FLUIDS Documentation

Release 2.0

Jerry Zhao, Andrew Cui, Michael Laskey, Berkeley AUTOLAB

Sep 20, 2018

Contents:

FLUIDS - a First-Order Local Urban Intersection Driving Simulator by Berkeley AUTOLAB. `Read our paper`

CHAPTER 1

Why FLUIDS?

To study and compare Reinforcement and Imitation Learning algorithms, the most commonly used benchmarks are OpenAI Gym, Mujoco, and ATARI games. However, these benchmarks generally fail to capture real-life challenges to learning algorithms, including multi-agent interactions, noisy sensors, and generalization. FLUIDS aims to fill this gap by providing a fast, birds-eye simulation of cars and pedestrians in an urban driving setting. Below, we highlight several notable capabilities of FLUIDS.

1.1 Tests Generalization

FLUIDS is designed to test how well agents generalize to new environments that are both in and out of sample from the initial state distribution. The initial state distribution can be specified to be a range of cars on the road starting at various lane positions. The randomness stems from the number of cars currently on the road and the location of each car.

To test how robust a policy is to out of sample distributions, FLUIDS allows for perturbations such as enabling pedestrians, varying traffic light timing and changing the amount of noise in the sensor readings.

1.2 Multi-Agent Planning

Another advantage of FLUIDS is that the number of supervisors that can be controlled by an agent and that controlled by the simulator is variable. Experiments such as coordinating a fleet of self-driving cars traversing an intersection or having a single self-driving car pass an intersection with multiple cars and pedestrians are all supported.

1.3 Built-in Supervisors

In order to collect consist training data for Imitation Learning experiments and a baseline for performance. FLUIDS provides access to an array of supervisors, which can perform the driving tasks by having access to the global state of the world. The algorithms for the supervisors are the same planning stack used for the background agents.

CHAPTER 2

Installation

2.1 Installing FLUIDS Core

The FLUIDS core simulator provides the core simulation environment. Currently only install from source is supported.

```
git clone https://github.com/BerkeleyAutomation/Urban_Driving_Simulator.git
cd Urban_Driving_Simulator
pip3 install -e .
```

2.2 Installing FLUIDS Gym Environments

The FLUIDS Gym Environment is compatible with agents designed for OpenAI's Gym benchmark API. This interface layer depends on FLUIDS core.

```
git submodule update --init --recursive
pip3 install -e gym_fluids
```

CHAPTER 3

FLUIDS Core API

3.1 Simulator Interface

The most powerful way to interact with fluids is to create a `fluids.FluidSim` object. This object creates the environment, sets up all cars, and pedestrians, and controls background objects in the scene. The initialization arguments to this object control the parameters of the generated environment. A `fluids.State` object controls the layout of the scene.

class fluids.**FluidSim**(*visualization_level=1*, *fps=30*, *obs_space='fluids_obs_none'*, *obs_args={}*, *background_control='fluids_background_null'*, *reward_fn='fluids_reward_path'*, *screen_dim=800*)

This class controls the generation and simulation of the urban environment.

Parameters

- **state** (*str*) – Name of json layout file specifiying environment object positions. Default is "fluids_state_city"
- **visualization_level** (*int*) – 0 is no visualization. Higher numbers turn on more debug visuals.
- **controlled_cars** (*int*) – Number of cars to accept external control for
- **background_cars** (*int*) – Number of cars to control with the background planner
- **background_peds** (*int*) – Number of pedestrians to control with the background planner
- **fps** (*int*) – If set to a positive number, caps the FPS of the simulator. If set to 0, FPS is unbound. Default is 30
- **obs_space** (*str*) – Controls what observation representation to return for controlled cars. fluids.BIRDSEYE or fluids.NONE
- **screen_dim** (*int*) – Height of the visualization screen. Default is 800

get_control_keys()

Returns Keys for every controlled car in the scene

Return type list of keys

get_observations(*keys={}*)
Get observations from controlled cars in the scene.

Parameters **keys** (*dict of keys*) – Keys should refer to cars in the scene

Returns Dictionary mapping keys of controlled cars to FluidsObs object

Return type dict of (key -> FluidsObs)

get_supervisor_actions(*action_type=<class 'fluids.actions.SteeringAccAction'>*, *keys={}*)
Get the actions assigned to the selected car by the FLUIDS multiagent planer

Parameters

- **action_type** (*fluids.Action*) – Type of action to return. VelocityAction, SteeringAccAction, and SteeringAction are currently supported
- **keys** (*set*) – Set of keys for controlled cars or background cars to return actions for

Returns Dictionary mapping car keys to actions

Return type dict of (key -> fluids.Action)

set_state(*state*)
Sets the state to simulate

Parameters **state** (fluids.State) – State object to simulate

step(*actions={}*)
Simulates one frame

Parameters **actions** (*dict of (key -> action)*) – Keys in dict should correspond to controlled cars. Action can be of type KeyboardAction, SteeringAction, SteeringAccAction, or VelocityAction

class fluids.**State**(*layout='fluids_state_city'*, *controlled_cars=0*, *background_cars=0*, *background_peds=0*, *use_traffic_lights=True*, *use_ped_lights=True*, *vis_level=1*)
This class represents the state of the world

Parameters

- **layout** (*str*) – Name of json layout file specifiying environment object positions. Default is "fluids_state_city"
- **controlled_cars** (*int*) – Number of cars to accept external control for
- **background_cars** (*int*) – Number of cars to control with the background planner
- **background_peds** (*int*) – Number of pedestrians to control with the background planner
- **use_traffic_lights** (*bool*) – Sets whether traffic lights are generated
- **use_ped_lights** (*bool*) – Sets whether pedestrian lights are generated

3.2 Action Types

FLUIDS supports four action types. All action types are acceptable for `FluidSim.step`.

class fluids.actions.**KeyboardAction**
This action passes control to keyboard input

class fluids.actions.**SteeringAction**(*steer*)
This action provides a steering control. The supervisor will control the acceleration

Parameters **steer**(*float in range (-1, 1)*) –

class fluids.actions.**SteeringAccAction**(*steer*, *acc*)
This action provides both steering and acceleration control.

Parameters

- **steer**(*float in range (-1, 1)*) –
- **acc**(*float in range (-1, 1)*) –

class fluids.actions.**VelocityAction**(*vel*)
This action provides a target velocity for the car to track

Parameters **vel**(*float in range (0, 1)*) –

class fluids.actions.**LastValidAction**
This action causes car to replay its last valid action. This is useful when testing coarse planning methods.

3.3 Observation Types

FLUIDS supports two observation types currently, a BirdsEye observation and a Grid observation.

class fluids.obs.**FluidsObs**
The base FLUIDS observation interface

get_array()
Returns a numpy array representation of the observation

Returns

Return type np.array

class fluids.obs.**GridObservation**(*car*, *obs_dim=500*, *shape=(500*, *500)*)
Grid observation type. Observation is an occupancy grid over the detection region. Observation has 9 dimensions: terrain, drivable regions, illegal drivable regions, cars, pedestrians, traffic lights, way points, point trajectory and edge trajectory. Array representation is (grid_size, grid_size, 9)

class fluids.obs.**BirdsEyeObservation**(*car*, *obs_dim=500*)
Bird's-eye 2D top-down image centered on the vehicle, similar to what is visualized. Minor difference is that drivable regions are colorless to differentiate from illegal drivable regions. Array representation is (obs_dim, obs_dim, 3).

3.4 Command Line Interface

FLUIDS also provides a command line interface for visualizing the environment running without a user agent. Controlled agents in this mode are controllable by keyboard only.

```
python3 -m fluids
```

Run with -h flag to see arguments

```
>>> python3 -m fluids -h
usage: __main__.py [-h] [-b N] [-c N] [-p N] [-v N] [-o str] [--state file]

FLUIDS First Order Lightweight Urban Intersection Driving Simulator

optional arguments:
-h, --help    show this help message and exit
-b N          Number of background cars
-c N          Number of controlled cars
-p N          Number of background pedestrians
-v N          Visualization level
-o str        Observation type
--state file  Layout file for state generation

Keyboard commands for when visualizer is running:
 .            Increases debug visualization
 ,            Decreases debug visualization
 o            Switches observation type
```

CHAPTER 4

FLUIDS Gym Env API

The FLUIDS Gym Environment provides a simpler common interface to the FLUIDS simulator that is compatible with general agents following the Gym API.

```
import gym
import gym_fluids

env = gym.make("fluids-v2")
```

The current Gym environment supports 1 controlled car interacting with 10 background cars and 5 pedestrians. The action space is a `[steering, acc]` pair. The observation space is a `500 x 500` RGB image from the car's perspective. A reward function considering collisions and distance traveled along the given trajectory is provided.

4.1 Supervisor Agent

FLUIDS packages a supervisor agent that interfaces with FLUIDS's multiagent planner. The supervisor is useful for collecting data for and benchmarking imitation learning algorithms. To use the supervisor, pass the observation and info dictionary to the fluids_supervisor object. Since FLUIDS's multiagent planner requires full information of the world, the observation itself is not sufficient.

:: obs, rew, done, info = env.step(action) action = gym_fluids.agents.fluids_supervisor(obs, info)

Index