NavSim

2.10.10

STTC, UCF

Aug 20, 2021

Contents

CHAPTER 1

Introduction

A navigation simulator (navsim) API built on top of Python, Pytorch.

In the future, navsim may be compatible with a variety of simulators, but for now it uses A Realistc Open environment for Rapid Agent training(ARORA) Simulator, that is a highly attributed Unity3D GameEngine based Berlin city environment.

1.1 Versions

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

This version of navsim has been tested with latest release of ARORA binary is 2.10.7.

CHAPTER 2

navsim_envs Package Tutorial

navsim_envs is a package that wraps the environments for Reinforcement Learning. Current implementation has the following sims encapsulated:

- ARORA from UCF

2.1 Installation

pip install --upgrade navsim_envs

Note: We assume that you have a virtual env activated. If not, please create a conda env with `conda create -n navtest python=3.8` and then activate it with `conda activate navtest`.

If you would like to create our optimized conda env for navsim, then refer to the navsim conda and container page.

2.1.1 Test the installation

In order to test the install:

- Create an experiment folder, we use folder `~/exp` and change into this folder.
- Create a minimal yaml file with the following contents:

```
env_path: /path/to/Berlin_Walk_V2.x86_64
```

  Let us say you named this file `min_env_config.yml`.

- Run the tests: `navsim_env_test min_env_config.yml`. If you are using `navsim` container then follow the instructions on the container page.

2.2 How to use the ARORA env

If you only want to use `AroraGymEnv`, then either subclass it or use it as follows:

```
import navsim_envs
from navsim_envs.env.arora import AroraGymEnv, default_env_config
import gym

env_config = default_env_config.copy()

# or

env_config = {
  "agent_car_physics": 0,
  "debug": False,
  "episode_max_steps": 1000,
  "env_gpu_id": 0,
  "env_path": "/path/to/Berlin_Walk_V2.x86_64",
  "goal": 0,
  "goal_distance": 50,
  "log_folder": "./env_log",
  "obs_mode": 0,
  "obs_height": 256,
  "obs_width": 256,
  "seed": 123,
  "start_from_episode": 1,
  "reward_for_goal": 50,
  "reward_for_no_viable_path": -50,
  "reward_step_mul": 0.1,
  "reward_collision_mul": 4,
  "reward_spl_delta_mul": 1,
  "save_actions": True,
  "save_vector_obs": True,
  "save_visual_obs": True,
  "show_visual": False,
  "task": 0,
  "timeout": 600,
  "traffic_vehicles": 0,
}

# Either use gym.make to create an env
env = gym.make("arora-v0", env_config=env_config)
# or use the AroraGymEnv constructor to create an env
env = AroraGymEnv(env_config)
```

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

2.3 Config Parameters

TODO: Explain all the above parameters here from config dictionary

2.3.1 Observation Mode

Sets the return mode of observations to one of the following. The observations themselves are described in the Observation Space heading in next subsection.

- `0` - Vector only
- `1` - Visual only
- `2` - VectorVisual - Vector and Visual

2.3.2 Segmentation Mode

- `0` - Object Seg: Each 'class' of 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)

2.3.3 Task

- `0` - PointNav - The 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.

2.3.4 Goal : Only relevant for SimpleObjectNav and ObjectNav

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

2.3.5 Rewards

Reward Values

- `reward_for_goal` : For pointnav, the goal is the target position to complete the task.
- `reward_for_ep` : Exploration points are randomly placed in the environment to reward exploration.
- `reward_collision_mul` : This reward multiple is used to determine the reward upon collision are anything that is not a goal point or exploration point. This includes other cars, building, trees, etc.
- `reward_for_no_viable_path` : The map is a tiled specified bounded by the values of env.unity_map_dims(). If the agent goes outside of this area and falls -15m below the environment area or enters an area outside of the navigable area then this reward is activated. This will also result in a reset.
- `reward_step_mul` : This reward multiplier is used to determine the rewards given at every step in addition to any other reward recieved at the same step.
- `reward_spl_delta_mul` : This reward multiplier is used to determine the reward as the agent reduces the current SPL to the goal

Reward Specifications

```
spl_delta = spl_prev_step - spl_current_step
#If spl_delta is positive: the agent is closer to the goal according to spl
#If spl_delta is negative: the agent is further away from the goal according to spl

spl_reward = -(1 * reward_spl_delta_mul)  if delta==0 else spl_delta * reward_spl_delta_
↪mul
step_reward = -(reward_for_goal / start_spl)  * reward_step_mul
collision_reward = reward_collision_mul * step_reward

if `agent reached goal`:
    total_reward = goal_reward

elif `agent has no viable path`:
    total reward = -no_viable_path_reward

else:
    total_reward = spl_reward + step_reward + collision_reward
```

2.3.6 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

2.4 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` : -1.0 to 1.0 : Reduces the agents current velocity

2.4.1 Car motion explanation based on action space

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.
- Individual Actions:
  - `[1 to -1, 0, 0]` - Throttle is applied - forward for +ve, and backward for -ve values. The agent travels at (max_velocity/throttle_value) for the duration of the step. Throttle is only valid for current step and is not remembered in next step.
  - `[0, -1 to 1, 0]` - Steering is applied - left-turn for -ve, and right-turn for +ve values. No throttle or braking is applied, speed is zero, there is no travel.
    - For relative steering mode, the steering turn is remembered in next step.
    - For absolute steering mode, this is a useless operation.
  - `[0, 0, 1]` - Full braking is applied, with no throttle or steering. No agent travel. The breaking is not remembered in next step, thus this is a useless operation.
- Combining Actions
  - `[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. Thus, this is a useless operation.
  - `[1, 0, 0.5]` - Full forward throttle and half braking are applied. The agent travels forward at half throttle.

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,
Proximity_Forward, Proximity_45_Left, Proximity_45_Right]
```

- `Proximity_*` refers to the navigable / clear space before the agent collides with another object

2.5.2 The visual observation space

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

CHAPTER 3

navsim_envs API

3.1 env classes

class `navsim_envs.arora.`**`AroraGymEnv`**(*env_config*)

Bases: `gym_unity.envs.UnityToGymWrapper`

AroraGymEnv inherits from 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_envs.arora.arora_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'*`)`

Returns the image array based on the render mode

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

Returns For Observation Mode 1 and 2 - each render mode returns a numpy array of the image For Observation Mode 0 and 2 - render mode vector returns vector observations

`get_agent_obs_at`(*position: Optional[List[float]] = None*, *rotation: Optional[List[float]] = None*)
Get the agent observations at position or rotation

Parameters

- **position** – a list of x,y,z in Unity's coordinate system
- **rotation** – a list of x,y,z,w in Unity's coordinate system

If the position or rotation is not provided as argument, then it takes them from the current state.

Returns Observations if possible, else None

`set_agent_state`(*position: Optional[List[float]] = None*, *rotation: Optional[List[float]] = None*)
Set the agent position or rotation

Parameters

- **position** – a list of x,y,z in Unity's coordinate system
- **rotation** – a list of x,y,z,w in Unity's coordinate system

If the position or rotation is not provided as argument, then it takes them from the current state.

Returns True if the state is set, else False

`set_agent_position`(*position: Optional[List[float]]*)
Set the agent position

Parameters **position** – a list of x,y,z in Unity's coordinate system

If the position is not provided as argument, then it takes them from the current state.

Returns True if the state is set, else False

`set_agent_rotation`(*rotation: Optional[List[float]]*)
Set the agent rotation

Parameters **rotation** – a list of x,y,z,w in Unity's coordinate system

If the rotation is not provided as argument, then it takes them from the current state.

Returns True if the state is set, else False

`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, 1 to 3276, default = 256
- **resolution_y** – The size of the y axis of the resulting grid, 1 to 2662, default = 256
- **cell_occupancy_threshold** – If at least this much % of the cell is occupied, then it will be marked as non-navigable, 0 to 1.0, default = 50%

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

Note: Largest resolution is 3284 x 2666

`unity_to_navmap_location`(*unity_x*, *unity_z*, *navmap_max_x=256*, *navmap_max_y=256*)
Convert a location from Unity's 3D coordinate system to navigable map's 2D coordinate system

Parameters

- **unity_x** – x coordinate in unity
- **unity_z** – z coordinate in unity
- **navmap_max_x** – maximum x of navmap
- **navmap_max_y** – maximum y of navmap

Returns navmap_x, navmap_y

navmap_to_unity_location(*navmap_x*, *navmap_y*, *navmap_max_x=256*, *navmap_max_y=256*, *navmap_cell_center=True*)

Convert a location from navigable map's 2D coordinate system to Unity's 3D coordinate system

Parameters

- **navmap_x** – x, y location on navmap
- **navmap_y** – x, y location on navmap
- **navmap_max_x** – maximum x,y on navmap
- **navmap_max_y** – maximum x,y on navmap
- **navmap_cell_center** – Whether to return the point in cell center, default True.

Returns unity_x, unity_z

unity_to_navmap_rotation(*unity_rotation: List[float]*)

Convert a rotation from Unity's quarternion to navigable map's 2D coordinate system

Parameters **unity_rotation** – [x,y,z,w] in unity quaternion system

Returns [x,y] vector components of rotation

unity_rotation_in_euler(*unity_rotation: Optional[List[float]] = None*)

Position of agent in Euler coordinates roll_x, pitch_y, yaw_z

Parameters **unity_rotation** – [x,y,z,w] in unity quaternion system

Returns pitch_y, yaw_z, roll_x

navmap_to_unity_rotation(*navmap_rotation: List[float]*)

Convert a rotation from navigable map's 2D coordinate system to Unity's quarternion

Parameters **navmap_rotation** – x,y vector components of rotation

Returns [x,y,z,w] Unity's quarternion

sample_navigable_point(*x: Optional[float] = None*, *y: Optional[float] = None*, *z: Optional[float] = None*)

Provides a random sample of navigable point

Parameters

- **x** – x in unity's coordinate system
- **y** – y in unity's coordinate system
- **z** – z in unity's coordinate system

Returns

If x,y,z are None, returns a randomly sampled navigable point [x,y,z].

If x,z is given and y is None, returns True if x,z is navigable at some ht y else returns False.

If x,y,z are given, returns True if x,y,z is navigable else returns False.

`is_navigable`(*x: float*, *y: float*, *z: float*) → bool

Returns if the point is navigable or not

Parameters

- **x** – the unity coordinates of the point to check
- **y** – the unity coordinates of the point to check
- **z** – the unity coordinates of the point to check

Returns True if the point represented by x,y,z is navigable else False

`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`()

returns dummy observations

Returns:

`get_dummy_actions`()

returns dummy actions

Returns:

`property sim`

Returns an instance of the sim

`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

`property agent_obs`

Agent observations

`property agent_position`

Position of agent in unity coordinates x,y,z

`property agent_rotation`

Rotation of agent in unity quaternions x,y,z,w

`property agent_velocity`

Velocity of agent in unity coordinates x,y,z

`property goal_position`

Position of goal in unity 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`

the shortest navigable path length from current location to goal position

`class navsim_envs.arora.AroraUnityEnv`(*env_config*)

Bases: `mlagents_envs.environment.UnityEnvironment`

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

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

`logger = <Logger navsim_envs.arora.arora_unity_env (INFO)>`

CHAPTER 4

ARORA actions playback (AAR Mode)

The AAR mode uses the vector observation files created by the `AroraGymEnv`. Ensure that you execute `ArooraGymEnv` with the `save_vector_obs` set to `True`, this will store a file called `env_logs/vec_obs.csv`.

1. Execute the berlin executable with the `-playback` switch.

2. Type in the full path to the `vec_obs.csv` file into the text input field. Either type a return at the end, or click the load icon. 3. See the UI guidelines for further details.

4.1 UI Guidelines

1. The full path to the observation file should be entered in this field.
2. Loads the observation file listed in 1. This should only be done once.
3. Play – plays the episode in 7, at the rate specified by the playback rate
4. Rewind+, reduces the playback speed by 1x.
5. FF+, reduces the playback speed by 1x.
6. Increase or decrease the agent step by a single step.
7. Jump to an episode number, or step number. Jumping to a new episode number will cause the local scene to be (re)loaded.
8. Keystoke legend: T- Toggles mouse controls for the viewpoint. R- Resets the camera to a relative location. L/R Arrow Key- Scrub the scene forward or backward. Space Bar- Pause or Resume the AAR replay. Tab/H Key- Hides or reveals this UI.
9. Tile radius- potentially provides improvement in performance or provides increased environment visibility depending on the terrain tile radius entered.

CHAPTER 5

navsim conda env or container

5.1 Pre-requisites

Following should be pre-installed on the host machine:

5.1.1 For running inside the containers

- nvidia driver
- docker
- nvidia container toolkit

5.1.2 For directly running on the host

- nvidia driver
- X-window system

5.2 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=$(realpath "/data/work/unity-envs/Build2.10.5");
envbin="Berlin_Walk_V2.x86_64";
expdir=$(realpath "$HOME/exp");
```

(continues on next page)

(continued from previous page)

```
run_id="demo";
repo="ghcr.io/armando-fandango";
cd $expdir
```

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

5.2.1 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:2.10.7 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.

5.2.2 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`

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

4. Optional: Install jupyter in `navsim`

```
conda activate navsim && source ezai-conda.sh && install_jupyter
```

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.

5.2.3 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.

5.3 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

5.4 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 $HOMEfolder - B < absolutepathofcurrentfolder > notneededifpathtocurrentfolderisinside$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 6

Contributing to NavSim API

6.1 General dev info:

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

6.2 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 conda environment on the host. Activate the navsim conda environment.
- `pip install -e /path/to/ai_coop_py/navsim-envs`
- `pip install -e /path/to/ai_coop_py/navsim-lab`

6.3 to modify and build the docs:

1. Checkout -docs branch
2. Make sure you have navsim conda env activated and local repo installed with pip.
3. Modify the .md files in /path/to/navsim-lab, /path/to/ai_coop_py, /path/to/navsim-envs
4. go to ai_coop_py/docs
5. run make html latexpdf. If it results in error, just run again with order reversed: make latexpdf html. If still errors, please call me and I will help fix
6. The pdf and html docs would be dumped in the ai_coop_py/docs from where you ran the make.

7. If you are happy with the PDF/formatting etc then commit and push the doc branch back

6.4 Testing from local repo

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

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

CHAPTER 7

NavSim API

7.1 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`

static **`get_device`**(*self*)

static **`load_from_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

CHAPTER 8

navsim_util API

Index

U