AUDRI Documentation

Author

May 14, 2018

Contents:

CHAPTER 1

audri module

Provides functionality for automated driving. AUDRI is able to receive feature vectors, learn from them, and then decide upon an action to perform when presented with another feature vector

class audri.**Agent**

Bases: `object`

A learning agent who will learn how to drive from data it is given, and is able to use its training to command a car

action(*stateVector*)

Return the predicted label for the given state vector

Parameters **stateVector** – (`dict`) The unlabeled state vector

load(*name*)

save(*name*)

train(*data*)

Train the agent using a set of data

Parameters **data** – (`list`) A list of `dict` state vectors

CHAPTER 2

config module

Configuration classes for all of the components

class config.**GUIConfig**

Bases: `object`

Configuration used by the *gui* module

`tuple` attributes store main text at index 0, and tooltip text at index 1

CompareText = 'Compete with AUDRI'

(`str`) Text for the button in the *MainFrame*, linking to the *Simulator* in compare mode

ConfAppearance = 'Appearance'

(`str`) Header text for the appearance parameters section in the *ConfigFrame*

ConfBackground = ('Scroll background', 'Whether the background should scroll down the v

ConfCarScale = ('Car scale', 'Multiplied against the normal size of the car sprite')

ConfCarSpeed = ('Car speed', '(metres per second)\nThe forward speed of the main car')

ConfExperiment = 'Experiment parameters'

(`str`) Header text for the experimental parameters section in the *ConfigFrame*

ConfFPS = ('Frames per second', 'Frame rate - how many times per second the screen is r

ConfInnerPadding = 15

(`int`) Padding between labels and their connected control in the *ConfigFrame*

ConfObstFreq = ('Obstacle frequency', '(seconds)\nTime between obstacles spawning')

ConfObstScale = ('Obstacle scale', 'Multiplied against the normal size of the obstacle

ConfObstSpeed = ('Obstacle speed', '(metres per second)\nThe forward speed of the obsta

ConfOffroad = ('Allow the car to drive offroad', 'If checked, the main car will be able

ConfRandomSeed = ('Random seed', 'The value used to control pseudo-random number genera

ConfRecordFreq = ('Snapshot interval', 'Gap in seconds between recordings of the state

ConfRowPadding = 10

ConfSave = ('Save', 'Save configuration and return to the main menu')

ConfText = 'Configuration'
(str) Text for the button in the *MainFrame*, linking to the *ConfigFrame*

ConfTickrate = ('Tickrate', 'Tick rate - how many times per second the game logic shou

ConfTitle = 'Configuration'
(str) Header text in the *ConfigFrame*

DataText = 'Gather training data'
(str) Text for the button in the *MainFrame*, linking to the *Simulator* in training mode

Font = {'family': 'sans-serif', 'size': 10}
(dict) Data used to create a normal font

FontBold = {'family': 'sans-serif', 'size': 10, 'weight': 'bold'}
(dict) Data used to create a bold font

FontHeading = {'family': 'sans-serif', 'size': 15, 'weight': 'bold'}
(dict) Data used to create a header font

FontMonospace = {'family': 'monospace', 'size': 10}
(dict) Data used to create a monospaced font

Height = 600
(int) Height of the main window

MainButtonXPad = 15

MainButtonYPad = 15

ModeText = ['manual', 'AUDRI', 'compare']
list of str mapping simulator modes to a description of that mode, used by SimulatorPanel

PanelWidth = 220
(int) Width of the Panel

SimPopupDupWarn = 'This name is already in use, overwrite?'
Duplicate warning message in (*NameDatasetPopup*)

SimPopupPad = 10
Padding in (*NameDatasetPopup*)

SimPopupText = 'Choose a name for this training set'
Message in (*NameDatasetPopup*)

SimPopupTitle = 'Name the training set'
Window title of the popup for naming a training set (*NameDatasetPopup*)

Subtitle = 'Teaching a computer to drive through example'
(str) Subtitle shown in the *MainFrame*

TestText = 'Test AUDRI'

Theme = 'clam'
(str) Tkinter theme to use

Title = 'AUDRI Simulator'
(str) Main title shown in the titlebar and the *MainFrame*

TooltipWidth = 250

VisualiserWidth = 500
(int) Width of the *visualiser.visualiser.SimulatorVisualiser*

class config.**SimulatorConfig**
Bases: object

Configuration used by the *visualiser* class

CarScale = 0.35
(float) Scale of the car's sprite

CarSpeed = 11
(float) Speed of the main car in metres per second

FPS = 70
(int) Targeted frames per second to be drawn

LaneWidth = 86
(int) Width of a lane

ObstacleInterval = 2
(float) Seconds between spawning obstacles

ObstacleSpeed = 8

(float) Speed of obstacle vehicles in metres per second

Should be less than *CarSpeed*

OffroadAllowed = False
(bool) Whether the car can go offroad

OffroadWidth = 120
(int) Width of a an offroad lane

PixelMetreRatio = 50
(float) Ratio of pixels to metres, affects the visualisation of speeds

RandomSeed = 'geqJQD6MfJ'
Random seed used to influence when vehicles spawn Can be int, str, and others. See random.seed() for details

RecordInterval = 0.2
(float) Seconds between each snapshot

ScrollBackground = True
(bool) Whether the background should move

TickRate = 5
(int) Targeted milliseconds between each simulation tick

class config.**Singleton**
Bases: type

Metaclass that provides singleton behaviour

Author Adam Forsyth <adam@adamforsyth.net>

Source https://stackoverflow.com/a/6798042

_instances = {}
(dict) stores the instances of each class

CHAPTER 3

data module

Provides functionality for manipulating training data sets and models

data.**DATA_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/audri/checkouts/latest/s**

(str) Path to store data in

data.**DATA_ORDER = ['action', 'aheadDistance', 'currentLane']**

list specifying the order of data

data.**MODEL_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/audri/checkouts/latest/s**

(str) Path to store models in

data.**dataExists**(*name*)

Parameters **name** – (str) The name of the data set to search for

Returns bool indicating if a data set named *name* exists

data.**loadData**(*name*)

Parameters **name** – (str) The name of the data set to load

Returns dict containing the training data

Returns dict containing the experimental settings used

data.**modelExists**(*name*)

Parameters **name** – (str) The name of the model to search for

Returns bool indicating if a model named *name* exists

data.**saveData**(*name*, *data*)

Save a data set to a file with a given name.

Additionally saves a metadata file containing the experimental settings used during the training

Parameters

- **name** – (str) The name of the data set to save

- **`data`** – (`iter` of `dict`) 2-dimensional data structure, each list entry is a feature vector.

  Each feature vector is an associative (named) array of features

CHAPTER 4

gui package

4.1 Submodules

4.2 gui.conf module

Provides ConfigFrame, the Tkinter frame that allows modifying the visualiser configuration

class `gui.conf.`**`ConfigFrame`**(*root*, *main*)

Bases: `tkinter.Frame`

Frame allowing the configuration of the visualiser

Contains a number of Tkinter controls allowing the modification of attributes in *`SimulatorConfig`*

`_labelControl`(*text*, *control*, *ctrlOpts={}*)

Create a label and a widget, which are placed adjacent on the same row.

Parameters

- **`text`** – `str` to show in the label
- **`control`** – `object` class of the control to create
- **`ctrlOpts`** – `dict` of kwargs to use when constructing the control.

Returns The created `Widget` control

`_load`()

Load values into the controls from the *`SimulatorConfig`*

`_matches = {'CarScale': 'carScale', 'CarSpeed': 'carSpeed', 'ObstacleInterval': 'ob`

`SimulatorConfig` attributes as keys, associated private attributes (with a set() method) as attributes. These config values are all `float`

`_save`()

Save values into the *`SimulatorConfig`* from the controls

4.3 gui.controls module

Provides custom Tkinter controls

class gui.controls.**LabeledScale**(*root*, *font*, *resolution=2*, ***kwargs*)

Bases: `tkinter.Frame`

`tkinter.ttk.Scale` and a `tkinter.Spinbox` joined in a frame

The `Scale` in shown to the left of the `Spinbox`

Parameters

- **root** – Parent `tkinter.Widget`
- **font** – `Font` to use in the `Spinbox`
- **resolution** – (`int`) Number of decimal places to round stored and displayed value to

_update(*val=None*)

Callback method for both the `Spinbox` and `Scale` so that each can update the other, and the value can be properly formatted

get()

Returns `float` - Value stored, rounded to specified resolution number of decimal places

set(*val*)

Set value of both the `Spinner` and `Scale`

Parameters **val** – `float` to set values to

class gui.controls.**ToolTip**(*widget*, *text=”*, *delay=500*, *width=250*)

Bases: `object`

Creates a tooltip that appears above the given widget when hovered

Authors

- **Fuzzyman:** http://voidspace.org.uk/python/weblog/arch_d7_2006_07_01.shtml#e387
- **vegaseat:** https://daniweb.com/programming/software-development/code/484591
- **crxguy52:** https://stackoverflow.com/a/36221216

Parameters

- **widget** – `tkinter.Widget` to bind to
- **text** – (`str`) Text content of the tooltip
- **delay** – (`int`) Milliseconds before displaying tooltip on hover
- **width** – (`int`) Maximum width of tooltip in characters

_cursorPos(*widget*)

_enter(*event=None*)

_hide()

_leave(*event=None*)

_schedule()

_show(*event=None*)

_unschedule()

4.4 gui.main module

The main window and main page which links to the others

class gui.main.**MainFrame**(*root*, *main*)

Bases: tkinter.Frame

Main 'page' of the application, linking to the *Simulator* and *ConfigFrame* using Button widgets. These cause the *MainWindow* to focus on the desired frame

Parameters

- **root** – Widget parent
- **main** – *MainWindow*

startSim(*mode*)

class gui.main.**MainWindow**(**args*, ***kwargs*)

Bases: tkinter.Tk

The main Tkinter window of the application

Encapsulates *Simulator* and *ConfigFrame*

_keyPress(*event*)

Tkinter key press callback method, sends events to the *Simulator* if it is currently focused

_simulator

The encapsulated *Simulator*

Getter Get the simulator

Type Widget

_tick()

Run *tick()* on the *Simulator*

Repeatedly calls itself in intervals of *TickRate* milliseconds while the Simulator window remains focused

back()

Return to the main menu by raising the *MainFrame* to the top

focus(*name*)

Focus on a specified contained frame. Starts the tick cycle by calling *_tick()* if focusing on the *Simulator*

Parameters **name** – (str) Class name of the encapsulated frame that should be focused

One of 'MainFrame', 'ConfigFrame'

4.5 gui.panel module

Provides the side panel shown alongside the visulaiser

class gui.panel.**SimulatorPanel**(*parent*, *sim*, *visualiser*, ***kwargs*)

Bases: tkinter.Frame

Side panel of the window, extends tk.Frame. Shows information about the visulisation.

Parameters

- **parent** – Parent `tkinter.Widget`
- **sim** – *Simulator* instance to control
- **visualiser** – *SimulatorVisualiser* instance to take information from
- **kwargs** – keyword arguments to pass to superclass constructor

_call (*func*)

Return a function that focuses the panel then calls *func*

Used for button commands

_makeLabel (*name*, *text*, *row*)

Create and attach multiple labels to the given row

Parameters

- **name** – `str` used in property names
- **text** – `str` to use as label
- **row** – `tkinter.Widget` to attach to

_newRow (**kwargs*)

Create a new row to be used

Returns The new `Frame`

tick ()

Update displayed information using the *SimulatorVisualiser*

4.6 gui.sim module

Provides the main simulator, which includes the Pygame visualiser and a Tkinter panel

class gui.sim.**NameDatasetPopup** (*main*)

Bases: `tkinter.Toplevel`

A Tkinter popup that requests a name for a training set.

It will check if the name is in use; if it is, another prompt asks whether it should be overriden or another name should be input.

accept (**args*)

Close the popup and set the input value on the *Simulator*

class gui.sim.**Simulator** (*root*, *main*)

Bases: `tkinter.Frame`

The GUI component of the visualiser Contains the `SimulatorVisualiser` (Pygame canvas) in a

`tkinter.ttk.Frame`. The `SimulatorPanel` is displayed alongside it.

finish ()

Stop the visualiser and return to the main menu

focus ()

If loaded in manual mode, create a *NameDatasetPopup* to choose the training data name.

If in AUDRI mode, train the model

keyPress (*event*)

Pass key press events to visualiser

`mode`

The current mode in which the simulator should run in.

0 = Manual: Collect training data

1 = AUDRI: Train AUDRI and let it control the car

2 = Compare: Allow both expert and AUDRI to control two different cars, side by side

Getter Get the current mode

Setter Set the current mode, also setting the mode of the *`SimulatorVisualiser`*. Prevents setting an invalid mode

Type `int`

`restart``()`

Reset random seed and restart visualiser using *`reset()`*

`setDataset``(`*name*`)`

Set the `_datasetName` property from the *`NameDatasetPopup`*, unpause the *`SimulatorVisualiser`*, and return focus to the main window

Parameters **`name`** – `str` name of dataset to store training data in

`tick``()`

Call visualiser and panel tick frequently

CHAPTER 5

visualiser package

Provides the visualiser modules

class visualiser.**SimulatorVisualiser**(*windowID*)

Bases: object

The visualiser of the simulator, showing a graphical representation of the highway.

Parameters **windowID** – str set as the SDL_WINDOWID environment variable. Used to embed the pygame window into the GUI.

_backgrounds = None

list storing the active BackgroundPiece

_bgHeight = None

(int) Height of the background sprite

_cachedSprites = None

(boolean) Whether all sprites have been cached

_fps = None

(int) Frames per second target

_lastAction = None

(*Actions*) last action performed

_lastDraw = None

(float) Timestamp of the last time the screen was drawn

_lastSpawn = None

(int) Last timestamp an obstacle vehicle was spawned

_lastTick = None

(float) Timestamp of the last *tick()* call

_obstacles = None

list storing the currently spawned *Obstacle* instances

_targetDrawDelay = None

(float) Targeted milliseconds in between screen draws

agentCar = None
The `Car` controlled by the agent

agentCollisions = None
`int` indicating the number of agent collisions in the current run

canvas = None
A reference to the `Surface` for drawing

car = None
The `Car` controlled by the expert

collisions = None
`int` indicating the number of expert collisions in the current run

distanceTravelled = None
`int` indicating the number metres travelled in the current run

doAct(*act*, *agent=False*)
Perform Action act

draw()
Updates the canvas Attempts to achieve target FPS by blocking As such, it should run in its own thread so other things can be done in the background Should run in and endless loop to continuously redraw

fps
The current target frames per second

Getter Get the current FPS

Setter Set the current FPS. Calculates the necessary target draw interval

Type `int`

keyPress(*key*)
Handle key presses and perform the actions they map onto

lastActionTime = None
`float` Timestamp when the last action was performed

mode
The mode of the visualiser. See `mode` for details.

Getter Return the current set mode

Setter Set the current mode. Also sets the correct dimensions of the display, doubling the width if in compare mode

Type `int`

pause = None
`bool` indicating whether the visualiser is paused

reset()
Restart the visualiser by resetting properties

sessionTime = None
`float` indicating the milliseconds since starting the current run

stateVector(*agent=False*)
Return a dictionary of features: {last action, current lane, distance of obstales in three lanes, offroad} Some features have been 'pruned' from our descision tree because they did not affect the accuracy of the tree. These have been commented out for clarity.

tick()

This method tries to run at regular intervals of *TickRate* milliseconds. Performs update logic of the cars, obstacles, and background using time since the previous tick. Stores the current tick timestamp in *_lastTick*

togglePause()

Toggle pause state of the game

5.1 Submodules

5.2 visualiser.util module

Utility functions and classes

class visualiser.util.**Actions**

Bases: `enum.Enum`

Enumerations of the actions that the simulator can receive

LEFT = 2

Move into the lane to the left

NONE = 1

No action

PAUSE = 4

Toggle the pause state of the simulator

RIGHT = 3

Move into the lane to the right

class visualiser.util.**Pos**(*vehicle*, *x=0*, *y=0*)

Bases: *visualiser.util.Vector*

2D vector that automatically updates its vehicle's rect position

x

The x component of the vector :getter: Return x value :setter: Set x value, rounded to nearest `int`

y

The y component of the vector :getter: Return y value :setter: Set y value, rounded to nearest `int`

class visualiser.util.**Vector**(*x=0*, *y=0*)

Bases: `object`

A 2D vector, with arithmetic magic methods implemented

x = None

The x component of the vector

y = None

The y component of the vector

visualiser.util.**loadSprite**(*path*, *scale=1*)

Load an image as a `pygame.Surface`. Automatically caches loaded images in the background

Parameters

- **path** – (`str`) Path to desired image, relative to the sprites/ directory
- **scale** – (`float`) Multiplied against the size of the image in order to scale as desired

5.3 visualiser.vehicles module

Vehicle classes used by the game

class visualiser.vehicles.**Car**(*game*)

Bases: *visualiser.vehicles.Vehicle*

The controlled car

Parameters **game** – The parent *SimulatorVisualiser*

tick(*dt*)

Car tick method, currently does nothing

class visualiser.vehicles.**Obstacle**(*game*)

Bases: *visualiser.vehicles.Vehicle*

An obstacle vehicle that the main car has to avoid

Parameters **game** – The parent *SimulatorVisualiser*

_agentCollided = None

bool indicating whether this vehicle has collided with the agent car

_collided = None

bool indicating whether this vehicle has collided with the user car

hasCollided

Getter Return whether the vehicle has just collided this tick. Updates *_collided*

hasCollidedAgent

Getter Returns whether the vehicle has started colliding with agent car this tick. Updates *_agentCollided*

speed

The virtual forward speed of the vehicle, in metres per second

Getter Get the vehicle speed

Setter Set the vehicle speed in m/s. Also sets the real velocity, which is relative to the user car

tick(*dt*)

Called regularly to perform update logic. Return False to indicate vehicle is entirely off screen and should be removed

visualiser.vehicles.**SPRITE_SCALES = {'car.png': 0.4, 'lorry.png': 0.75, 'police.png': 0**

dict mapping obstacle sprites onto their respective float scales

class visualiser.vehicles.**Vehicle**(*game*)

Bases: object

A vehicle in the game

Parameters **game** – The parent *SimulatorVisualiser*

_sprite = None

str name of the sprite file, relative to the sprites/obstacles directory

_spriteScale = None

(float) Size multiplier of the sprite

colliding(*veh*)

Return whether this vehicle is colliding with Vehicle veh

draw(*canvas*)
Draw the vehicle

lane
The current lane of the vehicle

Getter Get the current lane

Setter Set the lane of the vehicle. The vehicle is centered in its lane. Will only accept lanes available

Type `int`

pos
The position vector of the vehicle

Getter Get position

Setter Set the position

Type *`Pos`*

rect = None
A reference to the vehicle's `Rect`

speed
The virtual forward speed of the vehicle, in metres per second

Getter Get the vehicle speed

Setter Set the vehicle speed in m/s

sprite
The sprite object. Uses *`_sprite`* as the file name of the image to load

Getter Return the sprite

Type `pygame.Surface`

tick(*dt*)
Called regularly to perform update logic. Updates vehicle position using its velocity

5.4 visualiser.visualiser module

The visualiser of the simulator - the game the user plays Uses the PyGame library

class `visualiser.visualiser.`**SimulatorVisualiser**(*windowID*)
Bases: `object`

The visualiser of the simulator, showing a graphical representation of the highway.

Parameters **windowID** – `str` set as the SDL_WINDOWID environment variable. Used to embed the pygame window into the GUI.

_backgrounds = None
`list` storing the active `BackgroundPiece`

_bgHeight = None
(`int`) Height of the background sprite

_cachedSprites = None
(`boolean`) Whether all sprites have been cached

_fps = None
(`int`) Frames per second target

_lastAction = None
(*`Actions`*) last action performed

_lastDraw = None
(`float`) Timestamp of the last time the screen was drawn

_lastSpawn = None
(`int`) Last timestamp an obstacle vehicle was spawned

_lastTick = None
(`float`) Timestamp of the last *`tick()`* call

_obstacles = None
`list` storing the currently spawned *`Obstacle`* instances

_targetDrawDelay = None
(`float`) Targeted milliseconds in between screen draws

agentCar = None
The *`Car`* controlled by the agent

agentCollisions = None
`int` indicating the number of agent collisions in the current run

canvas = None
A reference to the `Surface` for drawing

car = None
The *`Car`* controlled by the expert

collisions = None
`int` indicating the number of expert collisions in the current run

distanceTravelled = None
`int` indicating the number metres travelled in the current run

doAct(*act*, *agent=False*)
Perform Action act

draw()
Updates the canvas Attempts to achieve target FPS by blocking As such, it should run in its own thread so other things can be done in the background Should run in and endless loop to continuously redraw

fps
The current target frames per second

Getter Get the current FPS

Setter Set the current FPS. Calculates the necessary target draw interval

Type `int`

keyPress(*key*)
Handle key presses and perform the actions they map onto

lastActionTime = None
`float` Timestamp when the last action was performed

mode
The mode of the visualiser. See *`mode`* for details.

Getter Return the current set mode

Setter Set the current mode. Also sets the correct dimensions of the display, doubling the width if in compare mode

Type `int`

`pause = None`
`bool` indicating whether the visualiser is paused

`reset``()`
Restart the visualiser by resetting properties

`sessionTime = None`
`float` indicating the milliseconds since starting the current run

`stateVector``(`*agent=False*`)`
Return a dictionary of features: {last action, current lane, distance of obstales in three lanes, offroad} Some features have been 'pruned' from our descision tree because they did not affect the accuracy of the tree. These have been commented out for clarity.

`tick``()`
This method tries to run at regular intervals of *`TickRate`* milliseconds. Performs update logic of the cars, obstacles, and background using time since the previous tick. Stores the current tick timestamp in *`_lastTick`*

`togglePause``()`
Toggle pause state of the game

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index