FLUIDS: A First-Order Lightweight Urban Intersection Driving Simulator

Hankun Zhao*[1], Andrew Cui*[1], Schuyler A. Cullen[3], Brian Paden[3], Michael Laskey[1], Ken Goldberg[1,2]

***Abstract*— To facilitate automation of urban driving, we present an efficient, lightweight, open-source, first-order simulator with associated graphical display and algorithmic supervisors. FLUIDS can efficiently simulate traffic intersections with varying state configurations for the training and evaluation of learning algorithms. FLUIDS supports an image-based birds-eye state space and a lower dimensional quasi-LIDAR representation. FLUIDS additionally provides algorithmic supervisors for simulating realistic behavior of pedestrians and cars in the environment. FLUIDS generates data in parallel at 4000 state-action pairs per minute and evaluates in parallel an imitation learned policy at 20K evaluations per minute. A velocity controller for avoiding collisions and obeying traffic laws using imitation learning was learned from demonstration. We additionally demonstrate the flexibility of FLUIDS by reporting an extensive sensitivity analysis of the learned model to simulation parameters. FLUIDS 1.0 is available at `https://berkeleyautomation.github.io/Urban_Driving_Simulator/`.**

I. Introduction

Several end-to-end driving simulators have been designed to prototype algorithms for autonomous cars. Most simulators focus on providing photo-realistic "drivers-eye" perspective renderings of the physical world and modeling accurate car dynamics through physics simulation [29][6][34]. While these simulators are ideal for testing the perception and control software stack of an autonomous car, they are not efficient for early prototyping of learning algorithms.

To provide large-scale data generation and rapid evaluation, we present FLUIDS (First-Order Lightweight Urban Intersection Driving Simulator). FLUIDS is a flexible and efficient simulator for multi-agent driving and pedestrian behaviors at intersections. FLUIDS provide the ability to test both traditional planning algorithms and model-free learning agents in a large number of traffic conditions. As shown in Fig. 1, it simulates intersections where cars must coordinate to achieve safe, collision-free behavior. FLUIDS is designed to be configurable to new types of traffic intersections using an API that allows for user-designed N-way intersections.

FLUIDS provides access to three different types of state representations. The first is a explicit representation, which gives the agents access to the full state of the world. The second is a bird's-eye perspective, which is a RGB image. The third is a quasi-LIDAR state featurization, which represents data from LIDAR and camera sensors. For both the quasi-LIDAR representation and the birds-eye view, FLUIDS provides a parameterized model of sensor noise.

*Denotes Equal Contribution

[1]Department of Electrical Engineering and Computer Science
[2]Department of Industrial Engineering and Operations Research
[3]Samsung Strategy and Innovation Center
[1−2]The AUTOLAB at UC Berkeley; Berkeley, CA 94720, USA
jerryz123@berkeley.edu , andycui97@berkeley.edu, laskeymd@berkeley.edu, goldberg@berkeley.edu

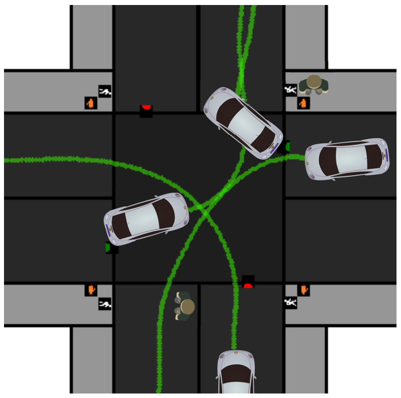
Fig. 1: An example of a driving intersection in FLUIDS showing four cars, two pedestrians, and traffic lights. FLUIDS focuses on efficiently modeling the interaction between different drivers and pedestrians. The colored lines represent paths generated by FLUIDS with collision-free velocities.

FLUIDS offers control spaces at different hierarchical levels. The top level dictates the behavioral logic of the scene, which specifies the starting location and destination for each agent. Another control space is the Cartesian path of (x, y) points for each agent to follow to its target goal state. The third control space is the target velocities to set for the agent to avoid collisions and obey traffic laws. A fourth level in the controls hierarchy is steering angle and acceleration.

This paper contributes:

1) An open-source Python-based simulator for multi-driver and pedestrian behavioral modeling at traffic intersections.
2) Experiments exploring how FLUIDS can be used to collect training data and evaluate driving algorithms.

II. Related Work

Existing driving simulators fall into one of two categories. End-to-end simulators provide environments which simulate both the perception and controls components of the autonomous vehicle stack, while first-order simulators offer a more focused, application-specific environment.

A. Realistic End-to-End Simulators

DeepGTAV [29] is a plugin for the open-world urban driving game Grand Theft Auto V that is used to collect data for autonomous driving [26]. This module provides a TCP interface to the driving portion of the game. DeepGTAV offers a detailed and realistic "drivers-eye viewpoint". However, the game engine backend prevents the user from running faster than real time, or extensively customized simulations. Furthermore, the behavior of non-player agents is not configurable.

CARLA [6] is a recent open-source 3D driving simulator. CARLA provides a high fidelity environment with varied weather, lighting, and pedestrian effects. A cost function drives pedestrian behavior, while a reactive finite state machine drives vehicle planning. Training agents in CARLA is challenging and data intensive, since the simulator is built on the computationally intensive Unreal Engine.

TORCS (The Open Car Racing Simulator) [34] is an open-source, highly configurable driving simulator, and has an extensive library of supporting software. However, TORCS models vehicle racing conditions only.

CarMaker accurately models car dynamics to test new vehicle models [19]. Unfortunately, CarMaker is not open-source, which leads to difficulty in collecting data and verifying experiments.

B. *Lightweight Simulators*

Less realistic, lightweight simulators can be used to develop and evaluate reinforcement and imitation learning algorithms. One example is OpenAI Gym's [4] set of simulation environments. These environments and their flexible API encourage rapid prototyping and parallel simulations. OpenAI Gym provides continuous control benchmarks like Mujoco, [32], a physics engine with a wide variety of simulated environments used to benchmark reinforcement learning algorithms [7].

In the driving domain, researchers have used first-order, application-specific simulators. These include a computational framework for developing traffic controllers called Flow [33], a simulator to study inter-vehicle reactions in driving [30], and several simulators to study driving behaviors in highway environments [17][9]. OpenAI Gym also provides a driving environment implemented using Box2D that simulates a car driving around a racetrack [5].

FLUIDS fills a gap by providing a lightweight, efficient, first-order, configurable driving simulator to model multi-agent behaviors at intersections.

C. *Autonomous Vehicle Planning*

The controls stack of an autonomous vehicle can be decomposed into four components [18]. The route planning and behavioral layers select a high-level goal position for the motion planning and positional control layers to follow. In a dynamic urban environment, the behavioral layer must consider not only the static features of the environment but also the dynamic actions of other vehicles and pedestrians. For this task, a predictive planner which estimates the behavior of other agents is appropriate [35][10].

While predictive planners can generate safe behaviors in dynamic environments, they do not scale well to scenarios with large numbers of agents Although it is possible to solve multi-agent planning as a global optimization problem in polynomial time, such an approach struggles when run in simulation with an unpredictable trial agent [22][12].

III. System Design

A. *Assumptions*

We assume all motion in a driving environment occurs in a 2D plane. For evaluating agent behavior, FLUIDS assumes that users of the simulator have access to an agent, which can accept a state of the world and return a control for the current time step. The agent can operate on either a bird's eye image representation of the world or a lower dimensional state, which specifies car and pedestrian positions and velocities. Agents provide a control in one of three representations: a Cartesian target trajectory of (x, y) points, a target velocity, or an acceleration and steering angle.

B. *Terminology*

Intersection: A layout of sidewalks, lanes, terrain, and traffic signals which govern when and where cars and pedestrians may travel.

Agent: A car or pedestrian which receives a state representation, and responds with an action.

State: The collection of all simulated objects in the environment, including their position, orientation, and velocity.

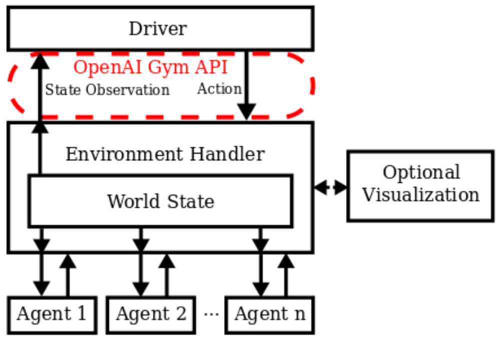

Fig. 2: FLUIDS' system architecture. The evaluated driver interfaces to the world via the OpenAI Gym interface. In the back-end, the simulator coordinates the behavior of additional background drivers and pedestrians.

C. *Architecture*

FLUIDS uses OpenAI's Gym API and Python to allow for easy interfacing with popular learning frameworks such as Tensorflow [1], PyTorch [20], and SciKit Learn [21].

FLUIDS can be run using parallel frameworks such as Python's native multiprocessing package for fast data collection and evaluation.

FLUIDS also provides algorithmic supervisors for controlling the background cars and pedestrians in a scene. These background agents are designed to test the ability of a user agent in different multi-agent interactions.

D. *Agent Dynamics*

Agents fall in three classes: pedestrians, cars, and signal lights.

1) Pedestrian Dynamics: A pedestrian is parameterized by the state space $\mathbf{x}_p(t) = [x, y, \theta, v]$, which corresponds to its positional coordinates, orientation and velocity. The control signal for a pedestrian at time t is specified as $\mathbf{u}_c(t) = [\psi, a]$, which refers to walking direction and acceleration.

For pedestrian dynamics FLUIDS uses a point model to reflect the omni-directional nature of pedestrian movement.

$$\dot{x} = v\cos(\psi) \tag{1}$$

$$\dot{y} = v\sin(\psi) \tag{2}$$

$$\dot{v} = a \tag{3}$$

$$\dot{\theta} = \psi \tag{4}$$

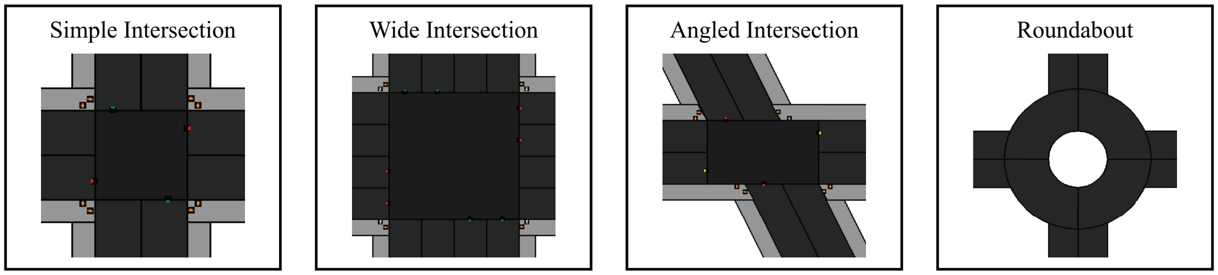

Fig. 3: A sample of the different types of traffic intersections that can be designed in FLUIDS. Intersection types include common intersection like a 4-way stop and more intricate ones such as a roundabout. Intersections can also contain sidewalks, traffic lights, and crosswalk lights.

2) Car Dynamics: The cars in FLUIDS are configurable in size, mass, and maximum velocity. The dynamic state of each car in FLUIDS is described by the state space $\mathbf{x}_c(t) = [x, y, \theta, v]$, which corresponds to positional coordinates, orientation and velocity. The control signal for each car at time t is specified as $\mathbf{u}_c(t) = [\psi, F]$, which refers to steering angle and acceleration force.

For cars, FLUIDS uses the kinematic bicycle model [23]. The differential equations that describe the model are as follows.

$$\dot{x} = v\cos(\theta + \beta) \tag{5}$$

$$\dot{y} = v\sin(\theta + \beta) \tag{6}$$

$$\dot{v} = F/m \tag{7}$$

$$\dot{\theta} = \frac{v}{l_r}\sin(\beta) \tag{8}$$

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r}\tan(\psi)\right) \tag{9}$$

This model defines β as the angle between the current velocity of the center of mass with respect to the longitudinal axis of the car. l_r and l_f are the distance from the center of the mass of the vehicle to the front and rear axles. The control force F is scaled by the mass of the car m to provide control over the car's acceleration.

3) Traffic Lights and Crosswalk Lights: Traffic and crosswalk lights can be set up at the end of each lane or sidewalk. Traffic lights switch among the standard three colors: red, yellow, and green. Crosswalk lights are either white or red. Cars are aware of traffic light states and pedestrians are aware of crosswalk light states. FLUIDS allows for both types of lights to loop, stopping for any specified time on each color. This allows for a wide variety of light synchronization schemes.

E. Collision Interactions

FLUIDS performs collision checking using the Shapely geometry library [24]. Shapely provides a fast, configurable interface for manipulating and designing two-dimensional polygons. Every object in FLUIDS is associated with a Shapely object that acts as its bounding volume. At every timestep, collision checking is performed and all collisions logged.

F. Designing Intersections

FLUIDS provides the ability to design a variety of intersections. Specifically, FLUIDS supports various static objects including lane, street, and sidewalk objects. Lane objects have an associated direction to prevent cars from driving the wrong way. FLUIDS also uses street objects, on which cars can travel in any direction, where lanes meet up at an intersection. Sidewalks represent areas where only pedestrians can travel.

In addition to static objects, FLUIDS also supports dynamic objects such as cars, pedestrians, and traffic lights. Obstacles represented by impassable terrain can be arranged arbitrarily through the scene, simulating realistic conditions such as parked cars, potholes, and road construction. Cars and pedestrians are generated automatically by lanes and sidewalks respectively, but traffic lights need to be manually placed and synchronized. Fig. 3 shows the variety of intersection types representable in FLUIDS.

An intersection is specified by K lanes where each lane, l, is represented by its width, height, position, and direction. Each intersection may also contain sidewalks with similar representations to lanes. The simulator also consists of N dynamic agents, where $N = N_p + N_c + N_l$. N_p and N_c refer to the number of pedestrian and cars respectively. N_l refers to the number of traffic lights and crosswalk lights in the scene. Lights are considered dynamic objects, as the colors which describe their state space and dictate traffic flow change over time.

IV. STATE SPACES IN FLUIDS

FLUIDS offers three options for accessing the state space of the world; explicit, bird's-eye view, and quasi-LIDAR perspective.

A. Explicit View

In the explicit view the agent has access to the true state space (i.e., x_c and x_p of all N agents in the scene) and the state of all traffic lights and pedestrian crosswalk lights.

B. Bird's-Eye View

In bird's-eye view, a top-down image of the intersection is available to the agent. These images use an RGB color representation of configurable resolution.

Viewpoints from above, such as those from unmanned aerial vehicles [31] or traffic cameras [13] motivate this state space. In this state representation, agents must infer from pixel data the location of other agents in the scene and the current state of traffic lights. This state representation is relevant for potentially evaluating the efficiency of recent advances in Convolutional Neural Networks for planning [15].

FLUIDS provides a sensor noise model for this state representation. Specifically, FLUIDS supports adding a bounded zero-mean Gaussian noise to each pixel value, with variance defined by λ_p.

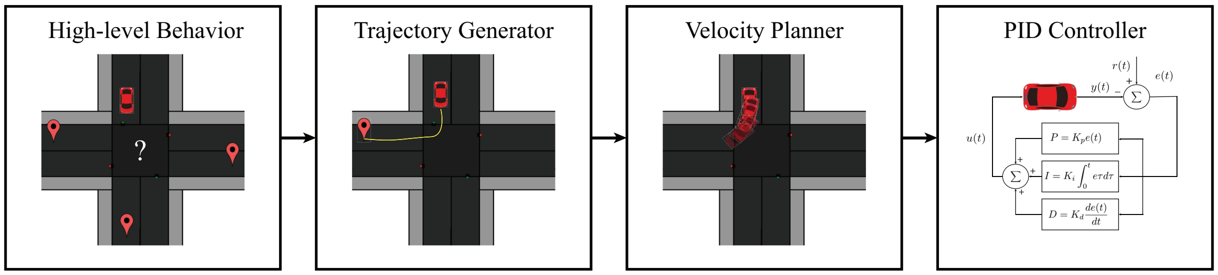

Fig. 4: The planner for the built-in driving agent. First, a high-level destination, specified as a destination lane, is randomly selected for the agent. Then a positional trajectory is generated via OMPL to take the agent to the goal position. A velocity planner is then called to ensure the agent reaches the destination without collisions or violation of traffic laws with respect to the traffic light. Finally, a PID controller is used to execute the planned path.

C. *Quasi-LIDAR Representation*

With recent advances in LIDAR and visual object detection, it is possible for a car to extract relative poses and class labels of other agents adjacent to it [2][8].

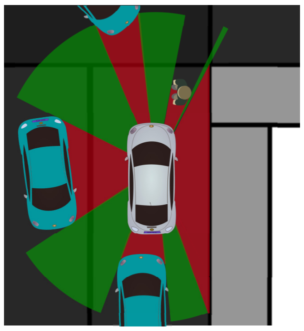

Fig. 5: Illustration of the quasi-LIDAR perspective state space. Rays are projected into the scene from the car's perspective. The red rays correspond to detected objects in the environment. The density of projected rays and noise model are both configurable.

To create the quasi-LIDAR state space m rays are projected from the agent's car at angular intervals along a 360° arc. If a ray collides with an object in the scene, it returns the following information $\{d, \text{label}, \psi_r, v_r\}$, which corresponds to the distance to, the class label of, the relative angle to, and the relative velocity to the object. Fig. 5 shows an illustration of this state space.

FLUIDS models the sensor noise of this state space, via additive element-wise zero-mean Gaussian noise to the relative distance, velocity, and angle. The parameter λ_l specifies the Gaussian noise. In addition to Gaussian noise, FLUIDS also models dropped observations. With probability ϵ a reading from a given ray is returned as empty space, to model the effect of dropout.

V. Hierarchical Control Spaces

FLUIDS supports the ability to test algorithms at four levels of the planning hierarchy: a high-level behavior component, a nominal trajectory generator, a velocity planner, and a PID controller. These four components encompass what is commonly referred to as the high-level behavioral layer, motion planner, and local feedback control components of the autonomous car control stack [18]. Fig. 4 displays how the components of the hierarchy contribute to the total autonomous control of a vehicle.

For each level of the hierarchy, FLUIDS provides an algorithmic supervisor that generates plausible behavior. Additionally, users can operate on one level of the hierarchy and use the supervisor for the others. Users are also able to implement their own controllers at each level of the hierarchy.

A. *Behavioral Logic*

Control Space Each agent in the driving simulator is assigned a target lane. Given K lanes, the control space for the agent is the selection of a target lane (i.e. $\mathcal{U} = \{l_0, ..., l_K\}$).

Algorithmic Supervisor The implemented planner operates by sampling a start lane uniformly and target lane with the following probability distribution.

$$p(l_t|l_g) = \begin{cases} \frac{1}{K-1} & \text{if } l_t \neq l_g \\ 0 & \text{if } l_t = l_g \end{cases},$$

B. *Nominal Trajectory Generator*

Control Space Once any car has an intended goal state, the next level in the hierarchy is to generate a path in Cartesian (x, y) space towards some designated goal state given an initial state representation of the world. Formally, the agent's control space is a set of 2D-positional points (i.e. $U \in \{\mathbb{R} \times \mathbb{R}\}^T$).

Algorithmic Supervisor The implemented planner uses deCastlejau spline interpolation to generate the nominal trajectory. We interpolate a path through a set of waypoints in the scene. Waypoints are located at each end of each lane. The waypoints to hit are selected by the high-level behavioral planner. The same is used for pedestrians.

C. *Velocity Planner*

Control Space Similar to [3], we have an agent that plans the velocity of the trajectory after generation of the nominal trajectory. The agent on this level is given a current state representation of the world and must select the velocity from the bounded range $\mathcal{U} \in [0, v_{max}]$, where v_{max} corresponds to the maximum value.

Algorithmic Supervisor The implemented velocity planner performs a forward projection for each agent for T time-steps in the environment for every possible velocity the car may choose to maintain. Given these projections, the problem of multi-agent pathing is formulated as a constraint satisfaction problem (CSP). Interactions with traffic lights and pedestrians form unitary constraints in this CSP. This CSP is solved with Python-Constraint to assign a collision-free target velocity for every vehicle in the scene [16].

D. *Control Input*

Control Space Given a trajectory of positional (x, y) points and target velocities, we now need to generate controls that can accurately track the trajectory. In this low-level of the hierarchy, the input is steering angle and acceleration, which an agent needs to determine given some current state representation of the world.

Algorithmic Supervisor FLUIDS uses two PID controllers. One controls the steering angle, ψ, to track the direction to the next positional point and the other controls the acceleration

a to the next target velocity. We empirically observe the PID controllers can track over 99% of 200 test trajectories generated.

VI. Bench-marking FLUIDS with Imitation Learning

To evaluate FLUIDS as a platform for developing autonomous vehicle agents at multiple levels, we explore learning a velocity planner. We collect training data from roll-outs of the supervisor on cars interacting in a four-way intersection. We study how quickly the data can be collected, how fast the learned policy can be evaluated, and how sensitive the learned policy is to noise.

In imitation learning (IL), a supervisor provides demonstrations to an agent, and a policy mapping observed states to controls is learned. For our task, the states are the LIDAR view and the learner will operate at the velocity control level of the controls hierarchy. The supervisor will be provided by FLUIDS' algorithmic velocity supervisor.

We formalize this learning as follows: denote a policy as a measurable function $\pi : \mathcal{X} \to \mathcal{V}$ from the driver's view state space $\mathcal{X}$ to target velocities inputs $\mathcal{V}$. We consider policies $\pi_\theta : \mathcal{X} \to \mathcal{V}$ parameterized by some $\theta \in \Theta$, in this case the weights of a three layer feed-forward neural network implemented in SciKit-Learn [21].

Under the assumptions, any such policy π_θ induces a probability density over the set of trajectories of length T:

$$p(\tau|\pi_\theta) = p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)\,,$$

where a trajectory τ of length T is a sequence of state and velocity tuples: $\tau = \{(\mathbf{x}_t, v_t)\}_{t=0}^{T-1}$

We collect demonstrations using FLUIDS' velocity planner π_{θ^*}, where θ^* may not be contained in Θ. We measure the difference between the two learners and supervisor using a surrogate loss $l : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ [28], [27]. The surrogate loss we consider is the indicator function, since the target velocities are discretized, $l(v_1, v_2) = \mathbb{1}(v_1 \neq v_2)$. The objective of LfD is to minimize the expected surrogate loss under the distribution induced by the robot's policy:

$$\min_\theta E_{p(\tau|\pi_\theta)} \sum_{t=0}^{T-1} l(\pi_\theta(\mathbf{x}_t), \pi_{\theta^*}(\mathbf{x}_t))\,. \tag{10}$$

However, in practice, this objective is difficult to optimize because of the coupling between the loss and the robot's distribution on states. Thus, we instead minimize an upper-bound on this objective [14] via sampling N trajectories from the supervisor's policy.

$$\min_\theta \sum_{n=0}^{N-1} \sum_{t=0}^{T-1} l(\pi_\theta(\mathbf{x}_{t,n}), \pi_{\theta^*}(\mathbf{x}_{t,n}))\,; \quad \tau \sim p(\tau|\pi_{\theta^*})\,.$$

VII. Experiments

We perform all timing experiments on a 6-core 12-thread 3.2GHz i7-970 CPU, with no GPU acceleration. Parallel timing experiments were collected using Python's multiprocessing module.

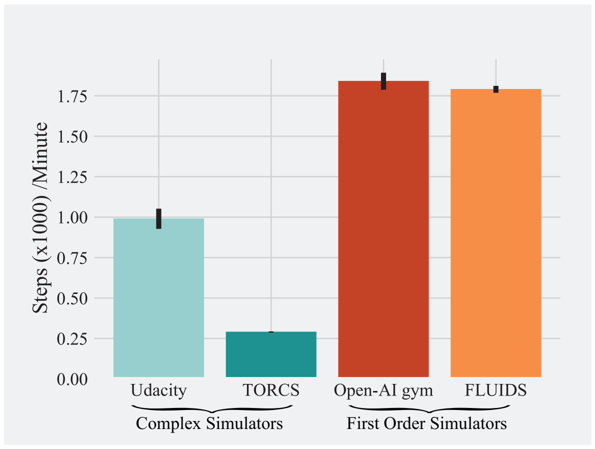

Fig. 6: Comparison of FLUIDS to other common driving simulators with rendering on. FLUIDS is simulating a four-way intersection with an idling car. FLUIDS demonstrates comparable steps per second to other lightweight simulators such as Open-AI gym and significant improvement over graphically heavy simulators such as TORCS and Udacity.

A. Benchmarking Simulation Speed

We compare the steps per second of three simulators Udacity, TORCS and OpenAI against FLUIDS. The recorded speeds [11], measured in thousands of steps per minute, with rendering enabled, are reported in Fig. 6. In each of these experiments, the scenario with fewest agents is loaded to get the absolute fastest speeds each simulator can run. In FLUIDS, we load a four-way intersection with traffic lights and one car.

FLUIDS is on par with OpenAI's lightweight racetrack simulation. Table I shows FLUIDS performing at nearly 4 times faster when rendering disabled.

FLUIDS Speed	mean	stdev
with rendering	1.79	0.021
without rendering	7.31	0.114

TABLE I: Steps (x1000) per minute for FLUIDS with and without rendering.

For the rest of the experiments we disable rendering.

We now examine how fast FLUIDS can collect data points for learning. In this experiment, we use the algorithmic supervisor as the demonstrator and generate 800 trajectories. We sample new demonstrations from an initial state distribution that ranges from $2-7$ cars.

For the learner's policy representation, we use SciKit-Learn's feed-forward neural-network implementation [21] and the quasi-LIDAR representation.

We collect 800 trajectories of data from the algorithmic supervisor. From each trajectory, we extract data points as state-action pairs for every vehicle in the trajectory. In Fig. 7, we evaluate how many data points we can obtain per minute with the algorithmic supervisor. Using Python multiprocessing to parallelize data generation over 10 cores, we generate over 8000 data points per minute. These results also suggest the importance of the parallelized data collection, which consistently shows speedups of over four times that of single-threaded data collection.

Additionally, we roll out the learned policy and collect data from it to study how FLUIDS performs with a fast agent

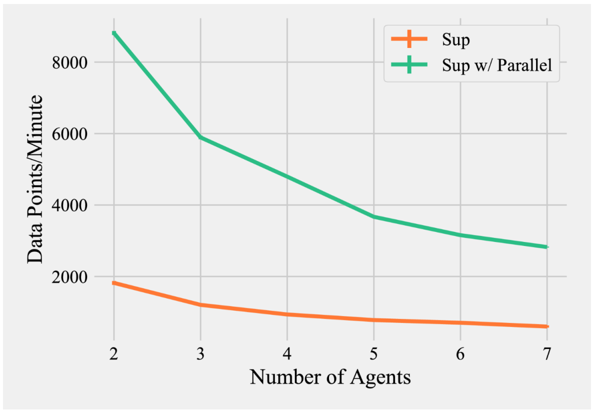

Fig. 7: The data collection rate for the FLUIDS built-in agent is compared to the number of agents simulated. The peak parallel data collection rate is over 8000 state-action pairs per minute for the supervisor, and nearly 24000 state-action pairs per minute for the imitation learned agent.

using learned behavior. The results, shown in Fig. 8, indicate that data collection speeds are improved by a factor of four again over data collection speeds on the supervisor. Run in parallel across 10 cores, and with 2 agents, FLUIDS manages over 24000 data points per minute. The performance dropoff due to overhead in the parallel experiments as the number of agents increases is not nearly as severe, with FLUIDS maintaining over 15000 data points per minute for up to 7 cars in a scene.

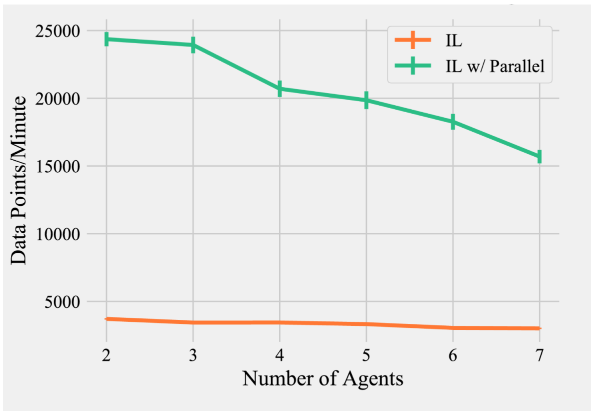

Fig. 8: The evaluation rate of the imitation learned agent is also compared to the number of agents simulated. We note the expected dropoff with the increased number of supervisor agents as well as the 5x speedup with parallelization

We also study the success rate of the supervisor versus the number of cars in the current scene. Fig. 9 shows that the supervisor agent can handle scenes with up to and including seven cars with almost 90% success rate. A run of the simulator is designated successful if all the cars make it through the intersection without gridlock, colliding with other cars, and breaking traffic laws. However, with more cars, there is a steep dropoff showing the supervisor struggles to coordinate on challenging scenes with many multi-agent interactions.

B. *Evaluation of Policy*

To evaluate how sensitive the learned policy is to observation noise and simulator parameters, we perform a grid search over six parameters of the simulator:

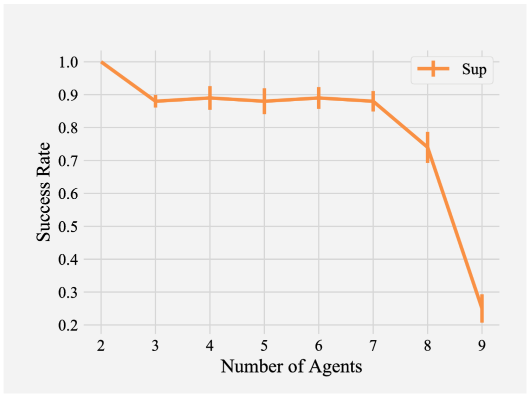

Fig. 9: The performance of the FLUIDS built-in supervisor. The FLUIDS supervisor performs reliably, with over a 90% success rate for up to 7 agents.

- The number of cars in the scene, in the range $[2, 5]$.
- The number of pedestrians in the scene, in the range $[0, 4]$.
- The variance, λ_l of quasi-LIDAR noise, in the range $[0, 1]$.
- The probability of omitting a sensor reading from the quasi-LIDAR state, in the range $[0, 1]$.
- The presence of traffic lights, $[0, 1]$.
- The mass of the cars, in the range $[0, 200]$.

See Fig. 10. FLUIDS evaluates the learned policy across 256 configurations, running in total over 5000 trajectories. The analysis suggessts that the performance of the learned policy is very sensitive to perturbations in the number of cars in the scene, with more cars providing a challenge and leading to a lower success rate. The sensitivity analysis also allows us to visualize the effect of quasi-LIDAR dropout, with our imitation learned agent being robust to increased omission probability until all the quasi-LIDAR observations are omitted, after which the success rate drops drastically. Additionally, the sensitivity analysis indicates the importance of traffic lights, which coordinate traffic through the intersection and prevent gridlock. Finally, the sensitivity analysis shows the importance of car mass. When model mismatch in this parameter is present, the learned agent tends to under and over-accelerate, causing collisions.

VIII. Discussion and Future Work

FLUIDS is a fast first-order simulator of multi-agent driving behaviors. The interface is lightweight, highly configurable, suitable for developing and evaluating autonomous agents in urban environments.

FLUIDS is limited in its current form to simulating short interactions across a single intersection. A town environment, similar to those provided in CARLA [6], could prove useful in evaluating agents across longer periods of time.

Additionally, the included algorithmic supervisors of FLUIDS, while sufficient for simulating simple multi-agent interactions, fail to exhibit the more complex behaviors of urban driving, such as yielding for pedestrians and right-of-way awareness. The greedy algorithmic supervisors also fail to capture cautious driving behaviours.

In the future, we wish to capture the diversity of driving behavior via leveraging real world data of traffic intersections.

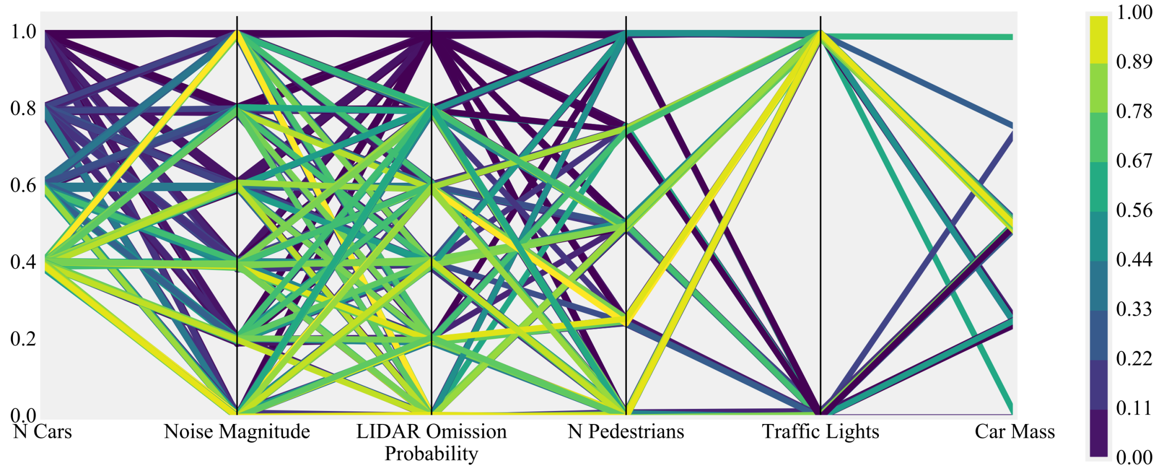

Fig. 10: Sensitivity analysis of the imitation learner to state complexity and observation noise, performed over 256 configurations by evaluating the learned policy over half a million data points. For visualization we normalize all parameters between 0 and 1. We sweep between 2 and 5 cars, 0 and 4 pedestrians, and vary the noise and dynamics parameters. We report success rate as the percentage of evaluations in which the learned policy successfully guides all vehicles to their goals. The analysis reports that our imitation learned agent is tolerant to noisy observations, but fails when run in an environment with a large number of cars or pedestrians. We also observe model mismatch when we alter the car mass and remove traffic lights.

Recent work by Ren et al., shows promise in being able to extend the realism of traffic intersection modeling with real world behavior [25].

There are also a few major components that will make FLUIDS more accessible to researchers. Creating an interface to programmatically generate, load, or save new intersections will allow researchers to conduct extensive studies across a wide variety of intersections, hone in on particularly tricky configurations for an in-depth study, and share driving scenarios. Creating a ROS interface can allow researchers with well-established software stacks to easily integrate with FLUIDS.

IX. 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, Berkeley Deep Drive (BDD), and the CITRIS "People and Robots" (CPAR) Initiative and with UC Berkeley's Center for Automation and Learning for Medical Robotics (Cal-MR). The authors were supported in part by donations from Samsung, Siemens, Google, Honda, Intel, Comcast, Cisco, Autodesk, Amazon Robotics, Toyota Research Institute, ABB, Knapp, and Loccioni.

We thank our colleagues who provided helpful feedback and suggestions, in particular, Daniel Seita, Sanjay Krishnan, and Joseph Gonzalez.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] R. Bergholz, K. Timm, and H. Weisser, "Autonomous vehicle arrangement and method for controlling an autonomous vehicle," Nov. 21 2000, uS Patent 6,151,539.

[3] A. Best, S. Narang, L. Pasqualin, D. Barber, and D. Manocha, "Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints," *arXiv preprint arXiv:1703.08561*, 2017.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[5] E. Catto, "Box2d: A 2d physics engine for games," 2011.

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning (CORL)*, 2017, pp. 1–16.

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[8] Q. Fan, Y. Yi, L. Hao, F. Mengyin, and W. Shunting, "Semantic motion segmentation for urban dynamic scene understanding," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE, 2016, pp. 497–502.

[9] M. P. Fanti, G. Iacobellis, A. M. Mangini, and W. Ukovich, "Freeway traffic modeling and control in a first-order hybrid petri net framework," *IEEE transactions on automation science and engineering*, vol. 11, no. 1, pp. 90–102, 2014.

[10] A. F. Foka and P. E. Trahanias, "Predictive autonomous robot navigation," in *Intelligent Robots and Systems (IROS), 2002. IEEE/RSJ International Conference on*, vol. 1. IEEE, 2002, pp. 490–495.

[11] W. Hsieh, "First order driving simulator," Master's thesis, EECS Department, University of California, Berkeley, May 2017. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-102.html

[12] M. M. Khorshid, R. C. Holte, and N. R. Sturtevant, "A polynomial-time algorithm for non-optimal multi-agent pathfinding," in *Fourth Annual Symposium on Combinatorial Search*, 2011.

[13] 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.

[14] 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.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[16] G. Niemeyer, "Python-Constraint: Solving constraint satisfaction problems in Python," 2017, [Online; accessed ¡today¿]. [Online]. Available: https://labix.org/python-constraint

[17] S. Noh, K. An, and W. Han, "Situation assessment and behavior decision for vehicle/driver cooperative driving in highway environments," in *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*. IEEE, 2015, pp. 626–633.

[18] B. Paden, M. p, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of

motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, March 2016.

[19] H. Palm, J. Holzmann, S.-A. Schneider, and H.-M. Koegeler, "The future of car design systems engineering based optimisation," *ATZ worldwide*, vol. 115, no. 6, pp. 42–47, 2013.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *The International Journal of Robotics Research (IJRR)*, vol. 24, no. 4, pp. 295–310, 2005.

[23] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 812–818.

[24] S. Popinet *et al.*, "Shapely," https://pypi.python.org/pypi/Shapely, 2014–2018.

[25] X. Ren, D. Wang, M. Laskey, and K. Goldberg, "Learning traffic behaviors for simulation via extraction of vehicle trajectories from online video streams," in *Automation Science and Engineering (CASE), 2018 IEEE International Conference on*. IEEE, 2018.

[26] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 102–118.

[27] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[28] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[29] A. Ruano, "DeepGTAV," https://github.com/aitorzip/DeepGTAV, 2016.

[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] G. Salvo, L. Caruso, A. Scordo, G. Guido, and A. Vitale, "Traffic data acquirement by unmanned aerial vehicle," *European Journal of Remote Sensing*, vol. 50, no. 1, pp. 343–351, 2017.

[32] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[33] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *arXiv preprint arXiv:1710.05465*, 2017.

[34] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs. sourceforge. net*, vol. 4, 2000.

[35] J. Zhu, D. I. Ferguson, and D. A. Dolgov, "System and method for predicting behaviors of detected objects," Feb. 25 2014, uS Patent 8,660,734.