MetaDrive模拟器介绍

Kyle
怕什么真理无穷，进一寸有进一寸的欢喜

EyeSight1019 等 15 人赞同了该回答 ›

零. 简介

MetaDrive 主页：MetaDrive | MetaDriverse

MetaDrive 文档：metadrive-simulator.readthedocs.io...

MetaDrive 代码：GitHub - metadriverse/metadrive: MetaDrive: Open-source driving simulator

MetaDrive 论文：arxiv.org/pdf/2109.1267...

> MetaDrive Env是一个高效的组合式 (Compositional) 的驾驶模拟器，驾驶的目标是控制一辆（或者多辆）汽车安全且按时地从起点开到终点。它具有以下特性:
> **组合式**：它支持生成具有各种道路地图和交通设置的（理论上可以无限的）场景，可用于 RL 泛化性的研究。
> **轻量化**：易于安装和运行。它在标准 PC 上可以运行高达每秒300帧 (up to 300 FPS)。
> **高逼真**：精确的物理模拟和多种类型的输入，包括激光雷达、RGB 图像、自上而下的语义地图和第一人称视角图像。用户可以自行选择强化学习中的 observation 的种类。

一. 安装

MetaDrive 的安装非常简单直接。

```
# Install Directly  (通过 pip 安装可能无法获得某些最新功能和错误修复)
pip install metadrive-simulator

# Install from source code  (recommended)
git clone https://github.com/metadriverse/metadrive.git
cd metadrive
pip install -e .
```

安装完成后，可以通过在 Python 命令行中运行如下命令验证是否安装成功：

```
from metadrive import MetaDriveEnv
env = MetadriveEnv()
obs = env.reset()
print(obs.shape)  # 输出 (259,)
```

有时资源可能会损坏或过期。在这种情况下，请运行以下代码强制更新本地资源：

```
python -m metadrive.pull_asset --update
```

二. 简单使用

MetaDrive 的使用与其他gym环境相同。几乎所有决策算法都与 MetaDrive 兼容（只要它们与 OpenAI gym✦ 兼容）。

```
from metadrive.envs.metadrive_env import MetaDriveEnv
import gymnasium✦ as gym

env = MetaDriveEnv(config={"use_render": True})
obs, info = env.reset()
for i in range(1000):
    obs, reward, terminated, truncated, info = env.step(env.action_space.sample
    if terminated or truncated:
        env.reset()
env.close()
```

三. 环境（暂不做重点了解）

默认情况下，有 5 种现成的 RL 环境：

- Generalization environment
- VaryingDynamics environment
- Safe RL environment
- MARL environment
- Real-world environment

四. 配置

创建环境时，外部配置 `config` 会覆盖 中某些字段的默认值 `default_config` 。例如：

```
from metadrive import MetaDriveEnv
default_config = MetaDriveEnv.default_config()
env = MetaDriveEnv(dict(num_scenarios=100, log_level=50))
env_config = env.config
print("default_config['num_scenarios']:", default_config["num_scenarios"])  # d
print("env_config['num_scenarios']:", env_config["num_scenarios"])          # e
```

默认参数的简单注释：

```
代理（Agent）相关参数
random_agent_model: 是否随机选择代理的车辆模型，从四种类型中随机选择。默认: False
agent_configs: 代理的配置，包含使用特殊颜色和生成车道索引等设置。默认: {DEFAULT_AGENT: c

多代理（Multi-Agent）相关参数
num_agents: 在多智能体环境中，代理的数量。若设置为-1，则尽可能多地生成车辆。默认: 1
is_multi_agent: 是否启用多智能体环境。默认: False
allow_respawn: 是否允许代理在每个回合结束后重生。默认: False
delay_done: 代理完成后静止在死亡位置的时间步数（MARL默认25个时间步）。默认: 0

动作/控制（Action/Control）相关参数
agent_policy: 控制代理的策略。默认: EnvInputPolicy
manual_control: 是否启用手动控制，若为真，则代理策略将被覆盖为手动控制策略。默认: False
controller: 使用的手动控制接口选项（方向盘、键盘或其他）。默认: "keyboard"
discrete_action: 如果为真，环境的动作空间将是离散的。默认: False
use_multi_discrete: 是否使用多离散动作空间。默认: False
discrete_steering_dim: 转向维度的离散动作数量。默认: 5
discrete_throttle_dim: 油门/刹车维度的离散动作数量。默认: 5
action_check: 检查动作是否在gym.space中，通常关闭以加快仿真速度。默认: False

观察（Observation）相关参数
norm_pixel: 是否将像素值从0-255规范化到0-1。默认: True
stack_size: 堆叠图像观察的时间步数。默认: 3
image_observation: 是否使用图像观察或激光雷达观察。默认: False
agent_observation: 用户可以使用自定义观察类。默认: None

终止（Termination）相关参数
horizon: 每个代理回合的最大长度。如果设置为None，则移除该限制。默认: None
truncate_as_terminate: 如果为真，当代理回合超出horizon时，也会被视为终止。默认: False

主摄像头（Main Camera）相关参数
use_chase_camera_follow_lane: 是否让摄像头跟随参考线而不是车辆，以便平滑运动。默认: Fal
camera_height: 主摄像头的高度。默认: 2.2
camera_dist: 摄像头与车辆之间的距离（the distance projecting to the x-y plane.）。默
camera_pitch: 摄像头的俯仰角度（如果为None，将自动计算）。默认: None
camera_smooth: 是否平滑摄像头移动。默认: True
camera_smooth_buffer_size: 用于平滑摄像头的帧数。默认: 20
camera_fov: 主摄像头的视场角（FOV of main camera）。默认: 65
prefer_track_agent: 在多代理（MARL）设置下，选择跟踪的代理。默认: None
top_down_camera_initial_x/y/z: 顶视摄像头（for 3D viewer）的初始位置。默认: 0, 0, 20

车辆（Vehicle）相关参数
vehicle_model: 车辆模型，候选值包括“s”、“m”、“l”、“xl”、“default”。默认: "default"
enable_reverse: 是否允许车辆倒退。默认: False
show_navi_mark: 是否显示导航点的标记。默认: True
show_dest_mark: 是否在目的地显示标记。默认: False
show_line_to_dest/navi_mark: 是否绘制从当前位置到目的地或导航点的线。默认: False
show_navigation_arrow: 是否显示导航方向的箭头。默认: True
use_special_color: 在顶视渲染或多代理设置下，车辆是否显示为绿色。默认: False
no_wheel_friction: 是否清除车轮摩擦，以便通过设置方向盘和油门/刹车来控制车辆。默认: False

图像捕捉（Image Capturing）相关参数
image_source: 使用的图像观察的摄像头。默认: "rgb_camera"
```

车辆生成和导航（Vehicle Spawn and Navigation）相关参数
navigation_module: 基本导航实例，需与道路网络类型匹配。默认: None
spawn_lane_index: 车辆生成的车道ID。默认: None
destination: 目的地的车道ID，仅在导航模块不为None时需要。默认: None
spawn_longitude/lateral: 在生成车道上的纵向和横向位置。默认: 5.0, 0.0
（spawn_position_heading=None,
spawn_velocity=None, # m/s
spawn_velocity_car_frame=False,）：如果指定了以上项目，车辆将以一定的速度在指定位置生成
spawn_position_heading/velocity: 车辆生成时的方向和速度。默认: None, None
overtake_stat: 车辆超车的统计（因bug已弃用）。默认: False
random_color: 是否使用随机颜色替换车辆默认纹理。默认: False
（width=None,
length=None,
height=None,
mass=None,）：车辆的形状是按其类别预先确定的。但在特殊场景（WaymoVehicle）我们可能想要其他

车辆模块配置（Vehicle Module Config）相关参数
lidar/side_detector/lane_line_detector: 激光雷达和其他传感器的配置。默认: dict(...)
show_lidar/show_side_detector/ show_lane_line_detector: 是否显示xxx。默认: False/
light: 是否开启车辆灯光，仅在启用渲染管道时可用。默认: False

传感器（Sensors）相关参数
sensors: 各种传感器的配置。默认: dict(lidar=(Lidar,), side_detector=(SideDetector

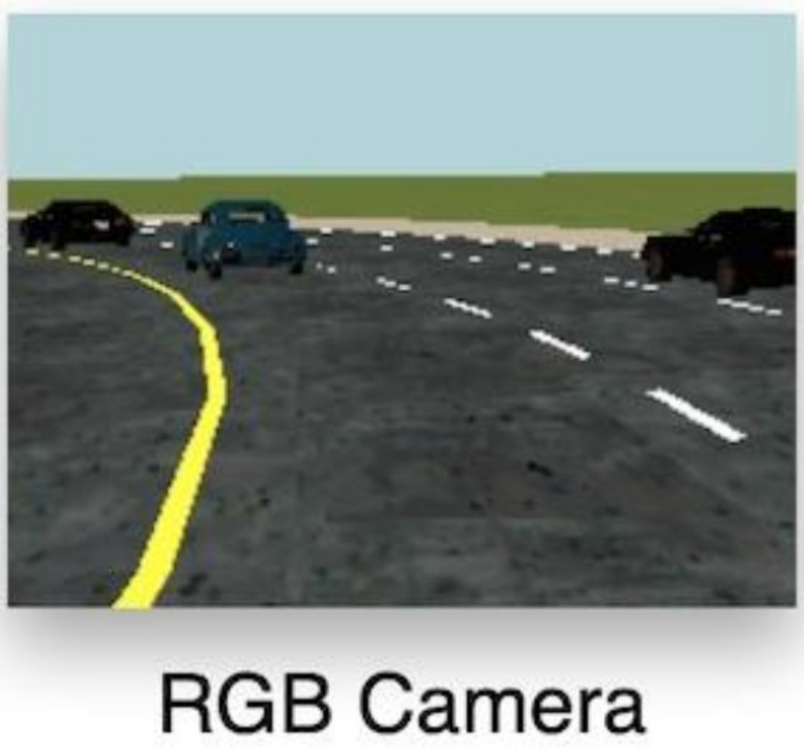
引擎核心配置（Engine Core Config）相关参数
use_render: 是否弹出窗口进行渲染。默认: False
window_size: 窗口的宽度和高度。默认: (1200, 900)
physics_world_step_size: 物理世界步进大小。默认: 2e-2
decision_repeat: 每次env.step()调用时的决策重复次数。默认: 5（这个是什么意思？）
image_on_cuda: 是否在CUDA上进行图像处理。默认: False
_render_mode: 渲染模式，自动确定，不要修改。默认: RENDER_MODE_NONE
force_render_fps: 渲染帧速率限制。默认: None
force_destroy: 是否强制销毁对象以提升效率。默认: False
num_buffering_objects: 每个类的缓冲对象数量。默认: 200
render_pipeline: 是否启用渲染管道。默认: False
daytime: 渲染时的时间设置。默认: "19:00"
shadow_range: 阴影范围。默认: 50
multi_thread_render: 是否使用多线程渲染。默认: True
multi_thread_render_mode: 多线程渲染模式。默认: "Cull"
preload_models: 是否预加载模型以避免首次创建时的延迟。默认: True
disable_model_compression: 是否禁用模型压缩以提高启动速度。默认: True

地形（Terrain）相关参数
map_region_size: 地图区域的大小。默认: 1024
cull_lanes_outside_map: 是否移除地图区域外的车道。默认: False
drivable_area_extension: 可通行区域的扩展宽度。默认: 7
height_scale: 山地的高度缩放。默认: 50
use_mesh_terrain: 是否使用网格碰撞。默认: False
full_size_mesh: 是否在整个区域使用物理体。默认: True
show_crosswalk/sidewalk: 是否显示人行横道和人行道。默认: True

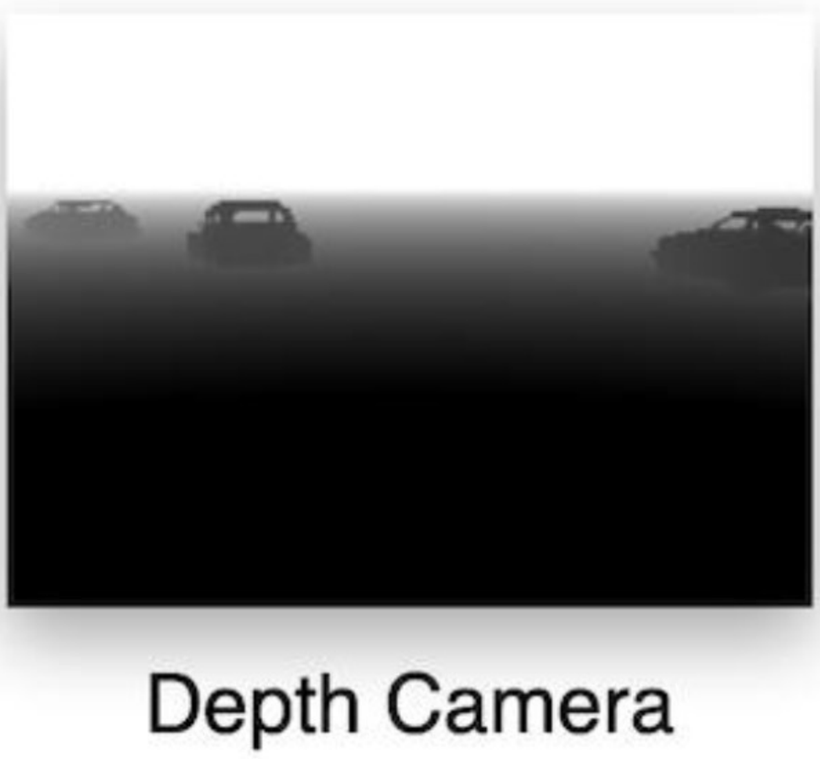
```
调试（Debug）相关参数
pstats: 是否开启性能分析。默认: False
debug: 是否输出更多调试信息。默认: False
log_level: 日志级别设置。默认: logging.INFO
show_coordinates: 是否显示地图和对象的坐标。默认: False

GUI相关参数
show_fps: 是否显示帧率。默认: True
show_logo: 是否显示 logo。默认: True
show_mouse: 是否显示鼠标指针。默认: True
show_skybox: 是否显示天空盒。默认: True
show_terrain/interface: 是否显示地形和界面。默认: True
show_policy_mark: 是否显示多策略设置的标记。默认: False
show_interface_navi_mark: 是否显示导航信息标记。默认: True
interface_panel: 在窗口中显示的传感器输出面板。默认: ["dashboard"]

记录/重放元数据（Record/Replay Metadata）相关参数
record_episode: 是否记录回合元数据。默认: False
replay_episode: 重放的日志数据。默认: None
only_reset_when_replay: 是否在重放时仅重置第一帧。默认: False
force_reuse_object_name: 是否在重放时使用与数据集中相同的ID。默认: False

随机化（Randomization）相关参数
num_scenarios: 环境中的场景数量。默认: 1
```

五. 观察空间

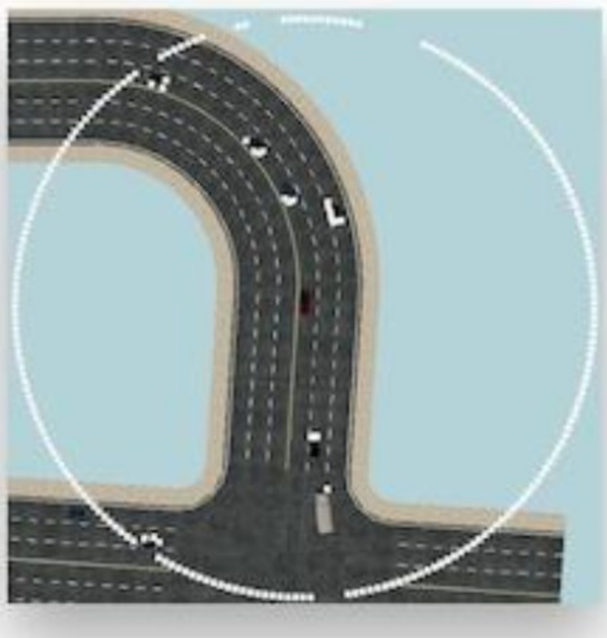
MetaDrive 提供各种传感器输入，如下图所示。对于低级传感器，RGB 相机、深度相机、语义相机、实例相机和激光雷达可以放置在场景中的任何位置，并可调整视野和激光数量等参数。同时，还可以提供高级场景信息，包括道路信息和附近车辆的速度和航向等信息作为观察值。



RGB Camera

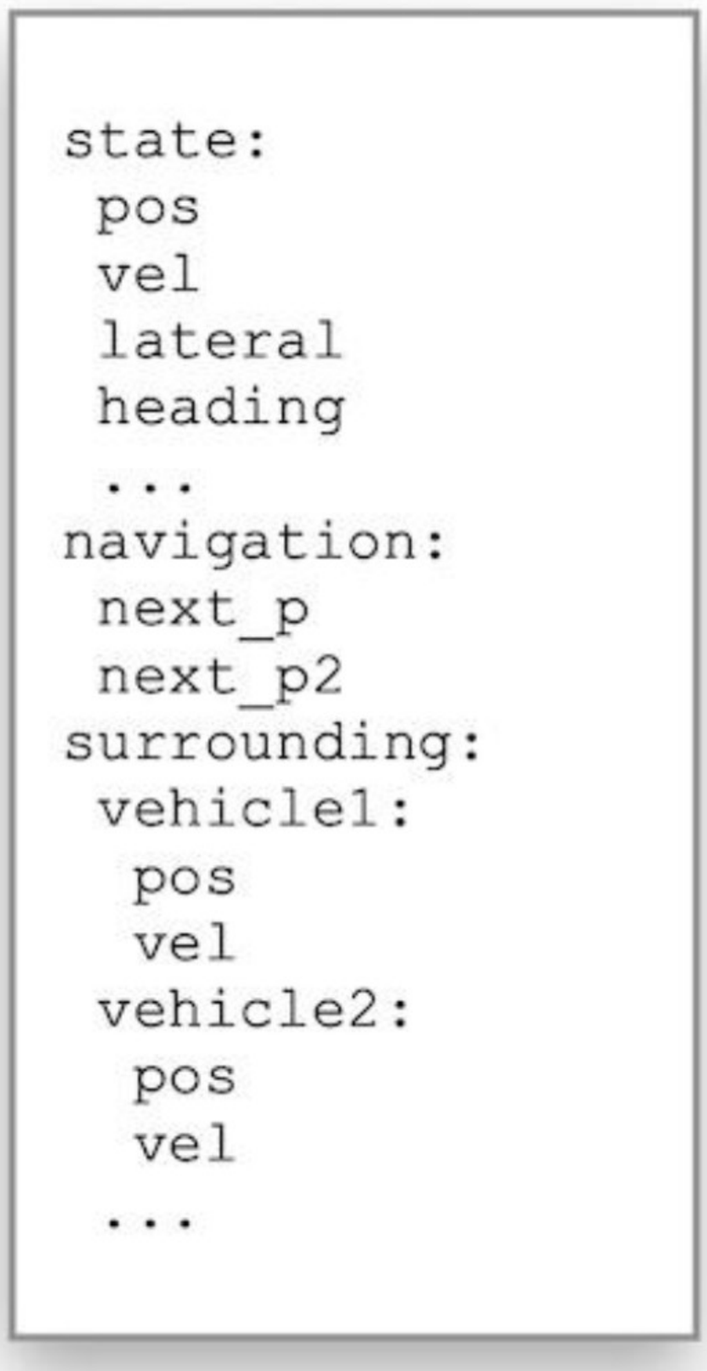


Depth Camera



Lidar

Bird-view



Sensory Data

需要注意的是，MetaDrive 旨在提供一个高效的平台来对 RL 研究进行基准测试，因此我们以牺牲真实感渲染效果为代价来提高模拟效率。

通常用于训练代理的观察有三种：（默认为LidarStateObservation✦，以下只讨论默认情况）

- LidarStateObservation
- ImageStateObservation
- TopDownObservation✦

车辆的观察空间为259维的 numpy 数组，数据类型为 float32，obs shape 为 (259,)。观察空间包含以下3部分内容：

- ego_state： 即当前状态，例如航向，转向，速度和到边界的相对距离。在StateObservation `vehicle_state` 函数中实现。
- navigation： 即导航信息，具体来说，MetaDrive 首先计算从生成点到自车目的地的路线。然后，一组检查点以一定的间隔分散在整个路线上。在收集导航信息时，将选择接下来的两个或更多个检查点，并将其位置投影到自车的本地坐标。最终的导航观测由这些位置向量和其他信息（即道路的方向和曲率）组成。在Navigation Class_中get_info_for_checkpoint实现。
- surrounding： 即周围的信息，包含类似激光雷达的云点的向量编码。通常使用 240 个激光器（单智能体）和 70 个激光器（多智能体）扫描半径为 50 米的邻近区域。此外，还可以选择包括感知车辆的信息，例如它们在自车本地坐标中表示的位置和航向。

上述信息被标准化为 [0,1]然后连接成状态向量喂给rl agent。

六. 动作空间

MetaDrive 环境的动作空间是2维的连续动作，其有效范围为 [-1, 1]。 通过这样的设计，每个智能体的动作空间都被固定为 gym.spaces.Box(low=-1.0, high=1.0, shape=(2,))。

- 第一个维度代表转向角 (steering)。当动作取1或者-1时分别代表方向盘向左或者向右转到最大转向角度，取0的时候代表方向盘朝向正前方。
- 第二个维度代表加速或者刹车。当范围在 (0,1) 区间时候代表加速，范围在 (-1,0) 代表刹车；取0的时候代表不采取任何动作。

MetaDrive 中的汽车是前轮转向，四轮驱动 (4WD)。

七. Reward, Cost, Termination, and Step Information

在env.step()后，环境会返回一个tuple (obs,reward,terminated,truncated,info)。

对于所有环境，奖励函数通常由密集驾驶奖励和稀疏终端奖励组成。密集奖励是沿着参考线或车道向目的地纵向移动。当episode因到达目的地或驶出道路而终止时，稀疏奖励将添加到密集奖励中。

奖励函数由以下三部分组成：

$$R = c_1 R_{driving} + c_2 R_{speed} + R_{termination}$$

- 驾驶奖励 (driving_reward)： $R_{driving} = d_t - d_{t-1}$,其中 d_t, d_{t-1} 表示目标车辆在当前参考车道上连续两个时间步长的纵坐标，提供密集的奖励以鼓励代理向目的地移动。
- 速度奖励 (speed_reward)： $R_{speed} = v_t / v_{max}$ ，激励agent快速开车。 v_t, v_{max} 分别表示当前速度和最大速度（80 公里/小时）。速度越大，则奖励越大。
- 终止奖励 (terminaltion_reward)： 在 episode结束时候，其他密集奖励将被禁用，并根据车辆的状态返回一个奖励。具体的情况可以分为：

```
success_reward = 10.0: 终止奖励之一。
out_of_road_penalty = 5.0: 将使用-5.0作为终止奖励。
crash_vehicle_penalty = 5.0: 将使用-5.0作为终止奖励。
crash_object_penalty = 5.0: 将使用-5.0作为终止奖励。
```

另外，metadrive提供一个配置调用 `use_lateral_reward` ，它是一个范围在 [0, 1] 内的乘数，表示自车是否远离当前车道的中心。乘数将应用于驾驶奖励。（TODO：是否即为c1？或者说driving_reward还有一个隐藏的系数）

其他默认的奖励配置

- `driving_reward = 1.0` : the c1 in reward function.
- `speed_reward = 0.1` : the c2 in reward function.
- `use_lateral_reward = False` : 禁用根据车道居中情况来加权驾驶奖励

---

info["cost"]与奖励函数类似，metadrive也提供了默认的成本函数来衡量驾驶过程中的安全性。

- `crash_vehicle_cost = 1.0`
- `crash_human_cost = 1.0`
- `crash_object_cost = 1.0`
- `out_of_road_cost = 1.0`

终止和截断：

1. 目标车辆到达目的地，
2. 车辆驶出道路，
3. 该车辆与其他车辆相撞，
4. 车辆撞上障碍物，
5. 车辆撞到人，
6. 达到最大步长（水平）限制，或
7. 车辆撞向建筑物（例如在多代理收费站环境中）。

`info` 包含有关环境和目标车辆的当前状态的丰富信息。步骤信息是从各种来源收集的，例如引擎、奖励函数、终止函数、流量管理器、代理管理器等。

环境大概了解完了，接下来跑跑代码debug再看看吧~

参考资料

metadrive-simulator.readthedocs.io...

di-engine-docs.readthedocs.io...

发布于 2024-09-19 13:44 · IP 属地辽宁