PGDrive

Release 0.1.1

DrivingForce

Nov 21, 2021

HOME

Welcome to the PGDrive documentation. PGDrive is an open-ended driving simulator with infinite scenes. The key features of PGDrive includes:

- **Lightweight**: Extremely easy to download, install and run in almost all platform.
- **Realistic**: Accurate physics simulation and multiple sensory input including RGB camera, Lidar and sensory data.
- **Efficient**: Up to 500 simulation step per second.
- **Open-ended**: Support generating infinite scenes and configuring various traffic, vehicle, and environmental settings.

This documentation let you get familiar with the installation and basic utilization of PGDrive. Please go through *Installing PGDrive* to install PGDrive and try the examples in *Getting Start with PGDrive* to enjoy PGDrive!

Interesting experiment results can be found in our paper. You can also visit our webpage and GitHub repo! Please feel free to contact us if you have any suggestions or ideas!

CHAPTER

ONE

TABLE OF CONTENT

1.1 Installing PGDrive

By leveraging the power of panda3d, PGDrive can be run on personal laptop, cluster, headless server with different OS.

1.1.1 Install PGDrive on macOS, Windows and Linux in the easiest way

The installation procedure on these different platforms is same and easy, we recommend to use the command following to install:

```
pip install git+https://github.com/decisionforce/pgdrive.git
```

or you can install via:

```
git clone https://github.com/decisionforce/pgdrive.git
cd pgdrive
pip install -e .
```

The basic functionality, namely the render-less simulation can be conducted extremely easily. However, if you wish to use rendering features such as the RGB, the installation need more efforts, especially in headless machine or cluster.

1.1.2 Verify the installation of PGDrive

To check whether PGDrive v0.1.1 is successfully installed, please run:

```
python -m pgdrive.examples.profile_pgdrive
```

You can also verify the efficiency of PGDrive through the printed messages. Note that the above script is supposed to be runnable in all places. Please do not run the above command in the folder that has a sub-folder called `./pgdrive`.

1.1.3 Install the PGDrive with offscreen rendering functionality

This section introduce the procedure to enable PGDrive with RGB rendering in headless machine. If the lidar information is enough for your task, you can simply install PGDrive on your headless machine using the way we mentioned above.

Note: You have to set the `config["engine_config"]["headless_machine_render"] = True` when training the agent using image as input.

However, if you want to use image to train your agent on headless machine, you have to compile a customized Panda3D. The customized Panda3D is built from the source code of panda3d, following the instructions in Panda3D: Building Panda3D. After setting up the Panda3D dependencies, we will build our own wheel through the following command:

```
python ./makepanda/makepanda.py --everything --no-x11 --no-opencv --no-fmodex \
  --python-incdir /path/to/your/conda_env/include/ \
  --python-libdir /path/to/your/conda_env/lib/ \
  --thread 8 --wheel
```

It will give you a Panda3D wheel which can run in EGL environment without the X11 support. Now please install the wheel file by:

```
pip install panda3d-1.10.xxx.whl
```

Now, PGDrive will utilize the power of cluster to train your agent!

Warning: Compiling Panda3D from source might require the **administrator permission** to install some libraries. We are working to provide a pre-built Panda3D for cluster users of PGDrive to make it easy to use on headless machines.

1.1.4 Verify the offscreen rendering functionality of PGDrive

---

Note: An easy installation of PGDrive in macOS will fail the following verification.

Please run commands below to verify the installation:

```
python -m pgdrive.tests.install_test.test_install
```

Successfully running this script means the PGDrive works well, and an image will be shown to help you check if PGDrive can launch and capture image in offscreen mode

To verify the installation on cluster, run following command instead:

```
python -m pgdrive.tests.install_test.test_headless_install
```

The script will generate images to local directory. Please fetch anc check those images from cluster to ensure PGDrive can draw scene and capture images without X11.

1.2 Getting Start with PGDrive

We provide a pre-trained RL agent to show the power of PGDrive. Please run the following script to watch its performance:

```
python -m pgdrive.examples.enjoy_expert
```

You can also manually control a vehicle with keyboard, please run:

```
python -m pgdrive.examples.enjoy_manual
```

To enjoy the process of generate map through our Block Incremental Generation (BIG) algorithm, you can also run:

```
python -m pgdrive.examples.render_big
```

Note that the above three scripts can not be run in headless machine.

You can verify the efficiency of PGDrive via running:

```
python -m pgdrive.examples.profile_pgdrive
```

You can also draw multiple maps in the top-down view via running:

```
python -m pgdrive.examples.draw_maps
```

1.2.1 Environment Usage

The usage of PGDrive is as same as other **gym** environments:

```
import pgdrive  # Import this package to register the environment!
import gym

env = gym.make("PGDrive-v0", config=dict(use_render=True))
env.reset()
for i in range(1000):
    obs, reward, done, info = env.step(env.action_space.sample())
    env.render()
    if done:
        env.reset()
env.close()
```

Any Reinforcement Learning algorithms and Imitation Learning algorithms are compatible with PGDrive.

1.2.2 Pre-defined Environments

Besides, we provide several environments for different purposes. The following table presents some predefined environment names. Please feel free to open an issue if you want to request some new environments.

Gym Environment Name	Random Seed Range	Number of Maps	Comments
PGDrive-test-v0	[0, 200)	200	Test set, not change for all experiments.
PGDrive-validation-v0	[200, 1000)	800	Validation set.
PGDrive-v0	[1000, 1100)	100	Default training setting, for quick start.
PGDrive-10envs-v0	[1000, 1100)	10	Training environment with 10 maps.
PGDrive-1000envs-v0	[1000, 1100)	1000	Training environment with 1000 maps.
PGDrive-training0-v0	[3000, 4000)	1000	First set of 1000 environments.
PGDrive-training1-v0	[5000, 6000)	1000	Second set of 1000 environments.
PGDrive-training2-v0	[7000, 8000)	1000	Thirds set of 1000 environments.
...			*More map set can be added in response to the requests*

1.3 Vehicle Configuration

We list the vehicle config here. Observation Space will be adjusted by these config automatically. Find more information and in our source code and test scripts!

- `lidar` (tuple): (laser num, distance, other vehicle info num)
- `rgb_camera` (tuple): (camera resolution width(int), camera resolution height(int), we use (84, 84) as the default value like what Nature DQN did in Atari.
- `mini_map` (tuple): (camera resolution width(int), camera resolution height(int), camera height). The bird-view image can be captured by this camera.
- `show_navi_mark` (bool): A spinning navigation mark will be shown in the scene
- `increment_steering` (bool): For keyboard control using. When set to True, the steering angle is determined by the key pressing time.
- `wheel_friction` (float): Friction coefficient

1.4 Environment Configuration

An PGDrive instance accepts a dict as the environmental config. For example, you can build a PGDrive instance with 200 generated maps via:

```
from pgdrive import PGDriveEnv
config = dict(
    environment_num=200,
    start_seed=0
)
env = PGDriveEnv(config)
```

In this page, we describe the meaning of each configuration options.

1.4.1 PGDriveEnv Config

We do generalization experiments under the default setting of PGDriveEnv. To reproduce our experiment results, no special configuration is needed.

However, PGDrive can also support other research topics, and we will simply introduce the meaning of some configuration options of PGDriveEnv.

1.4.2 Draw Scene & Visualization

- `use_render` (bool): Pop a window on your screen or not
- `offscreen_render` (bool): When you want to access the image of camera, it should be set to True.
- `force_fps` (Union[int, None]): Decide the render fps. "None" means that no fps limitation.
- `debug` (bool): For developing use, draw the scene with bounding box

1.4.3 Manual Control

- `controller` (str): "joystick" or "keyboard". Controlling vehicle by joystick is more recommended.
- `manual_control` (bool): Controllers above are available only when this flag is True
- `use_chase_camera` (bool): A perspective like racing game. usually True, when manual control
- `camera_height` (float): Chase camera height

1.4.4 TrafficManager Config

- `traffic_density` (float): Vehicle number per 10 meter, aiming to adjust the number of vehicle on road
- `traffic_mode`: Trigger mode (Triger) / reborn mode (Reborn). In Reborn mode vehicles will enter the map again after arriving its destination.
- `random_traffic` (bool): the traffic generation will not be controlled by current map seed. If set to *False*, each map will have same traffic flow.

1.4.5 Map Config

- `map` (int or string): You can set a *string* or *int* as the key to generate map in an easy way. An *int=N* means generating a map containing N blocks, and the block type is randomly selected. Since in PGDrive each block has a unique ID in *char* type, *string* can determine the block type sequence. For example, "SCrRX" means the first block is Straight, and the next blocks are Circular, InRamp, OutRamp and Intersection. We provide the following block types:

Block Type	ID
Straight	S
Circular	C
InRamp	r
OutRamp	R
Roundabout	O
Intersection	X
TIntersection	T
Fork	(WIP)

- `map_config` (dict): The original map config. Find more information in Map source code, or find usage in our test scripts.
  - `type` (str): The map can be generated by BIG according to BLOCK_NUM, BLOCK_SEQUENCE, or MAP_FILE
  - `config` (srt): A int telling BIG the total block number under BLOCK_NUM mode, or a str describing BLOCK_SEQUENCE (each block has a unique character severing as its ID, so combining them to get a map, the parameters of these block is sampled by BIG), and under MAP_FILE mode the config should be a dict describing the whole map.
  - `lane_width` (float): Width of lanes.
  - `lane_num` (int): Number of lanes in each road.

1.4.6 Generalization Environment Config

- `start_seed` (int): Random seed of first map.
- `environment_num` (int): Number of environments. BIG generates map by a random generator initialized by a specific seed. Therefore, "environment_num" maps are generated by seeds [seed for seed in range(start_seed, start_seed+environment_num)]

1.4.7 Observation Config

- `offscreen_render` (bool): If you want to use camera data, please set this to True.
- `rgb_clip` (bool): Squeeze the value between [0, 255] to [0.0, 1.0]
- `vehicle_config` (dict): Sensor parameters for vehicle
- `image_source` (str): decided which camera image to use (mini_map or front camera). Now we only support capture one image as a part of observation.

1.4.8 Action Config

- `decision_repeat` (int): The minimal step size of the world is 2e-2 second, and thus for agent the world will step decision_repeat * 2e-2 second after applying one action or step.

1.4.9 Reward Scheme

Coefficient of different kinds of reward to describe the driving goal Find more information by accessing our source code in PGDriveEnv You can adjust our primitive reward function or design your own reward function

1.4.10 Misc.

- `use_increment_steering` (bool): Keyboard control use discretized action such as -1, 0, +1. You can set this value to True to make the keyboard strokes serve as increments to existing action.
- `action_check` (bool): Check whether the value of action is between [0.0, 1.0] or not.
- `engine_config` (dict): Some basic settings for low-level physics world. More information can be found in source code.

1.4.11 PGWorld Config

This is the core of PGDrive, including physics engine, task manager and so on.

- `window_size` (tuple): Width, height of rendering window.
- `debug` (bool): The debug value in PGDriveEnv will be passed to PGWorld.
- `physics_world_step_size` (float): The minimum step size of bullet physics engine.
- `show_fps` (bool): Turn on/ turn off the frame rater.
- `onscreen_message` (bool): Turn on to show help message or your self defined messages by organizing them in a *dict* and pass it in *render(text=your_messgaes_dict)* function.
- `force_fps` (None or float): *None* means no render fps limit, while *float* indicates the maximum render FPS.
- `decision_repeat` (int): This will be written by PGDriveEnv to do ForceFPS.
- `debug_physics_world` (bool): Only render physics world without model, a special debug option.
- `headless_machine_render` (bool): Set this to true only when training on headless machine and use rgb image!!!!!!
- `use_render` (bool): The value is same as *use_render* in PGDriveEnv
- `offscreen_render` (bool): The value is same as *offscreen_render* in PGDriveEnv.

CHAPTER

TWO

CITATION

If you find this work useful in your project, please consider to cite it through:

```
@article{li2020improving,
  title={Improving the Generalization of End-to-End Driving through Procedural␣
↪Generation},
  author={Li, Quanyi and Peng, Zhenghao and Zhang, Qihang and Qiu, Cong and Liu,␣
↪Chunxiao and Zhou, Bolei},
  journal={arXiv preprint arXiv:2012.13681},
  year={2020}
}
```