2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)
Munich, Germany, August 20-24, 2018

Learning Traffic Behaviors by Extracting 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, we propose a framework using online public traffic cam video streams. We implement the Traffic Camera Pipeline (TCP), a system that 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. We benchmarked several deep learning detectors for the task of vehicle detection: SSD-VGGNet, SSD-InceptionNet, and SSD-MobileNet, and Faster-RCNN; we found that SSD-VGGNet had the highest precision and quality of bounding boxes. We captured four hours of video streams and used it to train generative models describing both the starting and ending positions of vehicles as well as the trajectories of points traversed. We find that the negative log-likelihood of held-out real data is more likely to occur under the learned models compared to baseline models used in a traffic intersection simulator. The extracted dataset of 234 annotated minute-long videos containing 2618 labeled vehicle trajectories and 8980 additional unannotated minute-long videos is available at https://berkeleyautomation.github.io/Traffic_Camera_Pipeline/.**

I. Introduction

Online video streams (e.g., a YouTube stream of a four-way traffic intersection at 7 Ave and Main St in Canmore, Alberta) can be a valuable source for real-world driving data.

We propose Traffic Camera Pipeline (TCP), which applies recent deep learning advances in object detection [23] to detect and track vehicles in the video stream. We use homography to register the located vehicles to the corresponding positions in the intersection. 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 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. A visualization of the learned trajectories is shown in Fig. 1. We compare the learned models of traffic behaviors against those used in a traffic intersection simulator, and we find that a held-out set of TCP-collected data has a lower negative log-likelihood of occurring under the models learned from data extracted by TCP, suggesting that TCP is able to produce models that simulate more plausible driving behaviors.

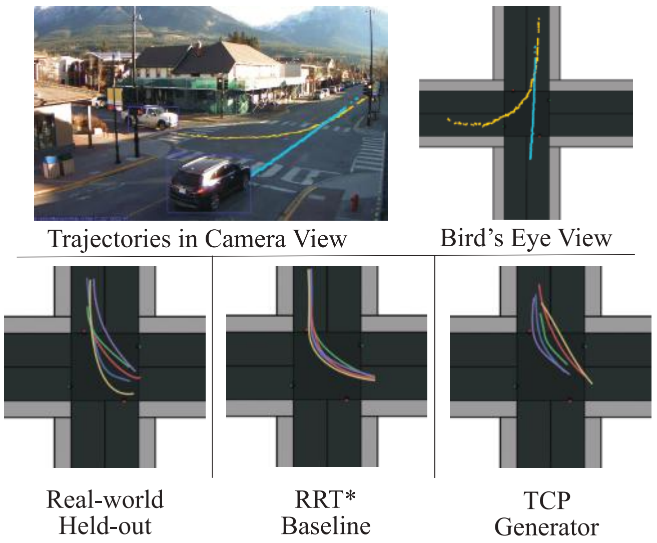

Fig. 1: TCP (Traffic Camera Pipeline) uses recent advances in deep learning to extract driving behaviors from online video streams. Example trajectories extracted by TCP: top left image is an illustration of trajectories overlaid onto camera perspective; top right image shows the trajectories in bird's 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* [17] algorithm without access to real driving data. The right figure shows TCP-generated trajectories sampled from a model trained on collected driving data. We observe that the held-out set is more likely to occur under the TCP model.

Summary of Contributions:

1) A novel pipeline for extracting vehicle trajectories from online traffic intersection camera video streams.
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

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 existing detection systems [39] 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 [20]. Another approach is to equip

*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

vehicles with GPS sensors and then analyze the decision-making data afterward to examine traffic congestion in urban settings [3]. With the rising popularity of deep convolutional neural networks, it is interesting to explore collecting traffic behavior through online 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 [7], 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 [41] is an open-source light-weight Python-based traffic intersection simulator intended for easily customizable extensions. We apply TCP to FLUIDS.

Data-Driven Simulation Improvement Cha et al. [4] 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 samples from drivers on real roads to build a database of primitives. The simulator is then able to produce realistic motions in response to user inputs. Chu et al. [5] analyze traffic at a highway using inductive loop detectors and use the observations to build a traffic flow model. Ngan et al. [25] take traffic videos and use the data to develop a model for traffic behaviors such as vehicle speeds and queue lengths. They also use their model for outlier detection. TCP also learns traffic behavior models in a data-driven approach, emphasizing how deep learning can be helpful in this domain.

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. [42] 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. [30] use IRL to learn behaviors for human drivers in a simulation environment with autonomous agents.

D. Learning from Online Videos

There have been many instances of learning tasks from online videos, which can be noisy and ill-constrained. Ulges et al. [37] utilized YouTube content for the autonomous training of a video tagger. Prest et al. [27] 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. [40] 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. [26] developed a system to learn human motions from YouTube videos. Srivastava et al. [33] use LSTM neural networks to perform unsupervised learning on YouTube videos. Sorschag [32] conducted a survey of video annotation techniques by examining how to collected large amounts of video for machine learning algorithms. Videos have also been used in learning from demonstration tasks in robotics such as grasping in clutter [19], [18] and needle insertion [15].

E. 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. 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. [38] applied a Kalman filter to a feature based object tracking algorithm, greatly improving the performance. More recently, Li et al. [21] developed a model based pedestrian tracking system with human segmentation. The system applies a human model prior as a seed for segmentation.

Recently, there have been many deep learning based approaches to object tracking. In 2005, MD-Net [24] became the first deep learning method to win The Visual Object Tracking challenge (VOT) [16]. 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. [2] 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 experiments, we use a pre-trained Re3 network to track vehicles in videos.

III. Problem Statement

A. Assumptions

We assume access to real-world fixed traffic camera streams with a full, unoccluded view of the traffic intersection.

B. 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 [7]. 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.). Trajectories consist of a sequence of Cartesian (x, y) points that capture the trajectory over time.

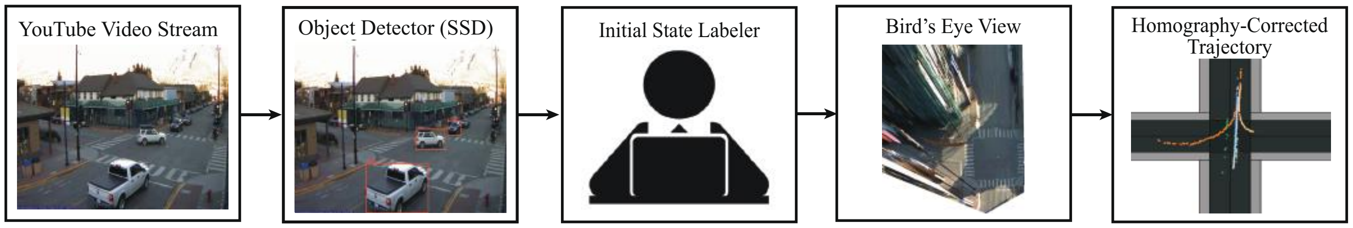

Fig. 2: TCP system architecture (excluding learning and analysis). First, we capture a video stream of a traffic intersection and use SSD, a deep object detection network, to identify and label vehicles. A human then supplies labels for when vehicles first appear at the intersection. Finally, we map the identified and tracked vehicles to a bird's eye view using homography and extract trajectories.

C. Metrics

We model learned traffic behaviors as probability distributions, from which we can sample from to generate behaviors for use in simulations. For the high-level behaviors, we use discrete multinomial distributions, and for trajctories, we use multivariate Gaussian distributions over cubic spline-fit parameters. We quantitatively evaluate the ability of the learned model to generate plausible traffic behavior by using the log-likelihood [36]. 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

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

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.

IV. System Architecture

A. Overview

Figure 2 shows an overview of TCP, which has five main steps:

1) Collect traffic video.
2) Detect vehicles using deep neural networks.
3) Manually label when vehicles first appear at the intersection.
4) Transform camera view to a bird's eye view.
5) Extract vehicle trajectories using a probabilistic filtering algorithm.

B. Example Intersection

Figure 3 illustrates the four-way intersection in Canmore. We downloaded footage from the traffic cam video stream over a week in October 2017 for further processing. The videos are in 720p resolution.

C. Vehicle Detection

Advances in deep learning have significantly improved perception and object detection in video. There are many deep network architectures for object detection, such as Single Shot Detector (SSD) [23] and Faster-RCNN [28], that have demonstrated surprising performance on many object detection datasets including PASCAL VOC [8] and COCO [22]. SSD is a network architecture that combines a feature extractor with additional layers that perform object detection at various scales. We compare several feature extractors: VGGNet [31], InceptionNet [34], 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.

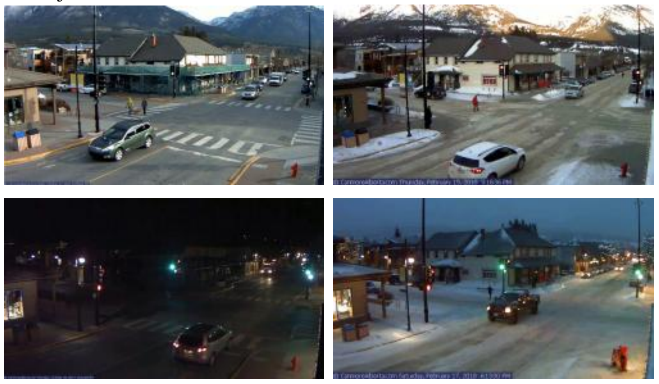
Fig. 3: TCP captures a four-way intersection in Canmore, Alberta at different times of day. It features a variety of lighting conditions, weather, and road conditions. For the following experiments, we labeled four hours of the daytime videos.

To benchmark these object detection networks for TCP, we hand-labeled held-out a dataset 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:

- 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 SSD 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.

The results are shown in Table I. The recall values are low due to the large number of objects in the dataset. The

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

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

Network	Precision	Recall	Average IoU
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 I: 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 label. The dataset contains a total of 718 cars.

images are of a busy intersection with many cars that are close together, often partially occluding each other. Also, there are cars at farther distances that are difficult for any network to detect. Low recall is tolerable in our system, since we also employ a filtering scheme to detect the cars over many frames. Thus, the system has robustness to not being detected at every timestep. In light of this, we use SSD-VGGNet because of its higher precision and better bounding box quality on our dataset, which is important for retrieving the pose of the car.

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 [35] to create 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.

Given a bounding box, we still need to specify a point from the bounding box that corresponds to where the vehicle is centered 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 [9], but we discuss potential ways to automate this step in section IV-F.

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 [6] 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.

F. Vehicle Tracking

As an alternative to the human labeling step, we explore an end-to-end deep learning method for object tracking: Real-Time Recurrent Regression Network (Re3) [9]. Re3 is a deep neural network that combines the image processing advantages of a CNN with the temporal data processing power of an RNN. This is a fully autonomous procedure that processes videos. In this study, we use a pre-trained model of Re3[††].

We use SSD to generate and supply potential initial bounding boxes and Re3 to accomplish the tracking. Algorithm 1 details 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 intersection over union (IoU) metric: if two bounding boxes have an IoU greater than 0.7, an empirically chosen threshold, then we consider the two bounding boxes "matching". This new pipeline is illustrated in Figure 4.

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

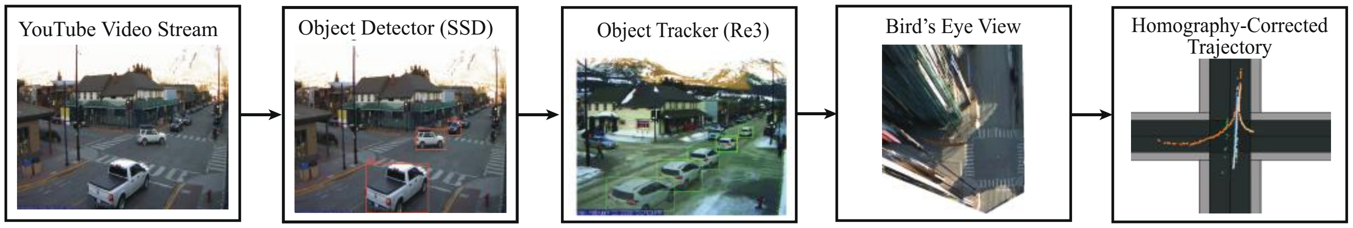

Fig. 4: Fully automated version of the TCP system architecture. Instead of having a manual labeling step, we use another deep neural network, Re3, for object tracking.

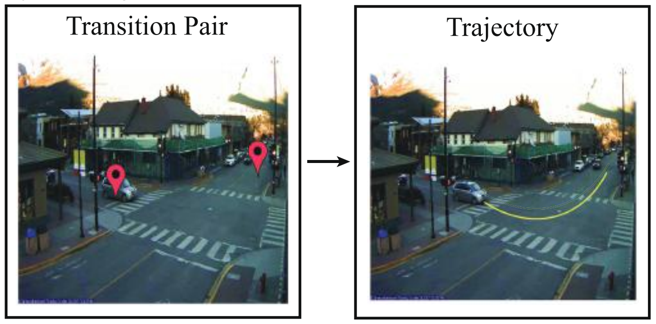

Fig. 5: We can simulate vehicles at a four-way intersection by specifying traffic behaviors in two steps. (Left) First, we compute a starting lane for a vehicle (the west lane in the figure), and an ending lane (north lane). (Right) After the starting and end lanes are chosen, we compute a trajectory consisting of a sequence of points.

Algorithm 1 Tracking with SSD and Re3

```
$T \leftarrow []$
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 IoU(b, s) < 0.7 for all s in $B_{SSD}$ then
            $T$.add(Re3.track($f$, $b$))
        if $b$ outside intersection then
            $T$.remove(Re3.track($f$, $b$))
```

V. Learning Driving Behaviors for Simulation

We consider behaviors on two levels, as shown in Figure 5.

A. Start, End Transitions

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 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 [29]:

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

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

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 $-\log \mathcal{L}(\mu_{TCP}, \Sigma_{TCP}|\varphi)$. 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.

VI. Experiments

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[§].

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

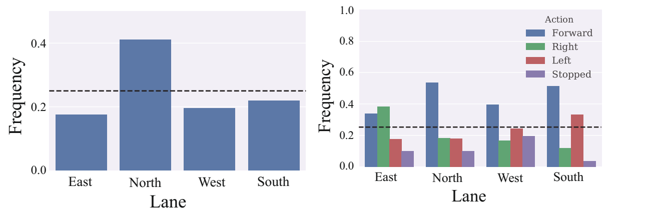

Fig. 6: The left figure shows the distribution of vehicles by starting lane. The cleaned up data contains 1872 vehicle trajectories. The right figure shows distribution of vehicle trajectory primitives at each cardinal direction. The dotted lines show the default uniform distribution

For context, Canmore, Canada, is 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 at a busy commercial district.

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. 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.

2) ***Time Efficiency:*** Table II contains the average time it takes at each stage of TCP to process a minute-long video clip. SSD [23] is a real-time deep convolutional neural network running in TensorFlow. SSD inference performance is heavily dependent on the computing hardware. Table II shows that a Nvidia Titan Xp GPU can accomplish the object detection task in or close to 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
Re3 Tracking (Nvidia Titan Xp)	**23.60**	8.59
Re3 Tracking (Nvidia Titan X pascal)	38.08	14.21
Homography & Trajectory Extraction	1.01	0.45

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

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 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 III.

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 6 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. Figure 6 explains that the north direction is more popular due to the abundance of businesses and connectivity to a major national highway. This pattern 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 6 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 III.

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 III: 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.

3) ***Vehicle Arrival:*** We analyze when vehicles appear in the intersection. Figure 7 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.

C. Trajectory Generator

We use RRT* [17] as the baseline trajectory generator for comparison, the implementation in FLUIDS: given a start and end position in the intersection, the RRT* algorithm

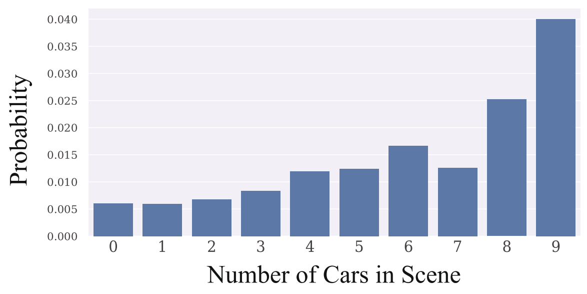

Fig. 7: Probability of new vehicle appearing given the number of vehicles in current frame.

attempts to minimize distance traveled, while obeying traffic rule restrictions such as lane directions and road boundaries.

In order to compare to our baseline, we sample 227 trajectories from the planner and then fit a cubic-spline fit to each trajectory. We then learn a generative model by fitting a multivariate Gaussian distribution over the parameters of the cubic-spline fit for 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 IV. 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 8 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, or start and end states. 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 IV: 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 that describe pairs of starting and ending positions.

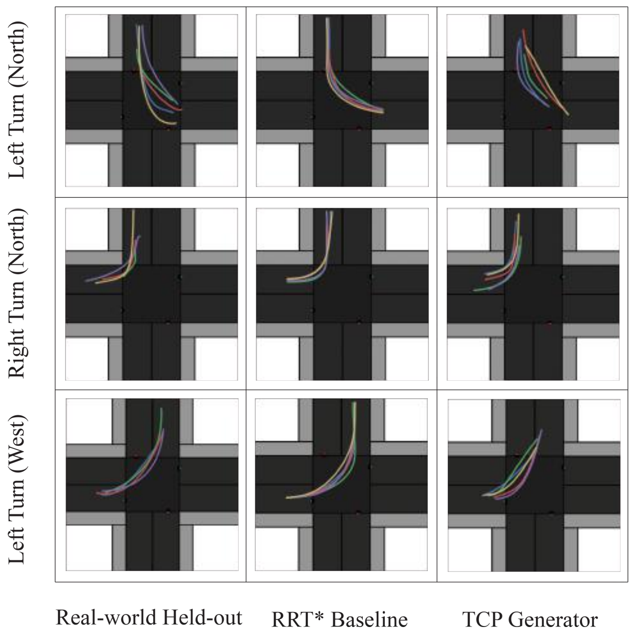

Fig. 8: Examples of held-out trajectories, trajectories sampled from the baseline trajectory generator, and trajectories sampled from the learned TCP generative model. We show five examples each for three primitive behaviors: left turn from the north, right turn from the north, and left turn from the 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 online public video streams.

A. Evaluation

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.

B. Pedestrians

We hope to expand our pipeline to include pedestrians in the next version: TCP-IP (Traffic Camera Pipeline - Including Pedestrians).Since pedestrians are much smaller than vehicles, they are much harder for SSD to robustly detect and track.

C. Velocity

We also wish to be able to extract velocity data of vehicles in video streams. We have attempted this, but the variance in the quality of the bounding box for a particular vehicle over time causes the extracted velocity to oscillate, which makes it difficult to get a smooth and consistent velocity reading over time.

For more details and to download the data and code, 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, Berkeley DeepDrive (BDD), 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] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 749–765.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] W. Hu, T. Tan, L. Wang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 3, pp. 334–352, 2004.

[13] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE CVPR*, 2017.

[14] 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.

[15] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, "Ddco: Discovery of deep continuous options for robot learning from demonstrations," in *Conference on Robot Learning*, 2017, pp. 418–437.

[16] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Cehovin, G. Nebehay, G. Fernandez, T. Vojir, A. Gatt *et al.*, "The visual object tracking vot2013 challenge results," in *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE, 2013, pp. 98–111.

[17] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[18] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg, "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 358–365.

[19] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE, 2016, pp. 827–834.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] H. Nam and B. 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] 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.

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

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] H. Zhao, A. Cui, S. A. Cullen, B. Paden, M. Laskey, and K. Goldberg, "Fluids: A first-order local urban intersection driving simulator," in *CASE*. IEEE, 2018.

[42] 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.