# NavSim

*1.0.7*

**Armando Fandango, Troyle Thomas**

**May 05, 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

# Pre-requisites

Following should be pre-installed on the host machine:

- nvidia driver https://github.com/NVIDIA/nvidia-docker/wiki/Frequently-Asked-Questions#how-do-i-install-the-nvidia-driver
- docker https://docs.docker.com/get-docker/
- nvidia container toolkit https://github.com/NVIDIA/nvidia-docker

CHAPTER 3

# How to use the navsim env

Assuming your code is in a folder defined in environment variable `expdir`. In your code import `NavSimGymEnv` from the `navsim` package. Either use it to instantiate env objects or extend it by subclassing.

Follow the instructions in "how to run the navsim training" section below, replace `navsim` command with your own command, for example: `my-training`

# CHAPTER 4

# How to run the navsim training

You can either run directly on a host machine or in a container.

## 4.1 Option 1: Container

1. Download and extract the unity binary zip file, and set the following, after changing first two lines for your system:

```
envdir=$(realpath "/data/work/unity-envs/Build2.6.3")
expdir=$(realpath "$HOME/exp")
envbin="Berlin_Walk_V2.x86_64"
ver="1.0.7"
repo="ghcr.io/armando-fandango"
cname="$(hostname)_navsim_1"
```

1. Run the container:

```
cd $expdir
docker run --rm --privileged -it --runtime=nvidia \
--name $cname \
-h $cname \
-e XAUTHORITY \
-e NVIDIA_VISIBLE_DEVICES=all -e NVIDIA_DRIVER_CAPABILITIES=all \
-e USER_ID=$(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 \
$repo/navsim:$ver DISPLAY=:0.0 <navsim command>
```

### 4.1.1 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 if GPU where environment binary can run.

### 4.1.2 The `<navsim command>`

- `navsim --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.

## 4.2 Option 2: TODO: Run on host directly

### 4.2.1 Fix the following parts of readme Headless Run with X-Server

Assumption: X is installed, nvidia-drivers

Install tmux (useful for persistence and shell management) (Cheat Sheet: https://gist.github.com/MohamedAlaa/2961058)

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

### 4.2.2 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 # 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 5

# General dev info:

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

# CHAPTER 6

## How to setup dev laptop to code for navsim API

- clone the ai_coop_py repo

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

- Install miniconda and the env

```
NAVSIM_ROOT=~/projects/ai_coop_py
cd $NAVSIM_ROOT/navsim_env
sudo mkdir /opt/conda
sudo chown $(id -u) /opt/conda
source ezai-conda.sh && install-miniconda
exit

NAVSIM_ROOT=~/projects/ai_coop_py
cd $NAVSIM_ROOT/navsim_env
rm -rf /opt/conda/envs/navsim; source ezai-conda.sh; ezai_conda_create --venv "/
↪opt/conda/envs/navsim"
conda activate navsim
cd $NAVSIM_ROOT
pip install -e .
```

# CHAPTER 7

## Testing from local repo

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

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

CHAPTER 8

# NavSim API

## 8.1 NavSim Environment

**class** navsim.**NavSimGymEnv**(*conf: navsim.util.dict.ObjDict*)

Bases: gym_unity.envs.UnityToGymWrapper

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

The configuration provided is as follows:

Observation Mode

- Vector 0
- Visual 1
- VectorVisual 2

Segmentation Mode

- Object Segmentation 0
- Tag Segmentation 1
- Layer Segmentation 2

Task

- PointNav 0
- SimpleObjectNav 1
- ObjectNav 2

Goal

- 0 - Tocus
- 1 - sedan1
- 2 - Car1

- 3 - Car2
- 4 - City Bus
- 5 - Sporty_Hatchback
- Else - SEDAN

```
env_conf = ObjDict({
    "log_folder": "unity.log",
    "seed": 123,
    "timeout": 600,
    "worker_id": 0,
    "base_port": 5005,
    "observation_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"]
})
```

Action Space: [Throttle, Steering, Brake]

- Throttle: -1.0 to 1.0
- Steering: -1.0 to 1.0
- Brake: 0.0 to 1.0

Observation Space: [[Raw Agent Camera],[Depth Agent Camera],[Segmentation Agent Camera],[Agent Position, Agent Velocity, Agent Rotation, Goal Position]] 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

**close**() → None
Override _close in your subclass to perform any necessary cleanup. Environments will automatically close() themselves when garbage collected or when the program exits.

**get_navigable_map**() → numpy.ndarray
Get the Navigable Areas map

**Parameters**

- **resolution_x** – The size of the x axis of the resulting grid, default = 200
- **resolution_y** – The size of the y axis of the resulting grid, default = 200
- **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 which has 0 for non-navigable and 1 for navigable cells

**Note:** This method only works if you have called `reset()` or `step()` on the environment at least once.

**Note:** Largest resolution that was found to be working was 2000 x 2000

**info**()
Prints the information about the environment

**info_steps**(*save_visuals=False*)
Prints the initial state, action sample, first step state

**property metadata**
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

**dict(iterable) -> new dictionary initialized as if via:** d = {} for k, v in iterable:

d[k] = v

**dict(**kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

**property observation_space_shapes**
Returns the dimensions of the observation space

**property observation_space_types**
Returns the dimensions of the observation space

**render**(*mode=''*) → None
Not implemented yet

**Parameters** **mode** –

Returns:

**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.

**property reward_range**
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

**seed**(*seed: Optional[Any] = None*) → None
Sets the seed for this env's random number generator(s). Currently not implemented.

**property sim**
Returns itself

Added for compatibility with habitat API.

Returns: link to self

**start_navigable_map**(*resolution_x=200*, *resolution_y=200*, *cell_occupancy_threshold=0.5*)
Get the Navigable Areas map

**Parameters**

- **resolution_x** – The size of the x axis of the resulting grid, default = 200
- **resolution_y** – The size of the y axis of the resulting grid, default = 200
- **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 which has 0 for non-navigable and 1 for navigable cells

---

**Note:** This method only works if you have called `reset()` or `step()` on the environment at least once.

---

---

**Note:** Largest resolution that was found to be working was 2000 x 2000

---

**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)

**property unwrapped**
Completely unwrap this env.

**Returns** The base non-wrapped gym.Env instance

**Return type** gym.Env

## 8.2 NavSim Utilities

**class** navsim.util.**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