camera perspective. To obtain the locations of the agents in the simulator, we utilize homography [33] to rotate the camera to a bird's eye viewpoint.

Homography works by estimating a projective transformation matrix that morphs pixel locations from a source domain into a target domain. In this study, we are transforming pixel positions in the camera view domain to locations in the bird's eye view of the intersection. We estimated the matrix on four pairs of corresponding points between the camera and the top-down views. The points were selected such that the corners of the intersection in the traffic camera matched the corners of a square centered in the top-down view.

After homography, we still need to specify a point from the bounding box that corresponds to where the vehicle is centered on the road. To determine a point on the road, we use the midpoint of the bottom edge of the bounding box. However, this selected point may not correspond the vehicle's true point on the road, so we learn a linear mapping from the selected point to a corrected point to adjust for the inaccuracy using a hand-labeled dataset of corrections to apply.

5.2 Likelihood Estimation of Trajectory

Given the bounding boxes in each frame, we need to assign each bounding box to a trajectory over time. We convert the bounding boxes into points in the top-down view space with homography as described in section 5.1. Each extracted point denotes an observation $\mathbf{y}_t \in \mathbb{R}^2$ in the bird's eye view. A filtered trajectory τ, is a sequence of observations (i.e., $\tau = \{\mathbf{y}_t\}^T$). Our filtering algorithm works by first having a human manually label the initial state of each vehicle (i.e., decide when a vehicle enters the scene), which creates a list of trajectories, each with a single initial observation. This manual step is required even in recently developed tracking methods of objects in videos [9], but we discuss potential ways to automate this step in section 5.3.

Given existing trajectories, τ, we can define the probability of a new observation as $p(\mathbf{y}_{t+1}|\tau)$. The probability of a new observation on a trajectory, $p(\mathbf{y}_{t+1}|\tau)$, is specified as a Gaussian over

four features $\Phi(\mathbf{y}_{t+1}, \tau) = [x_{t+1}, y_{t+1}, \psi_{t+1}, \lambda_{t+1}]$. $(x_{t+1}, y_{t+1}) = \mathbf{y}_{t+1}$ indicates the position of the vehicle, ψ_{t+1} indicates the angle giving the orientation of the vehicle, and λ_{t+1} is an indicator if the evaluated state violates the traffic laws with respect to the rest of the trajectory. Let $\mathbf{y}_s$ be the latest observation added to τ. We define ψ_{t+1} and λ_{t+1}.

$$\psi_{t+1} = \text{arctan2}(y_{t+1} - y_s, x_{t+1} - x_s)$$
$$\lambda_{t+1} = f(\mathbf{y}_{t+1}, \mathbf{y}_s)$$

f is an indicator function for violation of traffic laws (e.g., merging into lanes with oncoming traffic) based on the new observation and the most recent observation in τ from which we can infer if there is a violation. Then, $p(\mathbf{y}_{t+1}|\tau)$ is a Gaussian $\mathcal{N}(\mu, \Sigma)$ with $\mu = \Phi(\mathbf{y}_s, \tau)$. We manually set the parameters of the covariance matrix, Σ, to be $\text{diag}([6.0, 6.0, 2.0, 100.0])$, a diagonal matrix with the given entries along the diagonal. These values represent a preference for choosing points that are close together in position, and then using the similarity of angles of the two points as a second criteria. Finally, a substantial weight is placed on the violation of a traffic law if it occurs. The algorithm works by iteratively assigning each new observation to the existing trajectories with the highest probability.

Due to noise in object detection and dropped frames, once we have a filtered trajectory, $\tau = \{\mathbf{y}_t\}^T$ consisting of a sequence of T observations, we still want a smooth representation of the trajectory using two dimensional cubic polynomials, a good low-approximation of the vehicle's path. We consider a function $f : \mathbb{R} \times \mathbb{R}^8 \to \mathbb{R}^2$, which corresponds to a parameterized polynomial with 8 parameters, which we denote as φ. See [5] for more details.

We can fit this function to a trajectory via the following optimization problem

$$\varphi^* = \arg\min_{\varphi \in \mathbb{R}^8} \sum_{t=0}^{T-1} \|\mathbf{y}_t - f\left(\frac{t}{T-1}, \varphi\right)\|_2^2$$

to get the parameters to fit the curve. Using this fitted curve, we extract two high-level features of the trajectory: the starting location of the trajectory, and the high-level action taken (left turn, right turn, forward, or stopped). Finally, a Gaussian filter is applied to further smooth the curve.

5.3 Deep Learning Based Tracking

Section 5.2 relies on the deep neural network, SSD, to extract vehicle bounding boxes, and then estimates trajectories under human supervision and probabilistic estimations. In this section, we explore end-to-end deep learning methods for object tracking, in particular, Re3 [8]. Re3 is a deep neural network that combines the image processing advantages of a CNN with the temporal data processing power of an RNN. In this study, we use a pre-trained version of Re3[§] released by its paper's authors.

Re3 can be trained offline. This means that the network's tracking ability can generalize to arbitrary objects, and does not need to be trained on samples of specific target objects. The inputs

[§]https://gitlab.cs.washington.edu/xkcd/re3-tensorflow

to the Re3 network during tracking are the sequence of images and an initial bounding box of the object of interest. The network also supports the tracking of multiple objects in the same frame, which helps us track multiple vehicles in the same video clip.

We use SSD to generate and supply potential initial bounding boxes and Re3 to accomplish the tracking. Algorithm 1 illustrates how we combine these two networks. We define T to be a list of trajectories being tracked by Re3, f to be a frame of a video as an image, B_{Re3} to be a list of tracked bounding boxes from querying Re3, and B_{SSD} to be a list of bounding boxes from object detector SSD. When discussing if two bounding boxes "match", we consider the metric: intersection over union (IoU). If two bounding boxes have an IoU greater than 0.7, an empirically chosen threshold, then we consider the two bounding boxes "matching".

$T \leftarrow []$
while *has next frame,* f **do**
  $B_{Re3} \leftarrow$ Re3.getBBox(f,T)
  $B_{SSD} \leftarrow$ SSD.getBBox(f)
  for b *in* B_{Re3} **do**
    if b *does not match any in* B_{SSD} **then**
      T.add(Re3.track(f, b))
    end
    if b *outside intersection* **then**
      T.remove(Re3.track(f, b))
    end
  end
end

Algorithm 1: Tracking with SSD and Re3

With Re3, we eliminate the need for a human supervisor who tediously labels all the initial appearances of vehicles. With the 0.7 IoU metric, we essentially reduce SSD's task to only generating initial labels, and reassigned the tracking job to Re3. By eliminating the inefficient human-in-the-loop, TCP is now able to operate in real time, taking advantage of the high fps performance of SSD and Re3.

Figure 5.1 illustrates the comparison between SSD and Re3 outputs. Note that SSD is time and trajectory agnostic, and does not assign bounding boxes to trajectories. Contrarily, Re3 assigns each detected object to a trajectory. For example, "7_1" is the first vehicle trajectory in the video, and "15_2" is the second pedestrian trajectory in the video. Note that SSD failed to detect the car in trajectory "7_2" nor the person in trajectory "15_3" in this video frame. The failure causes frame skip in the likelihood based method. Re3, however, is guaranteed to output a bounding box for each trajectory in each frame.

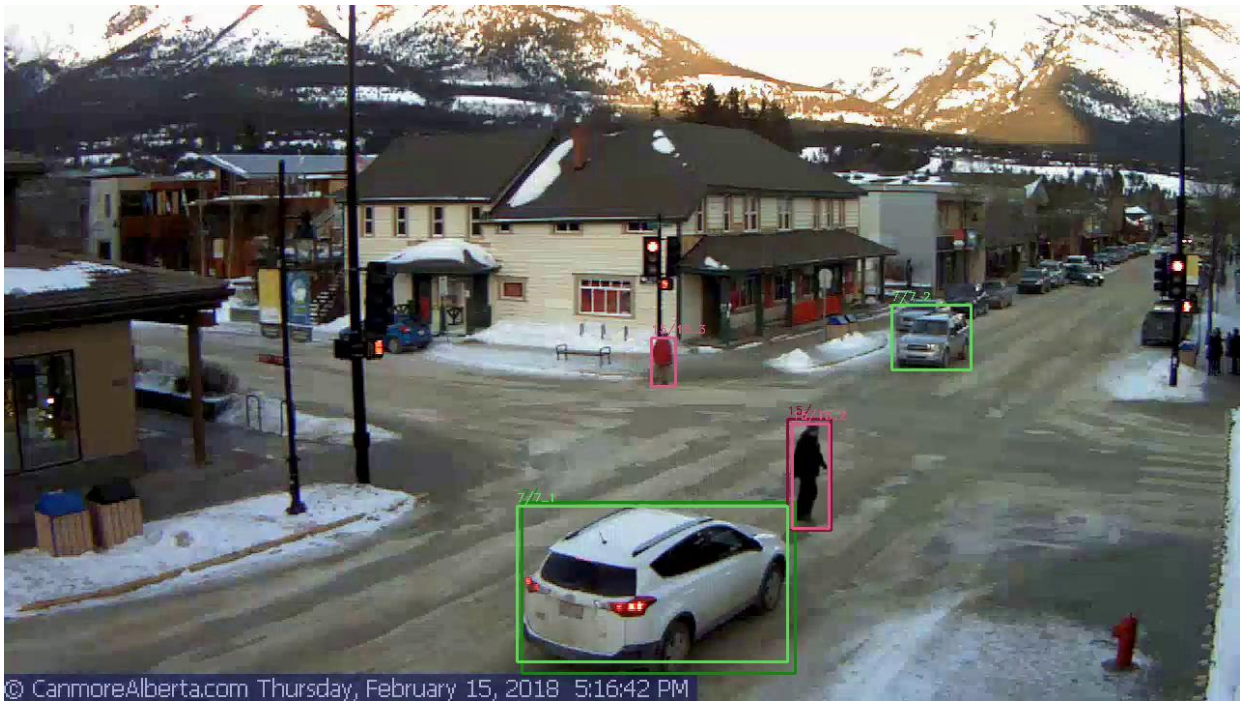

Figure 5.1: SSD and Re3 output bounding boxes with class and trajectory labels. Green represents cars (class 7) and pink represents humans (class 15). Dark green and pink rectangles show bounding boxes from SSD; light green and pink rectangles show bounding boxes from Re3. Both pre-trained neural networks produce consistent and robust bounding boxes for vehicles and pedestrians. In this image, we also show a failure mode of SSD, where the further objects are not successfully detected.

Chapter 6

Learning Driving Behaviors for Simulation

We learn traffic behaviors at a four-way intersection, which can later be used in simulation, by using the trajectories collected by TCP. We consider behaviors on two levels, as shown in Figure 6.1.

6.1 High-Level Behaviors

High-level behaviors describe where a vehicle begins and ends at the four-way intersection. We use two types of distributions to capture these behaviors: distributions over the starting lanes of the vehicles, and distributions over the actions taken (left, right, forward, or stopped) by vehicles given the starting location. We want to learn realistic distributions based on observed trajectories at a real-world intersection.

The high-level behaviors are given by multinomial discrete probability distributions over a set S containing k elements. In the case of the start state distribution for vehicles, we have $k = 4$ for the four lanes. In the case of the action taken by the vehicle given the starting location, we also have $k = 4$ types of actions. Let $D_\varphi = \{\varphi_i \in \mathbb{R}^8\}_{i=0}^{N-1}$ be a set of cubic spline parameters determined by TCP (as described in Section 5.2) for a set of N extracted trajectories from TCP. We choose to use the spline representation of a trajectory due to its smoothness, which allows us to more accurately infer the trajectory's starting state and action. Then, if we have a function $g : \mathbb{R}^8 \to S$ that maps a cubic spline parameterization to element in S, we can estimate a distribution over S, such that for $s \in S$,

$$p(s) = \frac{|\{\varphi \in D_\varphi : g(\varphi) = s\}|}{N}.$$

Then, for a cubic spline parameterization φ corresponding to a held-out trajectory from TCP, we can compute the negative log-likelihood of observing $g(\varphi)$ by computing $-\log[p(g(\varphi))]$.

We will estimate several functions g. The first is g_L, which maps a cubic spline parameterization of a trajectory to one of the four starting lanes: "East, "North, "West, or "South. We also estimate $g_{A,E}$, which maps a cubic spline parameterization of a trajectory starting in the east lane to one of the four action primitives: "left turn", "right turn", "forward", or "stopped". Similarly, we estimate $g_{A,N}$, $g_{A,W}$, and $g_{A,S}$, the distributions of action primitives of trajectories starting in the north, west, and south lanes, respectively.

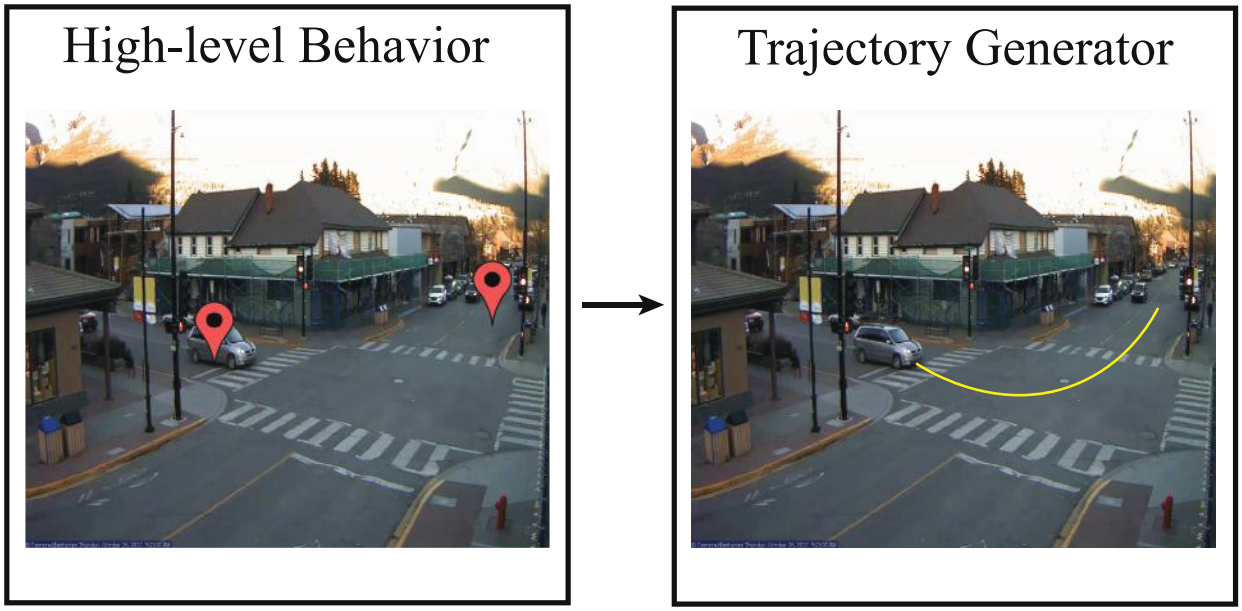

Figure 6.1: We can simulate vehicles at a four-way intersection by specifying traffic behaviors in two steps. (Left) First, we choose a starting lane for a vehicle (the west lane in the figure), and an ending lane (north lane). (Right) After the starting and end lanes are chosen, we can specify a trajectory consisting of a sequence of points for the vehicle to traverse.

6.2 Trajectories

An agent's motion at the traffic intersection can be specified by a sequence of Cartesian coordinates in the bird's eye view perspective. Using trajectories collected by TCP, we learn a data-driven trajectory generator model.

We partition the collected trajectories from TCP into 12 sets: one for each combination of starting lane and the action (left, forward, or right). Let $D_\varphi = \{\varphi_i\}_{i=0}^N$ be the set of cubic spline parameters determined by TCP in Section 5.2 corresponding to the trajectories in one of these sets. We fit a multivariate Gaussian distribution $\mathcal{N}(\mu_{TCP}, \Sigma_{TCP})$ to the D_φ by using the following [28]:

$$\mu_{TCP} = \frac{1}{N}\sum_{i=0}^{N}\varphi_i$$

$$\Sigma_{TCP} = \frac{1}{N}\sum_{i=0}^{N}(\varphi_i - \mu_{TCP})(\varphi_i - \mu_{TCP})^T$$

Then, for a cubic spline parameterization φ corresponding to a held-out trajectory from TCP corresponding to the same starting lane and action, we can evaluate the negative log-likelihood of observing φ from the learned distributions by computing $-\mathcal{LL}(\mu_{TCP}, \Sigma_{TCP}|\varphi)$. Note we repeat these processes for each of the 12 sets. We choose to learn distributions over the cubic spline fit parameters because they are low dimensional approximations of the trajectories.

Chapter 7

Experiments

We explore three questions.

1. How good is TCP in terms of collecting trajectories?
2. How well can we learn high level behaviors?
3. How well can we generate trajectories?

We perform all timing experiments on a 6-core 12-thread Intel Core i7-6850K CPU @ 3.60GHz. We use a traffic cam stream from an intersection in Alberta, Canada[†].

For context, Figure 7.1 shows the map of Canmore, a small tourist town surrounded by mountains. Trans-Canada Highway (Hwy 1) connects the town with the rest of the nation, and Main St, featured in the intersection, joins Canmore with Hwy 1. Additionally, Main Street is in the heart of a busy commercial district.

We labeled four hours of videos from our complete dataset to generate training data. This labeled subset of data correspond to 234 minute-long videos, and 2618 vehicle trajectories.

7.1 Evaluating Traffic Cam Pipeline

Trajectory Yield

The trajectory filtering method sometimes fails when TCP initially detects vehicles in the center of the intersection, where traffic rules are not rigid or well-defined. Hence, we discard filtered trajectories that do not have a clear starting location, action primitive, or contain insufficient (less than 20) data points. With 30 fps input videos, this means that this candidate has less than 0.6 seconds of screen time. Most trajectory rejections are due to object detection failures. Unusually shaped vehicles like delivery trucks are often only detected when they are closer to the camera.

Out of a total of 2618 candidate vehicle trajectories, 13.98% are discarded for having an undefined primitive, and 14.51% are rejected for insufficient data.

[†]https://canmorealberta.com/webcams/main-street

Pipeline Component	Mean Time (s)	Std. Dev. (s)
SSD Detection (Nvidia Titan Xp)	26.11	4.54
SSD Detection (Nvidia Tesla K40)	99.82	18.22
Manual Initial State Labeling	90.36	27.72
Homography & Trajectory Extraction	1.01	0.45

Table 7.1: Average time for a pipeline component to process a one minute clip video (30 FPS), averaged over a total of 100 videos.

Time Efficiency

Table 7.1 contains the average time it takes at each stage of TCP to process a minute-long video clip. SSD [21], or Single Shot MultiBox Detector, is a real-time deep convolutional neural network running in TensorFlow. It omits the need for region proposal, and can achieve fast and accurate detection results that are comparable to that of the state-of-the-art Faster-RCNN [27]. SSD inference performance is heavily dependent on the computing hardware. Table 7.1 shows that a Nvidia Titan Xp GPU can accomplish the object detection task in real-time, but a Nvidia Tesla K40 GPU struggles.

Manual labeling of initial state bounding boxes is the most expensive and labor-intensive step, where a human manually labels which bounding box contains a vehicle never seen before. It is typical in tracking to hand label the initial appearance of an object, and redetect it in subsequent frames[25, 26]. This stage not nessassary with the Re3 tracking. In comparison, homography transformation and trajectory extraction are very efficient.

7.2 Learning High-level Behaviors

Start Location

We use TCP to estimate the vehicle start location distribution. As a comparison, we use FLUIDS (First-Order Local Urban Intersection Driving Simulator), which uses uniform lane sampling for the vehicle start location distribution, in which vehicles randomly appear at one of the four lanes of the intersection.

Figure 7.2 shows frequencies at which vehicles appear from each of the four cardinal directions. The data shows that significantly more vehicles come from the north of the intersection. The dashed line in Figure 7.2 represents the uniform distribution in the baseline.

For a quantitative analysis of the learned distributions of start states for vehicles, we use a held-out set of trajectories collected by TCP, each of which have a corresponding start state. We compare the negative log-likelihoods of learned model and the default model in the baseline given the held-out observations, and we find that the held-out set is more likely under the learned model, as shown in the first row of Table 7.2.

Trajectory Primitive

We classified each vehicle's trajectory into sets of primitives given the starting lane: "forward", "right turn", "left turn", or "stopped".

Figure 7.3 illustrates the frequency of each primitive executed by vehicles coming from each of the four directions. For comparison, we examine the distribution used in the FLUIDS, which chooses each of the four actions uniformly at random.

We examine the vehicle primitives. With the exception of the east, all other directions have more vehicles driving forward. East has more vehicles making right turns toward north instead. Figures 7.1 and 7.2 explain that the north direction is more popular due to the abundance of businesses and connectivity to a major national highway. This knowledge extracted by TCP is currently not reflected by the baseline, which samples from a uniform distribution over the trajectory primitives. The vehicle trajectory primitive information in Figure 7.3 can be used in driving simulators to capture behaviors in a real-world intersection: in this case, to go forward more, and turn more onto the major route.

Similar to the start locations, we also examine the quality of the learned distribution of primitive behaviors by examining the negative log-likelihood of the held-out samples under the learned distributions and the default distributions. Again, we find that the learned distributions perform better than the default distributions, shown in Table 7.2.

Distribution	Baseline Negative Log-Likelihood	TCP Negative-Log Likelihood
Vehicle Start State	1.39 ± 0.0	1.34 ± 0.04
Vehicle Primitives (East)	1.39 ± 0.0	1.19 ± 0.09
Vehicle Primitives (North)	1.39 ± 0.0	1.14 ± 0.10
Vehicle Primitives (West)	1.39 ± 0.0	1.30 ± 0.07
Vehicle Primitives (South)	1.39 ± 0.0	1.17 ± 0.16

Table 7.2: We examine several distributions and compare the confidence intervals for the negative log-likelihood of observing the samples in the held-out set under the default distributions and the learned distributions using TCP (lower is better). We find that the learned model better approximates the held-out distribution in all cases. Note that the confidence intervals for the baseline consist of a single point due to the uniformly random distributions.

Vehicle Arrival

We analyze when vehicles appear in the intersection. Figure 7.4 shows the probability of a new vehicle appearing in the next video frame given the number of vehicles in the current frame. A scene containing many vehicles means that the road is busy. Hence, it is more likely for another vehicle to appear in the next frame. The vehicle arrival distribution can be applied to simulators,

which should decide if a new vehicle should be added according to the current number of vehicles in the scene.

7.3 Trajectory Generator

We use RRT* [16] as the baseline trajectory generator for comparison, the implementation in FLUIDS: given a start and end position in the intersection, the RRT* algorithm attempts to minimize distance traveled, while obeying traffic rule restrictions such as lane directions and road boundaries.

Using a similar process to learn the generative model from TCP trajectories, we learn the parameters of a similar model on a set of 277 trajectories from the baseline: a multivariate Gaussian distribution models the parameters of a cubic-spline fit for trajectories from each combination of starting lane and primitive action. Then, we evaluate the models by computing the negative log-likelihood given samples in the held-out set. The results are shown in Table 7.3. We observe that the negative log-likelihood of the generator trained on TCP data is significantly lower than that of the baseline model for 10 of the 12 combinations of vehicle behavior.

Figure 7.5 shows the qualitative results. It includes examples of real-world held-out trajectories, trajectories sampled from the baseline generative model, and trajectories sampled from the learned TCP generative model for 3 of the 12 primitive behaviors. We observe that the trajectories generated by the baseline generally have less variance than the real-world held-out trajectories due to the many possible trajectories that can be executed by real drivers: many drivers like to cut into the opposing lane for a left turn if they observe no opposing traffic. Similarly, drivers waiting for opposing traffic before a left turn like to pull forward far into the intersection before stopping and yielding. We also observe that the learned TCP model can better match this variability in extracted held-out trajectories.

Lane	Action	Baseline Negative Log-Likelihood	TCP Generator Negative Log-Likelihood
East	Forward	$2,537.1$	44.7
	Left	$2,559.8$	356.6
	Right	$6,161.5$	41.1
North	Forward	$9,238.8$	44.3
	Left	$3,174.4$	50.0
	Right	$2,952.5$	48.0
West	Forward	$18,717.5$	47.8
	Left	$7,707.6$	59.4
	Right	703.4	42.1
South	Forward	$18,340.8$	56.3
	Left	$2,727.0$	857.3
	Right	$3,652.9$	52.0

Table 7.3: We compare trajectories from the learned generator model and the baseline generator model by computing the negative log-likelihood of observing the held-out trajectories given the model (lower is better, bold indicates statistical significance with 95% confidence intervals). We find that the learned generator model is more likely to produce the trajectories in the held-out set than the baseline model for 10 of the 12 primitive behaviors.

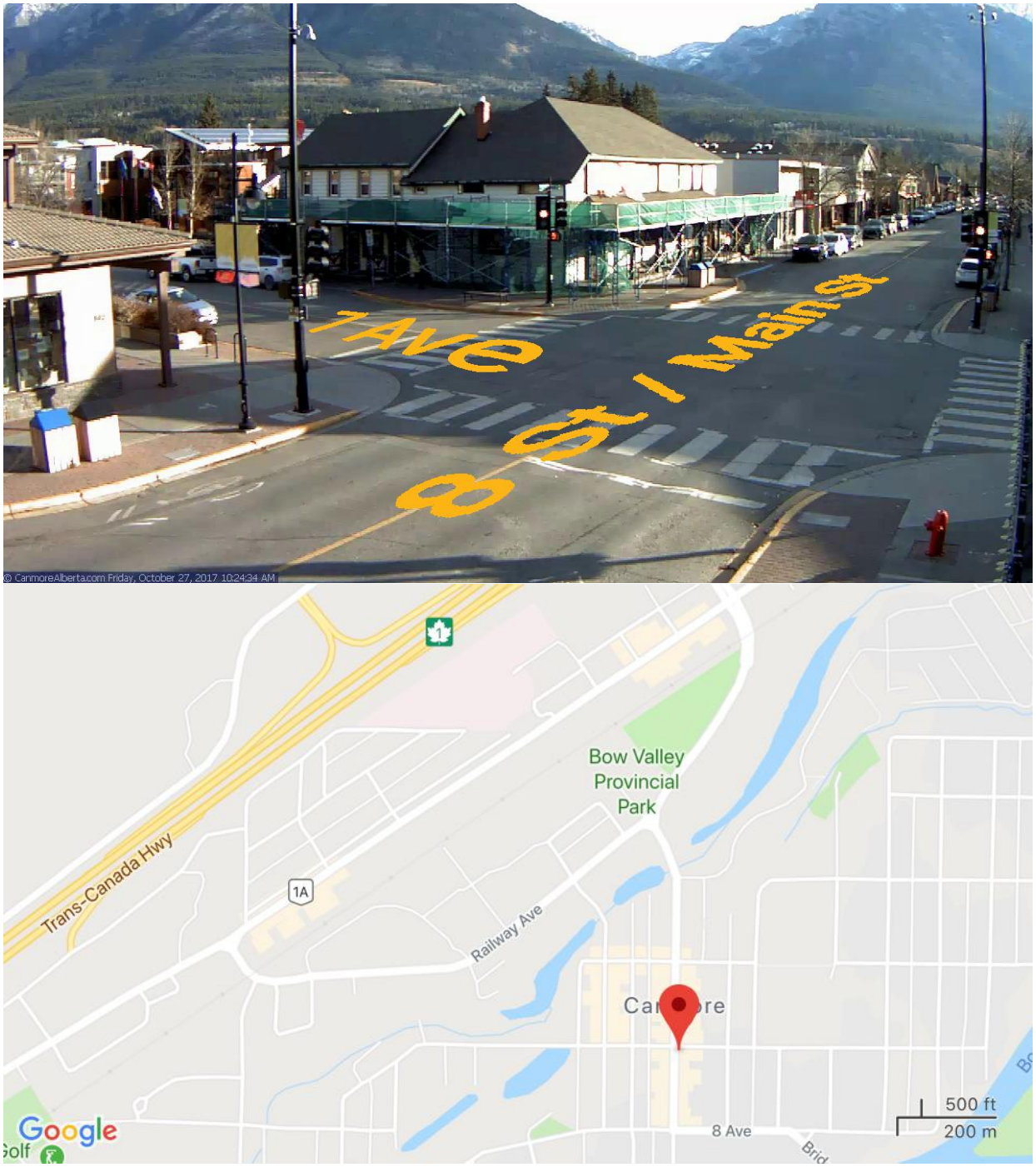

Figure 7.1: Google Map view of the intersection in TCP. Top image shows the street names of the intersection. Bottom image shows the surrounding area of the intersection, and the dropped pin shows the location of the intersection.

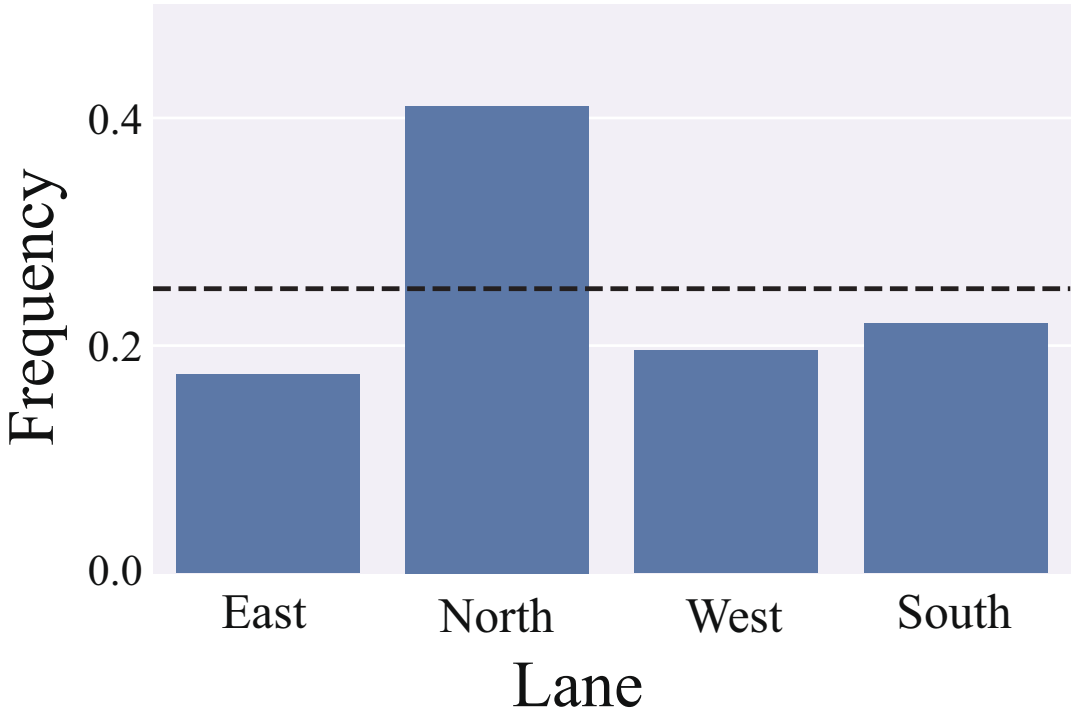

Figure 7.2: Distribution of vehicles by starting lane. The cleaned up data contains 1872 vehicle trajectories. The dotted lines show the baseline uniform distributions.

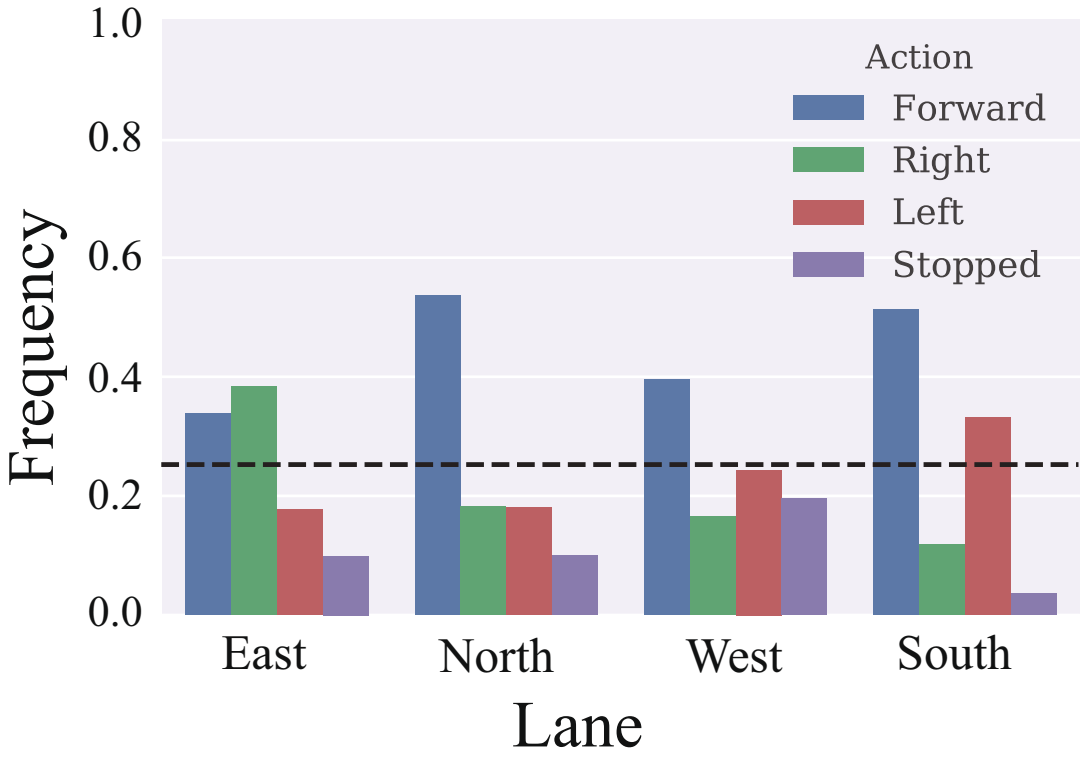

Figure 7.3: Distribution of vehicle trajectory primitives at each cardinal direction. The dotted lines show the default uniform distribution.

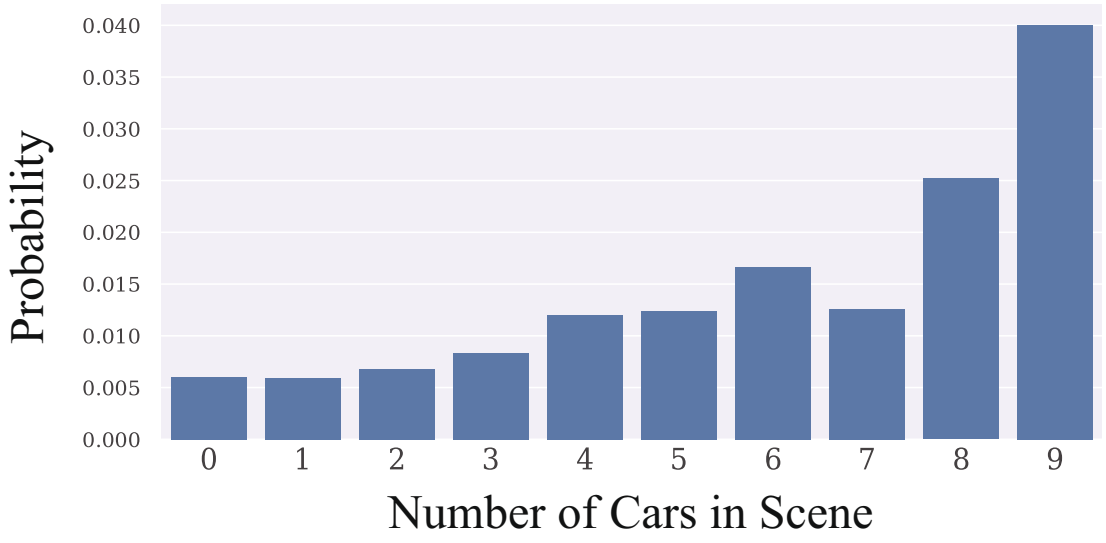

Figure 7.4: Probability of new vehicle appearing given the number of vehicles in current frame.

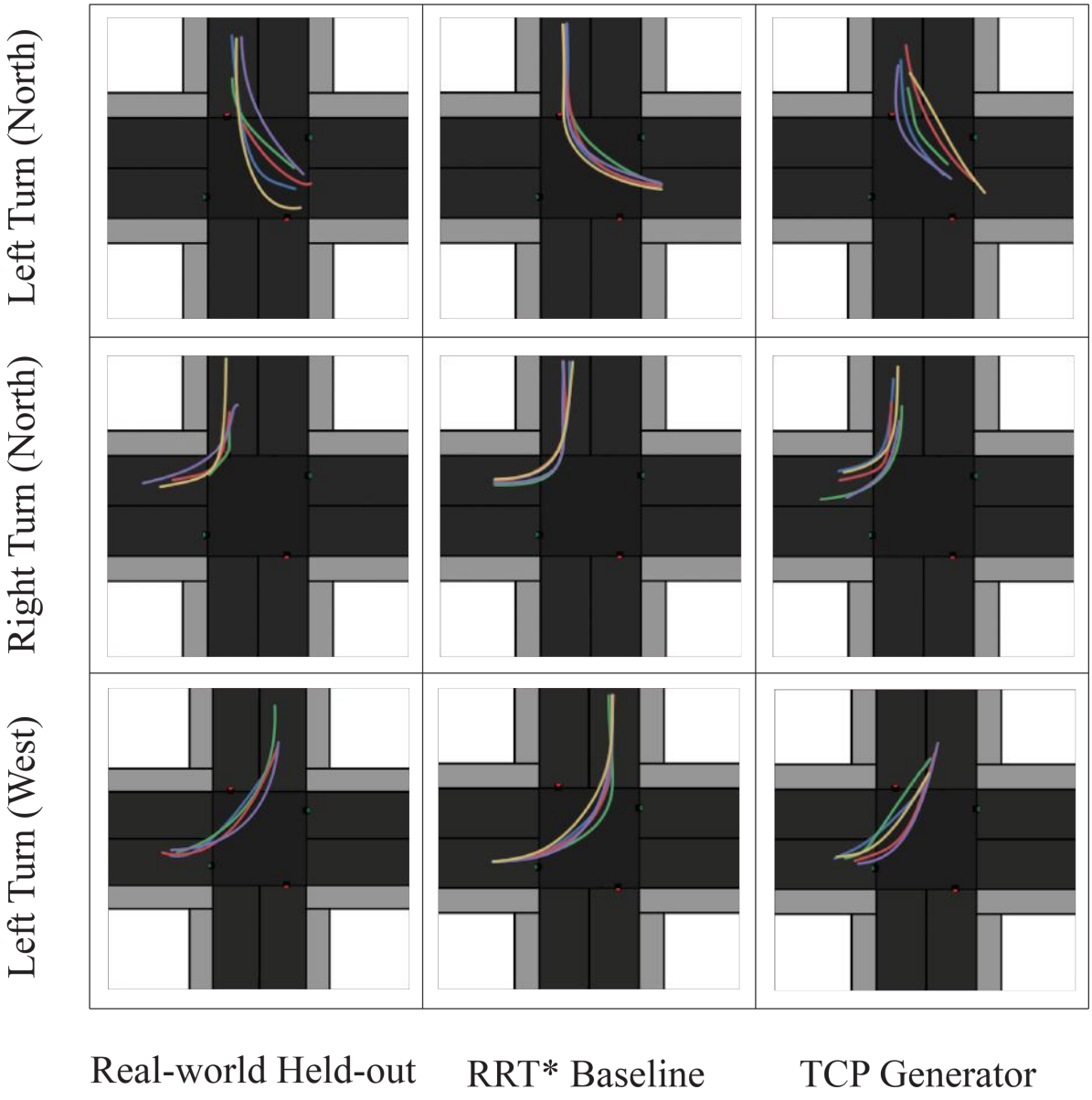

Figure 7.5: Examples of held-out trajectories, trajectories sampled from the baseline trajectory generator, and trajectories sampled from the learned TCP generative model. We show five examples each for three primitive behaviors: left turn from the north, right turn from the north, and left turn from the west. We see that the real-world held-out trajectories exhibit greater variance in paths, and the learned generator better matches this behavior. However, the difference is not as apparent in the bottom row.

Chapter 8

Discussion and Future Works

TCP can extract vehicle data from readily available, yet often unstructured, online public video streams. By learning probability distributions over driving behaviors inferred from the data, we can generate new samples of plausible driving behavior that can be used in driving simulators.

8.1 Evaluation

One challenge with measuring the performance of TCP in learning real-world driving behavior is the lack of ground-truth data to evaluate against. In future work, we will develop better methods to evaluate the learned models by using other metrics, examining intersections with accurate sensors to record ground-truth measurements, or using human annotations as ground-truth.

8.2 3D Reconstruction of Vehicle Orientations

Currently, TCP uses the midpoint of the bottom edge of bounding boxes as an approximation of the of the vehicle's position. To compensate for the inaccuracy in this heuristic if the vehicle is not sideways towards the camera, we trained a linear offset based on the vehicle's position in the scene. A more robust way to represent a vehicle as a point is to reconstruct the 3D model orientation of the car, and then project the centroid of that model onto the camera image plane.

This is a single-view 3D reconstruction task. Tulsiani et al. [34] improved existing techniques by enforcing consistency between 3D model and 2D observation with the introduction of a differentiable ray consistency (DRC) term. Although, in this study, we only wish to identify the orientation and approximate centroid of a vehicle. We do not need our 3D models to accurately match those of cars in the scene. Leotta et al. [17] tracked vehicles by fitting a deformable, generic model of a vehicle to segments of a vehicle in a scene. Improving the projection and homography step of TCP can yield bird's eye view trajectories that better match the ground truth.

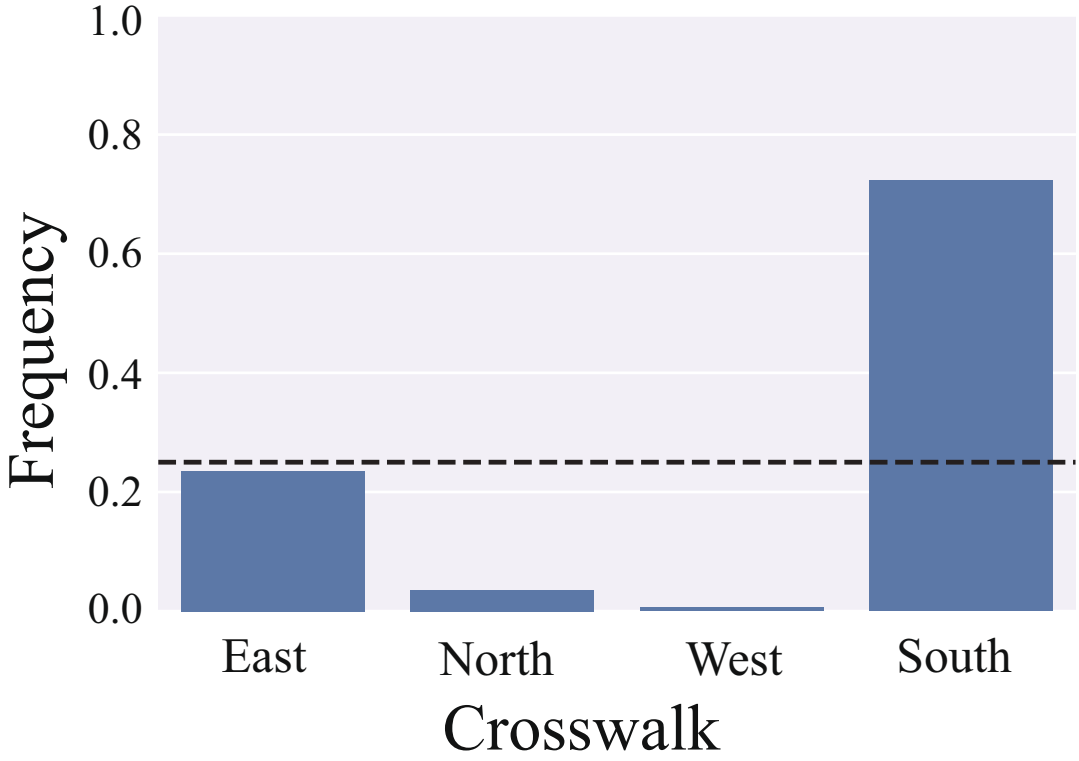

Figure 8.1: Distribution of pedestrians by crosswalk location. The cleaned up data contains 209 pedestrian trajectories. The dotted lines show the default uniform distribution.

8.3 Pedestrians

We hope to expand our pipeline experiment to include pedestrians in the next version: TCP-IP (Traffic Camera Pipeline - Including Pedestrians).

We have conducted some preliminary experiments with pedestrians to extract their start state distribution, shown in Figure 8.1. We speculate that the skew in data is because the camera is mounted in the southeast corner of the intersection. Since pedestrians are much smaller than vehicles, pedestrians in the north and west of the intersection are much harder for SSD to robustly detect and track. Identifying individual pedestrians in groups is also difficult. As a result, 63.81% of the 1213 pedestrian trajectories are rejected due to insufficient data points.

For more details and to download our dataset, see: `https://berkeleyautomation.github.io/Traffic_Camera_Pipeline/`.

Bibliography

[1] Luca Bertinetto et al. “Fully-convolutional siamese networks for object tracking”. In: *European conference on computer vision*. Springer. 2016, pp. 850–865.

[2] Raffaele Carli et al. “Automated evaluation of urban traffic congestion using bus as a probe”. In: *CASE*. IEEE. 2015, pp. 967–972.

[3] Moohyun Cha, Jeongsam Yang, and Soonhung Han. “An interactive data-driven driving simulator using motion blending”. In: *Computers in Industry* 59.5 (2008), pp. 520–531.

[4] Kang-Ching Chu et al. “Validation of stochastic traffic flow model with microscopic traffic simulation”. In: *CASE, 2011 IEEE Conference on*. IEEE. 2011, pp. 672–677.

[5] Carl De Boor et al. *A practical guide to splines*. Vol. 27. Springer-Verlag New York, 1978.

[6] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *arXiv preprint arXiv:1711.03938* (2017).

[7] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*.

[8] Daniel Gordon, Ali Farhadi, and Dieter Fox. “Re3 : Real-Time Recurrent Regression Networks for Object Tracking”. In: *CoRR* abs/1705.06368 (2017). arXiv: 1705.06368. URL: http://arxiv.org/abs/1705.06368.

[9] Daniel Gordon, Ali Farhadi, and Dieter Fox. “Re3 : Real-Time Recurrent Regression Networks for Object Tracking”. In: *CoRR* abs/1705.06368 (2017). arXiv: 1705.06368.

[10] David Held, Sebastian Thrun, and Silvio Savarese. “Learning to track at 100 fps with deep regression networks”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 749–765.

[11] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[12] Weiming Hu et al. “A survey on visual surveillance of object motion and behaviors”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 34.3 (2004), pp. 334–352.

[13] Jonathan Huang et al. “Speed/accuracy trade-offs for modern convolutional object detectors”. In: *IEEE CVPR*. 2017.

[14] Dieter Koller et al. “Towards robust automatic traffic scene analysis in real-time”. In: *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*. Vol. 1. IEEE. 1994, pp. 126–131.

[15] Matej Kristan et al. “The visual object tracking vot2013 challenge results”. In: *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE. 2013, pp. 98–111.

[16] Steven M LaValle and James J Kuffner Jr. “Rapidly-exploring random trees: Progress and prospects”. In: (2000).

[17] Matthew J Leotta and Joseph L Mundy. “Vehicle surveillance with a generic, adaptive, 3D vehicle model”. In: *IEEE transactions on pattern analysis and machine intelligence* 33.7 (2011), pp. 1457–1469.

[18] Jesse Levinson et al. “Towards fully autonomous driving: Systems and algorithms”. In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 163–168.

[19] K. C. Li, H. C. Wang, and J. M. Chiu. “Pedestrian tracking system by using human shape prior model”. In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. Aug. 2014, pp. 1139–1143. DOI: 10.1109/CoASE.2014.6899469.

[20] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[21] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[22] Hyeonseob Nam and Bohyung Han. “Learning multi-domain convolutional neural networks for visual tracking”. In: *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. IEEE. 2016, pp. 4293–4302.

[23] Henry YT Ngan, Nelson HC Yung, and Anthony GO Yeh. “Modeling of traffic data characteristics by Dirichlet process mixtures”. In: *CASE*. IEEE. 2012, pp. 224–229.

[24] Juan Carlos Niebles et al. “Extracting moving people from internet videos”. In: *European conference on computer vision*. Springer. 2008, pp. 527–540.

[25] Björn Ommer, Theodor Mader, and Joachim M Buhmann. “Seeing the objects behind the dots: Recognition in videos from a moving camera”. In: *International journal of computer vision* 83.1 (2009), pp. 57–71.

[26] Alessandro Prest et al. “Learning object class detectors from weakly annotated video”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 3282–3289.

[27] Shaoqing Ren et al. “Faster R-CNN: towards real-time object detection with region proposal networks”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.6 (2017), pp. 1137–1149.

[28] Christian Robert. *Machine learning, a probabilistic perspective*. 2014.

[29] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[30] Robert Sorschag. “A high-level survey of video annotation and retrieval systems”. In: *International Journal of Multimedia Technology* 2.3 (2012), pp. 62–71.

[31] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. "Unsupervised learning of video representations using lstms". In: *International conference on machine learning*. 2015, pp. 843–852.

[32] Christian Szegedy et al. "Going deeper with convolutions". In: Cvpr. 2015.

[33] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[34] Shubham Tulsiani et al. "Multi-view supervision for single-view reconstruction via differentiable ray consistency". In: *CVPR*. Vol. 1. 2. 2017, p. 3.

[35] Adrian Ulges et al. "Learning automatic concept detectors from online video". In: *Computer vision and Image understanding* 114.4 (2010), pp. 429–438.

[36] F. B. Vidal and V. H. C. Alcalde. "Window-Matching Techniques with Kalman Filtering for an Improved Object Visual Tracking". In: *2007 IEEE International Conference on Automation Science and Engineering*. Sept. 2007, pp. 829–834. DOI: 10.1109/COASE.2007.4341822.

[37] Mr K Vishnu and PG Scholar. "A SURVEY ON THE PROPAGATION OF WEB VIDEOS". In: (2014).

[38] Yiyan Wang et al. "Video image vehicle detection system for signaled traffic intersection". In: *Hybrid Intelligent Systems, 2009. HIS'09. Ninth International Conference on*. Vol. 1. IEEE. 2009, pp. 222–227.

[39] Yezhou Yang et al. "Robot Learning Manipulation Action Plans by" Watching" Unconstrained Videos from the World Wide Web." In: *AAAI*. 2015, pp. 3686–3693.