NavSim

2.10.4-dev0

Armando Fandango, Troyle Thomas

Jul 22, 2021

Contents

CHAPTER 1

Introduction

A navigation simulator API built on top of Python, Stable Baselines 3, Pytorch.

Can use many simulator backends, for now uses the Aurora Simulator, that is a Unity3D GameEngine based Berlin city environment.

CHAPTER 2

Navsim Environment Tutorial

NavSimGymEnv Class is a wrapper to Unity2Gym that inherits from the Gym interface The configuration provided is as follows:

2.1 How to use the navsim env

If you only want to use `NavSimGymEnv`, then all you need to do is install `navsim` from pip and then either subclass it or use it as follows:

```
import navsim
import gym

env_config = navsim.util.ObjDict({
        "env_path": "/data/work/unity-envs/Build2.9.2/Berlin_Walk_V2.x86_64",
        "log_folder":"./env_log",
        "task": 0,
        "goal": 0,
        "goal_distance": 50,
        "reward_for_goal": 50,
        "reward_for_no_viable_path":-50,
        "reward_step_mul": 0.1,
        "reward_collision_mul": 4,
        "reward_spl_delta_mul": 1,
        "agent_car_physics": 0,
        "debug":False,
        "obs_mode":0,
        "seed":123,
        "save_vector_obs":True,
        "save_visual_obs":True
    })
```

(continues on next page)

(continued from previous page)

```
env = gym.make("navsim-v0", env_config=env_config)
# or use the following method to create an env
env = navsim.env.NavSimGymEnv(env_config)
```

If you want to use our `navsim` conda environment or `navsim` container then follow the instructions <insert_link_here>.

2.2 Config Parameters

```
env_config = {
    "log_folder": "unity.log",
    "seed": 123,
    "timeout": 600,
    "worker_id": 0,
    "base_port": 5005,
    "obs_mode": 2,
    "segmentation_mode": 1,
    "task": 0,
    "goal": 0,
    "goal_distance":50
    "max_steps": 10,
    "reward_for_goal": 50,
    "reward_for_ep": 0.005,
    "reward_for_other": -0.1,
    "reward_for_falling_off_map": -50,
    "reward_for_step": -0.0001,
    "agent_car_physics": 0,
    "episode_max_steps": 10,
    "env_path":args["env_path"]
}
```

2.2.1 Observation Mode

- `0` - Vector - Returns [Agent Position (3-x,y,z) ,Agent Velocity (3-x,y,z), Agent Rotation(4-x,y,z,w), Goal Position (3-x,y,z,w)]
- 1 - Visual- Returns [[Raw Agent Camera](84,84,3), [Depth Agent Camera](84,84,1), [Segmentation Agent Camera](84,84,3)]
- 2 - VectorVisual - Returns [[Raw Agent Camera](84,84,3), [Depth Agent Camera](84,84,1), [Segmentation Agent Camera](84,84,3), [Agent Position (3-x,y,z), Agent Velocity (3-x,y,z), Agent Rotation (4-x,y,z,w), Goal Position (3-x,y,z,w)]]

2.2.2 Segmentation Mode

- `0` - Object Seg: Each gameobject in the scene is a unique color
- `1` - Tag Seg: Gameobject colors are based on the tag assigned such that all objects with the same tag share a color. (E.g. Car, Tree, Buildings)
- `2` - Layer Seg: Similar to tag segmentation but with the physics layers. Current layers (Default, Trees, Tree Colliders, Agent Vehicle, Autonomous Vehicle, Parked Vehicle)

2.2.3 Task

- `0` - PointNav - Agent is randomly placed along with a randomly place goal position. The agent must navigate to the goal position.
- `1` - SimpleObjectNav1 - The Agent is place at a specified starting location (manually identified traffic intersection). Goal is a sedan 40m forward in a straight line of the agent. The goal is to reach that sedan.
- `2` - ObjectNav - The Agent is randomly place and goal object is defined by the goal parameter. The agent must reach one instance of the goal object. E.g. The goal object is a sedan and there any multiple sedans in the scene. Reaching any of the sedans results in a success.

2.2.4 Goal : Only relevant for SimpleObjectNav and ObjectNav

- `0` - Tocus
- `1` - sedan1
- `2` - Car1
- `3` - Car2
- `4` - City Bus
- `5` - Sporty_Hatchback
- `Else` - SEDAN

2.2.5 Rewards

- `reward_for_goal` : For pointnav goal is the target position to complete the task.
- `reward_for_ep` : Exploration points are randomly placed in the environment to reward exploration.
- `reward_for_other` : Other collision are anythin that is not a goal point or exploration point, this includes other cars, building, trees, etc.
- `reward_for_falling_off_map` : The map is a tiled XXkm area. If the agent goes outside of this area falls XXm below the environment area this reward is activated. This will also result in a reset.
- `reward_for_step` : This reward will be given at every step in addition to any other reward recieved at the same step.

2.2.6 Agent Car Physics

- `0` - Simple : Collisions and gravity only - An agent that moves by a specific distance and direction scaled by the provided action. This agent only experiences collision and gravity forces
- `1` - Intermediate 1 : Addition of wheel torque
- `2` - Intermediate 2 : Addition of suspension, downforce, and sideslip
- `10` - Complex : Addition of traction control and varying surface friction

2.3 Action Space:

```
[Throttle, Steering, Brake]
```

- `Throttle` : -1.0 to 1.0 : Moves the agent backward or forward
- `Steering` : -1.0 to 1.0 : Turns the steering column of the vehicle towards left or right
- `Brake` : 0.0 to 1.0 : Reduces the agents current velocity

2.4 Car Motion Explanation:

Simple Car Physics (Agent car physics `0`)In this mode, the steering and travel of a car is imitated without driven wheels. This means that the car will have a turning radius, but there is no momentum or acceleration that is experienced from torque being applied to wheels as in a real car.

- `[0, 0, 0]` - No throttle, steering, or braking is applied. No agent travel.
- `[1, 0, 0]` - Full forward throttle is applied. The agent travels forward at max velocity for the duration of the step.
- `[0, -1, 0]` - No throttle or braking is applied. Steering is applied as a full left-turn, but because the forward/backward speed is zero, there is no travel.
- `[1, -1, 0]` - Full forward throttle and full left-turn steering are applied. The agent travels forward at a leftward angle that is equal to a fraction of the max steering angle (25 degrees for the default car). This fraction is dependent on the length of the step in real time.
- `[-1, -1, 0]` - Full backward throttle and full left-turn steering are applied. Similar to previous example, but with backward travel.
- `[0.5, 0.5, 0]` - Half forward throttle and half right-turn steering are applied. The agent travels forward at half its max velocity and at a lesser rightward angle.
- `[1, 0, 1]` - Full forward throttle and full braking are applied. These cancel each other out and result in no agent travel.
- `[1, 0, 0.5]` - Full forward throttle and half braking are applied. The agent travels forward at half throttle.
- `[0, 0, 1]` - Full braking is applied, with no throttle or steering. No agent travel.

Torque-Driven Car Physics (Agent car physics >`0`)The agent car is driven forward by applying torque to each drive wheel. The agent will have momentum, so travel is possible in a step where no throttle is input. With those differences in mind, the action space examples are similar with some minor behavioral differences:

- `[0, 0, 1]` - Full braking is applied. The agent will slow to a complete stop if in motion.

- `[0, 0, 0.5]` - Half braking is applied. The agent will slow at a lesser rate to the previous example, until completely stopped.
- `[1, 0, 0]` - Full forward throttle is applied. The agent will travel forward at an acceleration resulting from max wheel torque (not velocity, as in the simple car physics)
- `[1, 0, 1]` - Full forward throttle and full braking are applied. The agent will not travel forward if it does not have any forward momentum, otherwise the agent will slow to a complete stop.

2.5 Observation Space:

2.5.1 The vector observation space

```
Agent_Position.x, Agent_Position.y, Agent_Position.z,
Agent_Velocity.x, Agent_Velocity.y, Agent_Velocity.z,
Agent_Rotation.x, Agent_Rotation.y, Agent_Rotation.z, Agent_Rotation.w,
Goal_Position.x, Goal_Position.y, Goal_Position.z
```

2.5.2 The visual observation space

```
[[Raw Agent Camera],[Depth Agent Camera],[Segmentation Agent Camera]]
```

2.6 Queries from the Env

2.6.1 Map

Used to request and receive a binary navigable map. The binary map indicates navigable and obstacle areas.

Map requests to Unity are sent using:

```
NavSimGymEnv.start_navigable_map(resolution_x, resolution_y, cell_occupancy_threshold)
```

The map is then retrieved with:

```
NavSimGymEnv.get_navigable_map()
```

Parameter value ranges:

```
                          Min   Max
resolution_x              1     3276
resolution_y              1     2662
cell_occupancy_threshold  0     1.0
```

The raw map array received from the Unity game is a row-major 1D flattened bitpacked array with the y-axis data ordered for image output (origin at top left).

For example, if reshaping to a 2D array without reordering with dimensions `(resolution_y, resolution_x)`, then the desired coordinate `(x,y)` is at array element `[resolution_y-1-y, x]`.Finding the agent map position based on world position*:`map_x = floor(world_x / (max_x / resolution_x) )map_y = (resolution_y - 1) - floor(world_z / (max_y / resolution_y) )`

*Note: When converting from the 3-dimensional world position to the 2-dimensional map, the world y-axis is omitted. The map's y-axis represents the world's z-axis.

2.6.2 Position Scan - Not Available

Given a position and this returns the attribution data of the first object found at the given position. Objects are searched for within a 1 meter radius of the given position. If the position is not loaded in the environment then None will be returned.

2.6.3 Shortest Path from Starting Location to Goal

`ShortestPath` : Returns the shortest path value from the agent's start location to the goal position from the navigable area.

```
NavSimGymEnv.get_shortest_path_length()
```

CHAPTER 3

How to use the navsim conda env or container

3.1 Pre-requisites

Following should be pre-installed on the host machine:

3.1.1 For running inside the containers

- nvidia driver
- docker
- nvidia container toolkit

3.1.2 For directly running on the host

- nvidia driver
- X-window system

3.2 Versions

There are three components: navsim binary, navsim python api, navsim container You can use any version of each of them as long as first two digits match. These are the latest releases of each of them:

- binary 2.10.x
- python api 2.10.x
- container 2.10.x

3.3 How to run the navsim training

You can either run directly on a host machine or in a container. If you are running on a host directly, first follow the instructions to setup the host.

1. Download and extract the unity binary zip file
2. The following environment variables need to be set in both cases:

```
envdir=$(realpath "/data/work/unity-envs/Build2.10.1-dev");
envbin="Berlin_Walk_V2.x86_64";
expdir=$(realpath "$HOME/exp");
run_id="demo";
repo="ghcr.io/armando-fandango";
cd $expdir
```

3. Now follow the container, or the host option below.

3.3.1 Option 1: Container

Note: Make sure you are in experiment directory, as container will dump the files there.

```
cd $expdir
docker run --rm --privileged -it --runtime=nvidia \
--name $run_id \
-h $run_id \
-e XAUTHORITY \
-e NVIDIA_VISIBLE_DEVICES=all -e NVIDIA_DRIVER_CAPABILITIES=all \
-e USER_ID=$(id -u) -e USER_HOME="$HOME" \
-v $HOME:$HOME \
-v /etc/group:/etc/group:ro \
-v /etc/passwd:/etc/passwd:ro \
-v /etc/shadow:/etc/shadow:ro \
-v $envdir:$envdir \
-v $expdir:$expdir \
-w $expdir \
$repo/navsim:2.10.0 DISPLAY=:0.0 <navsim command>
```

The Variable `DISPLAY=:0.0`

The display variable points to X Display server, and takes a value of `hostname:D.S`, where:

- `hostname` can be empty.
- `D` refers to the display index, which is 0 generally.
- `S` refers to the screen index, which is 0 generally but in a GPU based system, each GPU might be connected to a different screen. In our container, this number refers to the GPU on which the environment binary will run.

For the purpose of navsim container, use `DISPLAY=:0.0` and change the last zero to the index number of GPU for environment binary.

3.3.2 Option 2: Run on host directly - doesn't run headless.

Create conda env - to be done only once

1. Download following files:
   - `ezai-conda.sh`
   - `ezai-conda-req.txt`
   - `ezai-pip-req.txt`
2. miniconda: We suggest you install miniconda from our script, but if you have miniconda installed already then you can skip to next step to create conda environment. If next step doesn't work, then come back and follow the instructions to install miniconda.

```
CONDA_ROOT=/opt/conda
sudo mkdir $CONDA_ROOT
sudo chown $(id -u) $CONDA_ROOT
source ezai-conda.sh && install_miniconda
```

3. Create the conda env for `navsim`

```
ENVS_ROOT=$(conda info --base)/envs
source ezai-conda.sh && ezai_conda_create --venv "$ENVS_ROOT/navsim"
```

Run the navsim on host

First activate the `navsim` virtual environment - only once, with following command: `conda activate navsim || source activate navsim`.

Now the navsim env should be activated. If not then go to host setup steps and troubleshoot.

Run the `navsim` command as described in its section below.

3.3.3 The `<navsim command>`

- `navsim --help` shows the options
- `navsim --run_id $run_id --env $envdir/$envbin` - executes and/or trains the model
- `navsim-benchmark $envdir/$envbin` - benchmarks the model
- `navsim-saturate-gpu $envdir/$envbin` - Saturates the GPU
- Replace the navsim command with your own command if you are just importing the NavSim env and have your own code in experiment directory.

3.4 TODO: Clean up the following section

For tmux hotkeys press ctrl+b then following key

- Start tmux session: tmux new -s
- Open another tmux shell: ctrl + b, % (vertical pane) Or ctrl + b, ” (horizontal pane)
- Move between panes: ctrl + <left, right, up, down>
- Detach from tmux session: ctrl + b, d (detach from tmux session)
- Attach to existing tmux session: tmux attach -t
- Exit Session: Type exit into all open shells within session

3.5 TODO: To run the singularity container

Note: Do it on a partition that has at least 10GB space as the next step will create navsim_0.0.1.sif file of ~10GB.

singularity pull docker://$repo/navsim$:ver singularity shell –nv -B not needed if path to binary is inside $HOME folder - B < absolute path of current folder > not needed if path to current folder is inside$HOME folder navsim_$ver.sif

For IST Devs: From local docker repo for development purposes:

SINGULARITY_NOHTTPS=true singularity pull docker://$repo/navsim$:ver

CHAPTER 4

Contributing to NavSim API

4.1 General dev info:

- Use only google style to document your code: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html#example-google

4.2 How to setup dev laptop to code for navsim API

- clone the ai_coop_py repo

  ```
  git clone <blah blah>
  ```

- Follow instructions in setting up navsim on the host

4.3 Testing from local repo

For IST Devs: From local docker repo for development purposes:

```
repo="localhost:5000"
```

CHAPTER 5

NavSim API

5.1 NavSim environment and related utilities

navsim.env.**env_info**(*env*)

Prints the information about the environment

class navsim.env.**NavSimGymEnv**(*env_config*)

NavSimGymEnv Class is a wrapper to Unity2Gym that inherits from the Gym interface

Read the **NavSim Environment Tutorial** on how to use this class.

metadata = {'render.modes': ['rgb_array', 'depth', 'segmentation', 'vector']}

logger = <Logger navsim.env.navsim_gym_env (INFO)>

reset() → Union[List[numpy.ndarray], numpy.ndarray]

Resets the state of the environment and returns an initial observation. Returns: observation (object/list): the initial observation of the space.

step(*action: List[Any]*) → Tuple[numpy.ndarray, float, bool, Dict]

Run one timestep of the environment's dynamics. When end of episode is reached, you are responsible for calling *reset()* to reset this environment's state. Accepts an action and returns a tuple (observation, reward, done, info). :param action: an action provided by the environment :type action: object/list

Returns agent's observation of the current environment reward (float/list) : amount of reward returned after previous action done (boolean/list): whether the episode has ended. info (dict): contains auxiliary diagnostic information.

Return type observation (object/list)

render(*mode='rgb_array'*) → None

Returns the image array based on the render mode

Parameters mode – 'rgb_array' or 'depth' or 'segmentation' or 'vector'

Returns each render mode returns a numpy array of the image For Observation Mode 0 and 2: render mode vector returns vector observations

Return type For Observation Mode 1 and 2

`get_navigable_map`(*resolution_x=256*, *resolution_y=256*, *cell_occupancy_threshold=0.5*) → numpy.ndarray

Get the Navigable Areas map

Parameters

- **resolution_x** – The size of the agent_x axis of the resulting grid, default = 256
- **resolution_y** – The size of the y axis of the resulting grid, default = 256
- **cell_occupancy_threshold** – If at least this much % of the cell is occupied, then it will be marked as non-navigable, default = 50%

Returns A numpy array having 0 for non-navigable and 1 for navigable cells

Note: Largest resolution is 3284 agent_x 2666

`unity_loc_to_navmap_loc`(*unity_x*, *unity_z*, *navmap_max_x=256*, *navmap_max_y=256*)

`sample_navigable_point`()

Provides a random sample of navigable point

Returns: x,y point on the navigable map

`static register_with_gym`()

Registers the environment with gym registry with the name navsim

`static register_with_ray`()

Registers the environment with ray registry with the name navsim

`get_dummy_obs`()

`get_dummy_actions`()

`property sim`

Returns an instance of the sim

Added for compatibility with habitat API.

Returns: link to self

`property unity_map_dims`

Returns the maximum x,y,z values of Unity Map

Note: While converting to 2-D map, the Z-axis max of 3-D Unity Map corresponds to Y-axis max of 2-D map

Returns: maximum agent_x,y,agent_z from unity map

`property agent_position`

Position of agent in unity map coordinates x,y,z

`property agent_velocity`

Velocity of agent in unity map coordinates x,y,z

`property agent_rotation`

Rotation of agent in unity map coordinates x,y,z,w (Quaternions)

`property agent_rotation_in_euler`

Position of agent in Euler coordinates roll_x, pitch_y, yaw_z

property goal_position
Position of goal in unity map coordinates x,y,z

property current_episode_num
Currently executing episode number, 0 means env just initialized

property last_step_num
Last executed step number, 0 mean env just initialized or reset

property shortest_path_length
Return the Shortest Path Length

Returns: Return the Shortest Path Length

5.2 Rollback Memory

class navsim.memory.**CupyMemory**(*capacity: int*, *state_shapes: Union[list, tuple, int]*, *action_shape: Union[list, tuple, int]*, *seed: Optional[float] = None*)

Bases: `navsim.memory.Memory`

append(*s: Union[list, tuple, int, float]*, *a: Union[list, tuple, int, float]*, *r: float*, *s_: Union[list, tuple, int, float]*, *d: float*)

static get_device(*self*)

info()

static load_from_pkl(*filename*)

sample(*size*)

static sample_info(*s*, *a*, *r*, *s_*, *d*)

save_to_pkl(*filename*)

static set_device(*self*, *gpu_id=0*)

class navsim.memory.**NumpyMemory**(*capacity: int*, *state_shapes: Union[list, tuple, int]*, *action_shape: Union[list, tuple, int]*, *seed: Optional[float] = None*)

Bases: `navsim.memory.Memory`

Stores and returns list of numpy arrays for s a r s_ d

assumes you are storing s a r s_ d : state action reward next_state done # state / next_state : tuple of n-dim numpy arrays # a : 1-d numpy array of floats or ints # r : 1-d numpy array of floats # d : 1-d numpy array of booleans

append(*s: Union[list, tuple, int, float]*, *a: Union[list, tuple, int, float]*, *r: float*, *s_: Union[list, tuple, int, float]*, *d: float*)

info()

static load_from_pkl(*filename*)

sample(*size*)

static sample_info(*s*, *a*, *r*, *s_*, *d*)

save_to_pkl(*filename*)

5.3 Utilities

class navsim.util.dict.**ObjDict**

Bases: `dict`

A data structure that inherits from dict and adds object style member access

clear() → None. Remove all items from D.

copy() → a shallow copy of D

deepcopy()

Make a deep copy of itself

Returns ObjDict object

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

static **load_from_file**(*filename*)

load objdict from a file

Parameters **filename** – path or name of the file

Returns ObjDict object

static **load_from_json_file**(*filename*)

load objdict from a file

Parameters **filename** – path or name of the file

Returns ObjDict object

static **load_from_yaml_file**(*filename*)

load objdict from a file

Parameters **filename** – path or name of the file

Returns ObjDict object

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

save_to_json_file(*filename*, *sort_keys=False*, *indent=2*)

Save to json file

Parameters

- **filename** – path or name of the file
- **sort_keys** – whether to sort the keys
- **indent** – indentation of the spaces

Returns:

save_to_yaml_file(*filename*)
Save to yaml file

Parameters **filename** – path or name of the file

Returns:

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

to_dict()
convert to dict

Returns dict object

to_json(*sort_keys=False*, *indent=2*)
convert to json

Parameters

- **sort_keys** – Sort the keys of dict or not, default False
- **indent** – indentation for JSON struct, default 2 spaces

Returns json string

to_yaml()
convery to yaml representation

Returns yaml string

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

Index

R

S

T

U

V