Learning Traffic Behaviors for Simulation via Extraction of Vehicle Trajectories from Online Video Streams

Xinhe Ren*,[1] David Wang*,[1] Michael Laskey,[1] Ken Goldberg[1,2]

***Abstract*— 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 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.**

I. Introduction

Autonomous driving simulators offer the potential to rapidly prototype behavioral algorithms. However, a significant concern with developing in a simulation is how accurately it reflects the physical world.

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 4). Using online video streams for data collection has the potential to capture a significant amount of driver demonstration data. 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 [13] 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. 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 is able to produce models that simulate more plausible driving behaviors.

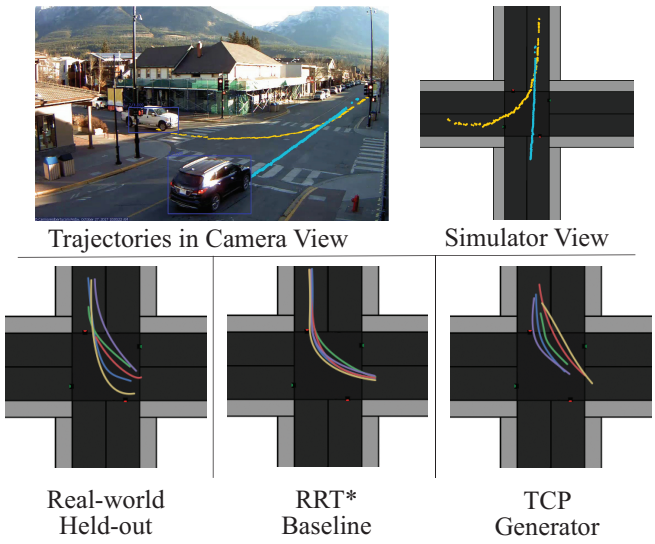

Fig. 1: Example trajectories extracted by TCP: top left image is an illustration of trajectories overlaid onto camera perspective; top right image shows the trajectories in bird's eye view in FLUIDS, a traffic intersection simulator. Bottom row compares three groups of left turn trajectories of vehicles coming from the top of the scene. A real-world held-out set consists of trajectories that real drivers took, but are not used in any training. The middle figure shows trajectories generated by the RRT* [10] algorithm. The right figure shows TCP-generated trajectories sampled from a model trained on collected driving data. We observe that the TCP-generated trajectories better approximate the held-out set.

Summary of Contributions:

1) A novel pipeline for extracting vehicle trajectories from online traffic camera video streams for simulating traffic intersections.
2) A dataset of 234 annotated, minute-long videos with 2618 labeled vehicle trajectories, and 8980 additional unannotated minute-long videos (6+ days).

II. Related Work

A. 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

*Denotes Equal Contribution
[1]Department of Electrical Engineering and Computer Science
[2]Department of Industrial Engineering and Operations Research
[1−2]The AUTOLab at UC Berkeley (automation.berkeley.edu).
{jim.x.ren, dmwang, laskeymd, goldberg}@berkeley.edu
†https://canmorealberta.com/webcams/main-street

used classical machine vision and digital signal processing techniques to produce real-time traffic scene analysis [9]. More recent work has investigated improving static intersection vehicle detection methods with existing video image vehicle detection systems [27] 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 [11]. 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.

B. 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.

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. [14] 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.

C. Inverse Reinforcement Learning for Driving Behaviors

Inverse reinforcement learning (IRL) is a method that uses demonstrations of a task given by an expert to learn a policy that matches the expert's behavior by recovering an underlying cost function. Our work is similar in that we attempt to learn traffic behaviors from driving data observed from a traffic cam stream, although we differ in implementation details in that we do not explicitly learn the cost function. Several driving applications have used IRL to learn behaviors. Abbeel et al. [1] used IRL to learn various driving styles in a highway driving simulation with multiple lanes. Ziebart et al. [29] proposed a maximum entropy approach to IRL, and they demonstrate their algorithm on the problem of learning taxicab drivers' decision-making behaviors in a road network. Sadigh et al. [20] use IRL to learn behaviors for human drivers in a simulation environment with autonomous agents.

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

D. Learning from Online Videos

There have been many instances of learning from online videos, which can be noisy and ill-constrained. Ulges et al. [25] utilized YouTube content for the autonomous training of a video tagger. Prest et al. [17] trains an object detector from weakly annotated videos on the internet using domain adaptation to improve the performance of the detector. In robotics, Yang et al. [28] explored learning robot manipulation tasks (grasping) by processing videos from the World Wide Web. The paper used CNN for object recognition, and action grammar parse tree to interpret the videos' unconstrained semantic structures. Niebles et al. [15] developed a system to learn human motions from YouTube videos. Srivastava et al. [22] use LSTM neural networks to perform unsupervised learning on YouTube videos. Sorschag [21] conducted a survey of video annotation techniques by examining how to collected large amounts of video for machine learning algorithms. Another survey by Vishnu et al. [26] examined how the prevalence of web videos has enabled large-scale learning.

In this paper, 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.

III. Problem Statement

A. Assumptions

We assume access to real-world traffic camera streams.

B. Metrics

We model the learned traffic behaviors as probability distributions, from which we can sample from to generate behaviors for use in simulations. We quantitatively evaluate the ability of the learned model to generate plausible traffic behavior by using the log-likelihood [24]. Denote Θ as the space of parameters for a probability distribution over a set X. For a particular distribution parameterized by some $\theta \in \Theta$ and $x \in X$, the log-likelihood is

$$\mathcal{LL}(x|\theta) = \log[p(x;\theta)],$$

the log of the probability of observing x given the model. We partition the set of trajectories extracted from TCP into a training set and a held-out set. Our goal is to learn a model $\theta_{TCP} \in \Theta$ using the training set that minimizes the negative log-likelihood of observing the held-out data. We compare the learned model to a baseline model $\theta_{BASE} \in \Theta$ used in a simulator of traffic intersections.

C. Objective

We want to learn traffic behaviors based on real data via online video streams on two primary levels: high-level behaviors and trajectories of agent movement. These are common components in the software stack for driving simulators [6]. High-level behaviors capture general actions of vehicles at the intersection: which lane they started on, and what action they took (turn left or right, move forward, etc).

Fig. 2: 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.

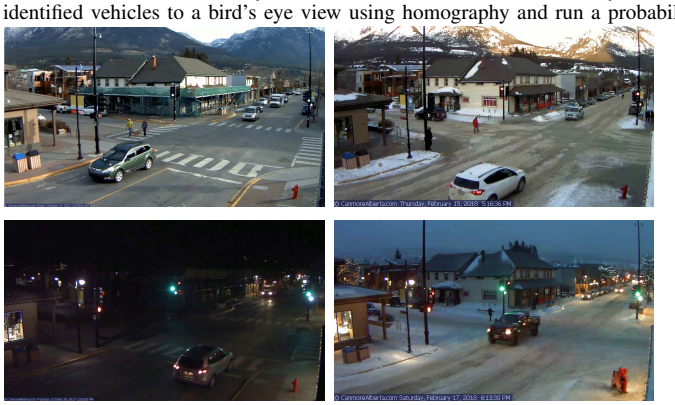

Fig. 3: TCP captures a four-way intersection in Canmore, Alberta at different times of day. It features a variety of lighting conditions, weather, and road conditions. For the following experiments, we only labeled a small subset of the daytime videos.

Trajectories consist of a sequence of Cartesian (x, y) points that capture the path taken. We can then use the learned behaviors to simulate more plausible driving behaviors.

IV. System Architecture

A. Overview

Figure 2 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.

B. Example Intersection

Figure 3 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.

C. Labeling Vehicles

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) [13] and Faster-RCNN [18] have demonstrated surprising performance on many object detection datasets, including PASCAL VOC [7] and COCO [12], 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. See Supplementary Materials for additional details. TCP utilizes the pre-trained SSD network[§] to label vehicles in the collected data with real-time performance.

D. 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 [23] to rotate the camera to a bird's eye viewpoint.

Homography works by estimating a projective matrix that morphs pixel locations from a source domain into a target domain. The target domain, in this case, is the simulator, and the source is the traffic camera view. We estimated the matrix on four pairs of corresponding points between the camera and the simulator views. The points were selected such that the corners of the intersection in the traffic camera matched the corners in the simulator.

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 the vehicle's true point on the road, so we learn a linear mapping from selected point to a corrected point to adjust for the inaccuracy using a hand-labeled dataset of corrections to apply.

E. Extracting Trajectories

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 [8], but we discuss potential ways to automate this step in future work.

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

[§]https://github.com/balancap/SSD-Tensorflow

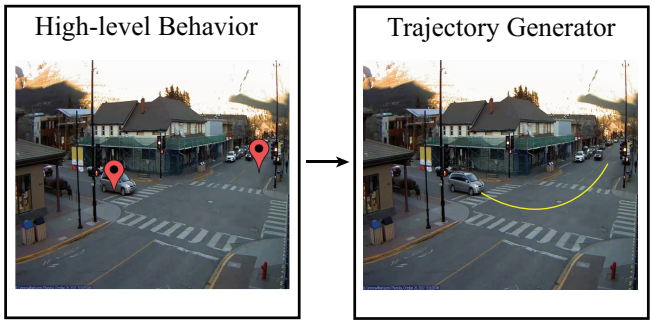

Fig. 4: We can simulate vehicles at a four-way intersection by specifying traffic behaviors in two steps. (Left) First, we choose a starting lane for a vehicle (the west lane in the figure), and an ending lane (north lane). (Right) After the starting and end lanes are chosen, we can specify a trajectory consisting of a sequence of points for the vehicle to traverse.

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.

V. 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 4.

A. 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 IV-E) 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.

B. 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 IV-E 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 [19]:

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

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

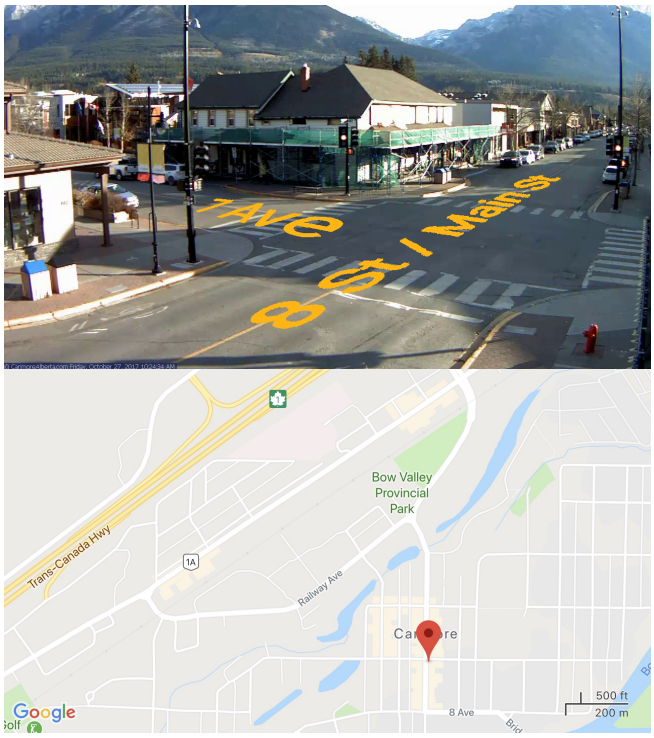

Fig. 5: Google Map view of the intersection in TCP. Top image shows the street names of the intersection. Bottom image shows the surrounding area of the intersection, and the dropped pin shows the location of the intersection.

distributions over the cubic spline fit parameters because they are low dimensional approximations of the trajectories.

VI. 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 5 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.

A. Evaluating Traffic Cam Pipeline

1) ***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.

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

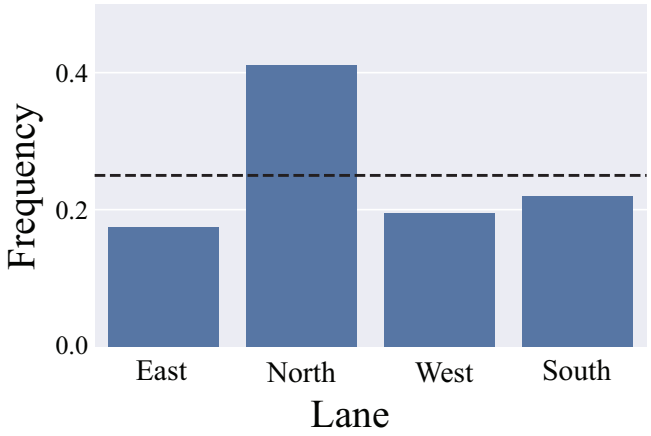

Fig. 6: Distribution of vehicles by starting lane. The cleaned up data contains 1872 vehicle trajectories. The dotted lines show the baseline uniform distributions.

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.

2) ***Time Efficiency****:* Table I contains the average time it takes at each stage of TCP to process a minute-long video clip. SSD [13], 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 [18]. SSD inference performance is heavily dependent on the computing hardware. Table I shows that a Nvidia Titan Xp GPU can accomplish the object detection task in real-time, but a Nvidia Tesla K40 GPU struggles.

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

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[16], [17]. Automating this stage is discussed in future work. In comparison, homography transformation and trajectory extraction are very efficient; all of the computations for a minute-long video clip take one second on average.

B. Learning High-level Behaviors

1) ***Start Location****:* We use TCP to estimate the vehicle start location distribution. As a comparison, we use the FLUIDS 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 6 shows frequencies at which vehicles appear from each of the four cardinal directions. The data shows that

Fig. 7: Distribution of vehicle trajectory primitives at each cardinal direction. The dotted lines show the default uniform distribution.

significantly more vehicles come from the north of the intersection. The dashed line in Figure 6 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 II.

*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 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 5 and 6 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 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 II.

*3) **Vehicle Arrival**:* We analyze when vehicles appear in the intersection. Figure 8 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.

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

TABLE II: 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.

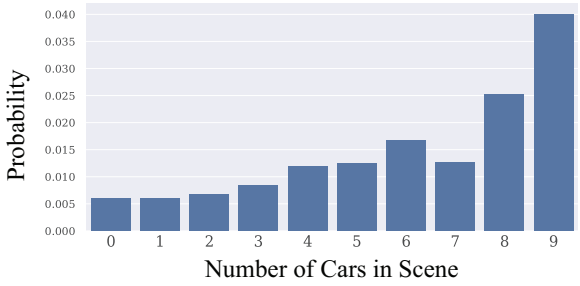

Fig. 8: Probability of new vehicle appearing given the number of vehicles in current frame.

C. Trajectory Generator

We use RRT* [10] 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 III. 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 9 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 III: 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.

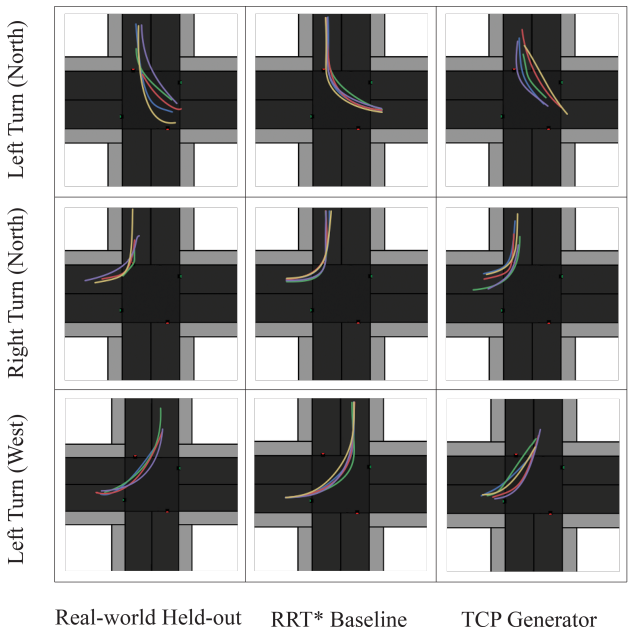

Fig. 9: Examples of held-out trajectories, trajectories sampled from the baseline trajectory generator, and trajectories sampled from the learned TCP generative model. We show five examples each for three primitive behaviors: left turn from the north, right turn from the north, and left turn from the 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.

VII. Discussion and Future Work

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.

A. 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.

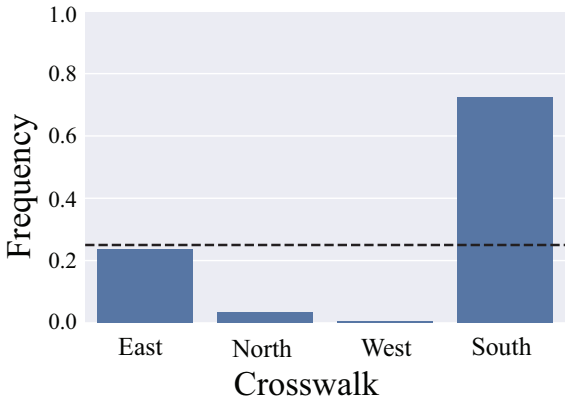

Fig. 10: Distribution of pedestrians by crosswalk location. The cleaned up data contains 209 pedestrian trajectories. The dotted lines show the default uniform distribution.

B. Towards Real-Time and Higher Robustness

Figure 2 and Table I show that the efficiency bottleneck of TCP is the manual initial state labeling. One potential solution to is the Re^3 algorithm by Gordon et al. [8]. They propose a deep neural network architecture that is part convolutional neural network for video frame processing and part recurrent neural network for temporal information handling. It can achieve real-time object tracking at 150 FPS without having to learn object classification explicitly. In future work, we will experiment with replacing manual initial state labeling and trajectory filtering with this new tracking algorithm, while using SSD object detection to find initial appearances of objects. We hope this will allow TCP to scale up its data-collecting ability and extend more easily to different intersections.

C. 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 10. 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/`.

VIII. Acknowledgments

This research was performed at the AUTOLab at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. The authors were supported in part by donations from Siemens, Google, Honda, Intel, Comcast, Cisco, Autodesk, Amazon Robotics, Toyota Research Institute, ABB, Samsung, Knapp, and Loccioni. Any opinions, findings, and conclusions or recommendations expressed in this material

are those of the author(s) and do not necessarily reflect the views of the Sponsors. We thank our colleagues who provided helpful feedback and suggestions, in particular Ajay Tanwani, Carolyn Chen, Christopher Powers, Daniel Seita, Jacky Liang, Jeff Mahler, Matthew Matl, Sanjay Krishnan, and Sequoia Eyzaguirre.

REFERENCES

[1] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[2] R. Carli, M. Dotoli, N. Epicoco, B. Angelico, and A. Vinciullo, "Automated evaluation of urban traffic congestion using bus as a probe," in *CASE*. IEEE, 2015, pp. 967–972.

[3] M. Cha, J. Yang, and S. Han, "An interactive data-driven driving simulator using motion blending," *Computers in Industry*, vol. 59, no. 5, pp. 520–531, 2008.

[4] K.-C. Chu, L. Yang, R. Saigal, and K. Saitou, "Validation of stochastic traffic flow model with microscopic traffic simulation," in *CASE, 2011 IEEE Conference on*. IEEE, 2011, pp. 672–677.

[5] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor, *A practical guide to splines*. Springer-Verlag New York, 1978, vol. 27.

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results."

[8] D. Gordon, A. Farhadi, and D. Fox, "Re3 : Real-time recurrent regression networks for object tracking," *CoRR*, vol. abs/1705.06368, 2017.

[9] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "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.

[10] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 163–168.

[12] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] H. Y. Ngan, N. H. Yung, and A. G. Yeh, "Modeling of traffic data characteristics by dirichlet process mixtures," in *CASE*. IEEE, 2012, pp. 224–229.

[15] J. C. Niebles, B. Han, A. Ferencz, and L. Fei-Fei, "Extracting moving people from internet videos," in *European conference on computer vision*. Springer, 2008, pp. 527–540.

[16] B. Ommer, T. Mader, and J. M. Buhmann, "Seeing the objects behind the dots: Recognition in videos from a moving camera," *International journal of computer vision*, vol. 83, no. 1, pp. 57–71, 2009.

[17] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari, "Learning object class detectors from weakly annotated video," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3282–3289.

[18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[19] C. Robert, "Machine learning, a probabilistic perspective," 2014.

[20] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions." in *Robotics: Science and Systems*, 2016.

[21] R. Sorschag, "A high-level survey of video annotation and retrieval systems," *International Journal of Multimedia Technology*, vol. 2, no. 3, pp. 62–71, 2012.

[22] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, 2015, pp. 843–852.

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

[24] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," *arXiv preprint arXiv:1511.01844*, 2015.

[25] A. Ulges, C. Schulze, M. Koch, and T. M. Breuel, "Learning automatic concept detectors from online video," *Computer vision and Image understanding*, vol. 114, no. 4, pp. 429–438, 2010.

[26] M. K. Vishnu and P. Scholar, "A survey on the propagation of web videos," 2014.

[27] Y. Wang, Y. Zou, H. Shi, and H. Zhao, "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.

[28] Y. Yang, Y. Li, C. Fermüller, and Y. Aloimonos, "Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web." in *AAAI*, 2015, pp. 3686–3693.

[29] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.