guibot Documentation

Release latest

Plamen Dimitrov and Thomas Jarosch

May 07, 2024

CONTENTS

CHAPTER

ONE

GUIBOT PACKAGE

1.1 Submodules

1.1.1 guibot.calibrator module

1.1.1.1 SUMMARY

Calibration and benchmarking for all CV backends on a given matching target.

1.1.1.2 INTERFACE

guibot.calibrator.**benchmark_blacklist = [('mixed', 'normal', 'mixed', 'east', 'hmm', 'adaptive', 'adaptive'), ('mixed', 'adaptive', 'mixed', 'east', 'hmm', 'adaptive', 'adaptive'), ('mixed', 'canny', 'mixed', 'east', 'hmm', 'adaptive', 'adaptive')]**

explicit blacklist of backend combinations to skip for benchmarking

class guibot.calibrator.**Calibrator**(*needle=None*, *haystack=None*, *config=None*)

Bases: object

Provides with a group of methods to facilitate and automate the selection of algorithms and parameters that are most suitable for a given preselected image matching pair.

Use the benchmarking method to choose the best algorithm to find your image. Use the calibration method to find the best parameters if you have already chosen the algorithm. Use the search method to find the best parameters from multiple random starts from a uniform or normal probability distribution.

benchmark(*finder*, *random_starts=0*, *uniform=False*, *calibration=False*, *max_attempts=3*, ***kwargs*)

Perform benchmarking on all available algorithms of a finder for a given needle and haystack.

Parameters

- **finder** (finder.Finder) – CV backend whose backend algorithms will be benchmarked
- **random_starts** (*int*) – number of random starts to try with (0 for nonrandom)
- **uniform** (*bool*) – whether to use uniform or normal distribution
- **calibration** (*bool*) – whether to use calibration
- **max_attempts** (*int*) – maximal number of refinements to reach the parameter delta below the tolerance

Returns

list of (method, similarity, location, time) tuples sorted according to similarity

Return type
[(str, float, location.Location, float)]

Note: Methods that are supported by OpenCV and others but currently don't work are excluded from the dictionary. The dictionary can thus also be used to assess what are the available and working methods besides their success for a given *needle* and *haystack*.

---

search(*finder*, *random_starts=1*, *uniform=False*, *calibration=True*, *max_attempts=3*, ***kwargs*)

Search for the best match configuration for a given needle and haystack using calibration from random initial conditions.

Parameters

- **finder** (finder.Finder) – CV backend to use in order to determine deltas, fixed, and free parameters and ultimately tweak to minimize error
- **random_starts** (*int*) – number of random starts to try with
- **uniform** (*bool*) – whether to use uniform or normal distribution
- **calibration** (*bool*) – whether to use calibration
- **max_attempts** (*int*) – maximal number of refinements to reach the parameter delta below the tolerance

Returns
maximized similarity

Return type
float

If normal distribution is used, the mean will be the current value of the respective CV parameter and the standard variation will be determined from its delta.

calibrate(*finder*, *max_attempts=3*, ***kwargs*)

Calibrate the available match configuration for a given needle and haystack minimizing the matchign error.

Parameters

- **finder** (finder.Finder) – configuration for the CV backend to calibrate
- **max_attempts** (*int*) – maximal number of refinements to reach the parameter delta below the tolerance

Returns
maximized similarity

Return type
float

This method calibrates only parameters that are not protected from calibration, i.e. that have *fixed* attribute set to false. In order to set all parameters of a background algorithm for calibration use the finder.Finder.can_calibrate() method first. Any parameter values will only be changed if they improve the similarity, i.e. minimize the error. The deltas of the final parameters will represent the maximal flat regions in positive and/or negative direction where the same error is still obtained.

Note: All similarity parameters will be reset to 0.0 after calibration and can be set by client code afterwards.

Note: Special credits for this approach should be given to Prof. Sebastian Thrun, who explained it in his Artificial Intelligence for Robotics class.

run_default(*finder*, ***_kwargs*)

Run a match case and return error from the match as dissimilarity.

Parameters
finder (finder.Finder) – finder with match configuration to use for the run

Returns
error obtained as unity minus similarity

Return type
float

run_performance(*finder*, ***kwargs*)

Run a match case and return error from the match as dissimilarity and linear performance penalty.

Parameters

- **finder** (finder.Finder) – finder with match configuration to use for the run
- **max_exec_time** (*float*) – maximum execution time before penalizing the run by increasing the error linearly

Returns
error obtained as unity minus similarity

Return type
float

run_peak(*finder*, ***kwargs*)

Run a match case and return error from the match as failure to obtain high similarity of one match and low similarity of all others.

Parameters

- **finder** (finder.Finder) – finder with match configuration to use for the run
- **peak_location** (*(int, int)*) – (x, y) of the match whose similarity should be maximized while all the rest minimized

Returns
error obtained as unity minus similarity

Return type
float

This run function doesn't just obtain the optimum similarity for the best match in each case of needle and haystack but it minimizes the similarity for spatial competitors where spatial means other matches in the same haystack. Keep in mind that since matching is performed with zero similarity requirement, such matches might not be anything close to the needle. This run function finds use cases where the other matches could resemble the best one and we want to find configuration to better discriminate against those.

1.1.2 guibot.config module

1.1.2.1 SUMMARY

Global and local (per target or region instance) configuration.

1.1.2.2 INTERFACE

class guibot.config.**GlobalConfig**

Bases: object

Handler for default configuration present in all cases where no specific value is set.

The methods of this class are shared among all of its instances.

toggle_delay = 0.05

time interval between mouse down and up in a click

click_delay = 0.1

time interval after a click (in a double or n-click)

delay_after_drag = 0.5

timeout before drag operation

delay_before_drop = 0.5

timeout before drop operation

delay_before_keys = 0.2

timeout before key press operation

delay_between_keys = 0.1

time interval between two consecutively typed keys

rescan_speed_on_find = 0.2

time interval between two image matching attempts (used to reduce overhead on the CPU)

wait_for_animations = False

whether to wait for animations to complete and match only static (not moving) targets

smooth_mouse_drag = True

whether to move the mouse cursor to a location instantly or smoothly

preprocess_special_chars = True

whether to preprocess capital and special characters and handle them internally

save_needle_on_error = True

whether to perform an extra needle dump on matching error

image_logging_level = 40

logging level similar to the python logging module

image_logging_step_width = 3

number of digits when enumerating the image logging steps, e.g. value=3 for 001, 002, etc.

image_logging_destination = 'imglog'

relative path of the image logging steps

display_control_backend = 'pyautogui'
name of the display control backend

find_backend = 'hybrid'
name of the computer vision backend

contour_threshold_backend = 'adaptive'
name of the contour threshold backend

template_match_backend = 'ccoeff_normed'
name of the template matching backend

feature_detect_backend = 'ORB'
name of the feature detection backend

feature_extract_backend = 'ORB'
name of the feature extraction backend

feature_match_backend = 'BruteForce-Hamming'
name of the feature matching backend

text_detect_backend = 'contours'
name of the text detection backend

text_ocr_backend = 'pytesseract'
name of the optical character recognition backend

deep_learn_backend = 'pytorch'
name of the deep learning backend

hybrid_match_backend = 'template'
name of the hybrid matching backend for unconfigured one-step targets

class guibot.config.**TemporaryConfig**

Bases: object

Proxies a GlobalConfig instance extending it to add context support, such that once this context ends the changes to the wrapped config object are restored.

This is useful when we have a global config instance and need to change it only for a few operations.

```
>>> print(GlobalConfig.delay_before_drop)
0.5
>>> with TemporaryConfig() as cfg:
...     cfg.delay_before_drop = 1.3
...     print(cfg.delay_before_drop)
...     print(GlobalConfig.delay_before_drop)
...
1.3
1.3
>>> print(GlobalConfig.delay_before_drop)
0.5
```

class guibot.config.**LocalConfig**(*configure=True*, *synchronize=True*)

Bases: object

Container for the configuration of all display control and computer vision backends, responsible for making them behave according to the selected parameters as well as for providing information about them and the current parameters.

configure_backend(*backend=None*, *category='type'*, *reset=False*)

Generate configuration dictionary for a given backend.

Parameters

- **backend** (*str or None*) – name of a preselected backend, see *algorithms[category]*
- **category** (*str*) – category for the backend, see *algorithms.keys()*
- **reset** (*bool*) – whether to (re)set all parent configurations as well

Raises

UnsupportedBackendError if *backend* is not among the supported backends for the category (and is not *None*) or the category is not found

configure(*reset=True*, ***kwargs*)

Generate configuration dictionary for all backends.

Parameters

reset (*bool*) – whether to (re)set all parent configurations as well

If multiple categories are available and just some of them are configured, the rest will be reset to defaults. To configure specific category without changing others, use *configure()*.

synchronize_backend(*backend=None*, *category='type'*, *reset=False*)

Synchronize a category backend with the equalizer configuration.

Parameters

- **backend** (*str or None*) – name of a preselected backend, see *algorithms[category]*
- **category** (*str*) – category for the backend, see *algorithms.keys()*
- **reset** (*bool*) – whether to (re)sync all parent backends as well

Raises

UnsupportedBackendError if the category is not found

Raises

UninitializedBackendError if there is no backend object that is configured with and with the required name

synchronize(**args*, *reset=True*, ***kwargs*)

Synchronize all backends with the current configuration dictionary.

Parameters

reset (*bool*) – whether to (re)sync all parent backends as well

1.1.3 guibot.controller module

1.1.3.1 SUMMARY

Display controllers (DC backends) to perform user operations.

1.1.3.2 INTERFACE

class guibot.controller.**Controller**(*configure=True*, *synchronize=True*)

Bases: *LocalConfig*

Screen control backend, responsible for performing desktop operations like mouse clicking, key pressing, text typing, etc.

property width

Getter for readonly attribute.

Returns
width of the connected screen

Return type
int

property height

Getter for readonly attribute.

Returns
height of the connected screen

Return type
int

property keymap

Getter for readonly attribute.

Returns
map of keys to be used for the connected screen

Return type
`inputmap.Key`

property mousemap

Getter for readonly attribute.

Returns
map of mouse buttons to be used for the connected screen

Return type
`inputmap.MouseButton`

property modmap

Getter for readonly attribute.

Returns
map of modifier keys to be used for the connected screen

Return type
`inputmap.KeyModifier`

property mouse_location

Getter for readonly attribute.

Returns
location of the mouse pointer

Return type
`location.Location`

configure_backend(*backend=None*, *category='control'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='control'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

Select a backend for the instance, synchronizing configuration like screen size, key map, mouse pointer handling, etc. The object that carries this configuration is called screen.

capture_screen(**args*)

Get the current screen as image.

Parameters
args ([int] or region.Region or None) – region's (x, y, width, height) or a region object or nothing to obtain an image of the full screen

Returns
image of the current screen

Return type
target.Image

Raises
NotImplementedError if the base class method is called

mouse_move(*location*, *smooth=True*)

Move the mouse to a desired location.

Parameters

- **location** (location.Location) – location on the screen to move to
- **smooth** (*bool*) – whether to sue smooth transition or just teleport the mouse

Raises
NotImplementedError if the base class method is called

mouse_click(*button=None*, *count=1*, *modifiers=None*)

Click the selected mouse button N times at the current mouse location.

Parameters

- **button** (*int or None*) – mouse button, e.g. self.mouse_map.LEFT_BUTTON
- **count** (*int*) – number of times to click
- **modifiers** (*[str]*) – special keys to hold during clicking (see inputmap.KeyModifier for extensive list)

Raises
NotImplementedError if the base class method is called

mouse_down(*button*)

Hold down a mouse button.

Parameters
button (*int*) – button index depending on backend (see inputmap.MouseButton for extensive list)

Raises
NotImplementedError if the base class method is called

mouse_up(*button*)

Release a mouse button.

Parameters
button (*int*) – button index depending on backend (see inputmap.MouseButton for extensive list)

Raises
NotImplementedError if the base class method is called

mouse_scroll(*clicks=10*, *horizontal=False*)

Scroll the mouse for a number of clicks.

Parameters

- **clicks** (*int*) – number of clicks to scroll up (positive) or down (negative)
- **horizontal** (*bool*) – whether to perform a horizontal scroll instead (only available on some platforms)

Raises
NotImplementedError if the base class method is called

keys_toggle(*keys*, *up_down*)

Hold down or release together all provided keys.

Parameters

- **keys** (*[str] or str (no special keys in the second case)*) – characters or special keys depending on the backend (see inputmap.Key for extensive list)
- **up_down** (*bool*) – hold down if true else release

Raises
NotImplementedError if the base class method is called

keys_press(*keys*)

Press (hold down and release) together all provided keys.

Parameters
keys (*[str] or str (no special keys in the second case)*) – characters or special keys depending on the backend (see inputmap.Key for extensive list)

keys_type(*text*, *modifiers=None*)

Type (press consecutively) all provided keys.

Parameters

- **text** (*[str] or str (second case is preferred and first redundant)*) – characters only (no special keys allowed)
- **modifiers** (*[str]*) – special keys to hold during typing (see inputmap.KeyModifier for extensive list)

Raises
NotImplementedError if the base class method is called

class guibot.controller.**AutoPyController**(*configure=True*, *synchronize=True*)

Bases: *Controller*

Screen control backend implemented through AutoPy which is a small python library portable to Windows and Linux operating systems.

property mouse_location

Custom implementation of the base method.

See base method for details.

configure_backend(*backend=None*, *category='autopy'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='autopy'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

capture_screen(**args*)

Custom implementation of the base method.

See base method for details.

mouse_move(*location*, *smooth=True*)

Custom implementation of the base method.

See base method for details.

mouse_click(*button=None*, *count=1*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

mouse_down(*button*)

Custom implementation of the base method.

See base method for details.

mouse_up(*button*)

Custom implementation of the base method.

See base method for details.

keys_toggle(*keys*, *up_down*)

Custom implementation of the base method.

See base method for details.

keys_type(*text*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

class guibot.controller.**XDoToolController**(*configure=True*, *synchronize=True*)

Bases: *Controller*

Screen control backend implemented through the xdotool client and thus portable to Linux operating systems.

property mouse_location

Custom implementation of the base method.

See base method for details.

configure_backend(*backend=None*, *category='xdotool'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='xdotool'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

capture_screen(**args*)

Custom implementation of the base method.

See base method for details.

mouse_move(*location*, *smooth=True*)

Custom implementation of the base method.

See base method for details.

mouse_click(*button=None*, *count=1*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

mouse_down(*button*)

Custom implementation of the base method.

See base method for details.

mouse_up(*button*)

Custom implementation of the base method.

See base method for details.

keys_toggle(*keys*, *up_down*)

Custom implementation of the base method.

See base method for details.

keys_type(*text*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

class guibot.controller.**VNCDoToolController**(*configure=True*, *synchronize=True*)

Bases: *Controller*

Screen control backend implemented through the VNCDoTool client and thus portable to any guest OS that is accessible through a VNC/RFB protocol.

configure_backend(*backend=None*, *category='vncdotool'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='vncdotool'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

capture_screen(**args*)

Custom implementation of the base method.

See base method for details.

mouse_move(*location*, *smooth=True*)

Custom implementation of the base method.

See base method for details.

mouse_click(*button=None*, *count=1*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

mouse_down(*button*)

Custom implementation of the base method.

See base method for details.

mouse_up(*button*)

Custom implementation of the base method.

See base method for details.

keys_toggle(*keys*, *up_down*)

Custom implementation of the base method.

See base method for details.

keys_type(*text*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

class guibot.controller.**PyAutoGUIController**(*configure=True*, *synchronize=True*)

Bases: *Controller*

Screen control backend implemented through PyAutoGUI which is a python library portable to MacOS, Windows, and Linux operating systems.

property mouse_location

Custom implementation of the base method.

See base method for details.

configure_backend(*backend=None*, *category='pyautogui'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='pyautogui'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

capture_screen(**args*)

Custom implementation of the base method.

See base method for details.

mouse_move(*location*, *smooth=True*)

Custom implementation of the base method.

See base method for details.

mouse_click(*button=None*, *count=1*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

mouse_down(*button*)

Custom implementation of the base method.

See base method for details.

mouse_up(*button*)

Custom implementation of the base method.

See base method for details.

mouse_scroll(*clicks=10*, *horizontal=False*)

Custom implementation of the base method.

See base method for details.

keys_toggle(*keys*, *up_down*)

Custom implementation of the base method.

See base method for details.

keys_type(*text*, *modifiers=None*)

Custom implementation of the base method.

See base method for details.

1.1.4 guibot.desktopcontrol module

1.1.4.1 SUMMARY

Old module for display controllers (DC backends) - to be deprecated.

1.1.4.2 INTERFACE

1.1.5 guibot.errors module

1.1.5.1 SUMMARY

Exceptions used by all guibot interfaces and modules.

1.1.5.2 INTERFACE

exception guibot.errors.**GuiBotError**

Bases: Exception

GuiBot exception base class

exception guibot.errors.**FileNotFoundError**

Bases: *GuiBotError*

Exception raised when a picture file cannot be found on disc

exception guibot.errors.**IncompatibleTargetError**

Bases: *GuiBotError*

Exception raised when a matched target is of type that cannot be handled by the finder

exception guibot.errors.**IncompatibleTargetFileError**

Bases: *GuiBotError*

Exception raised when a matched target is restored from a file of unsupported type

exception guibot.errors.**FindError**(*failed_target=None*)

Bases: *GuiBotError*

Exception raised when an Image cannot be found on the screen

exception guibot.errors.**NotFindError**(*failed_target=None*)

Bases: *GuiBotError*

Exception raised when an Image can be found on the screen but should not be

exception guibot.errors.**UnsupportedBackendError**

Bases: *GuiBotError*

Exception raised when a non-existent method is used for finding a target

exception guibot.errors.**MissingHotmapError**

Bases: *GuiBotError*

Exception raised when an attempt to access a non-existent hotmap in the image logger is made

exception guibot.errors.**UninitializedBackendError**

Bases: *GuiBotError*

Exception raised when a region is created within an empty screen (a disconnected display control backend)

1.1.6 guibot.fileresolver module

1.1.6.1 SUMMARY

Cached and reused paths for target files to search in and load target data from.

1.1.6.2 INTERFACE

class guibot.fileresolver.**FileResolver**

Bases: object

Handler for currently used target paths or sources of targets with a desired name.

The methods of this class are shared among all of its instances.

add_path(*directory*)

Add a path to the list of currently accessible paths if it wasn't already added.

Parameters
directory (*str*) – path to add

remove_path(*directory*)

Remove a path from the list of currently accessible paths.

Parameters
directory (*str*) – path to add

Returns
whether the removal succeeded

Return type
bool

clear()

Clear all currently accessible paths.

search(*filename*, *restriction=''*, *silent=False*)

Search for a filename in the currently accessible paths.

Parameters

- **filename** (*str*) – filename of the target to search for
- **restriction** (*str*) – simple string to restrict the number of paths
- **silent** (*bool*) – whether to return None instead of error out

Returns
the full name of the found target file or None if silent and no file was found

Return type
str or None

Raises
FileNotFoundError if no such file was found and not silent

class guibot.fileresolver.**CustomFileResolver**(**paths*)

Bases: object

Class to be used to search for files inside certain paths.

Inside the context of an instance of this class, the paths in the shared list in *FileResolver* will be temporarily replaced by the paths passed to the constructor of this class. This means that any call to *FileResolver.search()* will take only these paths into account.

1.1.7 guibot.finder module

1.1.7.1 SUMMARY

Computer vision finders (CV backends) to perform find targets on screen.

1.1.7.2 INTERFACE

class guibot.finder.**CVParameter**(*value*, *min_val=None*, *max_val=None*, *delta=10.0*, *tolerance=1.0*, *fixed=True*, *enumerated=False*)

Bases: object

A class for a single parameter used for CV backend configuration.

static from_string(*raw*)

Parse a CV parameter from string.

Parameters
raw (*str*) – string representation for the parameter

Returns
parameter parsed from the representation

Return type
CVParameter

Raises
ValueError if unsupported type is encountered

random_value(*mu=None*, *sigma=None*)

Return a random value of the CV parameter given its range and type.

Parameters

- **mu** (*bool or int or float or str or None*) – mean for a normal distribution, uniform distribution if None
- **sigma** (*bool or int or float or str or None*) – standard deviation for a normal distribution, quarter range if None (maximal range is equivalent to maximal data type values)

Returns
a random value comforming to the CV parameter range and type

Return type
bool or int or float or str or None

Note: Only uniform distribution is used for boolean values.

class guibot.finder.**Finder**(*configure=True*, *synchronize=True*)

Bases: *LocalConfig*

Base for all image matching functionality and backends.

The image finding methods include finding one or all matches above the similarity defined in the configuration of each backend.

There are many parameters that could contribute for a good match. They can all be manually adjusted or automatically calibrated.

static from_match_file(*filename*)

Read the configuration from a match file with the given filename.

Parameters
filename (*str*) – match filename for the configuration

Returns
target finder with the parsed (and generated) settings

Return type
finder.Finder

Raises
IOError if the respective match file couldn't be read

The influence of the read configuration is that of an overwrite, i.e. all parameters will be generated (if not already present) and then the ones read from the configuration file will be overwritten.

static to_match_file(*finder*, *filename*)

Write the configuration to a match file with the given filename.

Parameters

- **finder** (finder.Finder) – match configuration to save
- **filename** (*str*) – match filename for the configuration

configure_backend(*backend=None*, *category='find'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='find'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

can_calibrate(*category*, *mark*)

Fix the parameters for a given category backend algorithm, i.e. disallow the calibrator to change them.

Parameters

- **mark** (*bool*) – whether to mark for calibration
- **category** (*str*) – backend category whose parameters are marked

Raises
UnsupportedBackendError if *category* is not among the supported backend categories

copy()

Deep copy the current finder and its configuration.

Returns
a copy of the current finder with identical configuration

Return type
Finder

find(*needle*, *haystack*)

Find all needle targets in a haystack image.

Parameters

- **needle** (target.Target or [target.Target]) – image, text, pattern, or a list or chain of such to look for
- **haystack** (target.Image) – image to look in

Returns
all found matches (one in most use cases)

Return type
[match.Match]

Raises
NotImplementedError if the base class method is called

log(*lvl*)

Log images with an arbitrary logging level.

Parameters
lvl (*int*) – logging level for the message

class guibot.finder.**AutoPyFinder**(*configure=True*, *synchronize=True*)

Bases: *Finder*

Simple matching backend provided by AutoPy.

configure_backend(*backend=None*, *category='autopy'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Image) – target iamge to search for

See base method for details.

> **Warning:** AutoPy has a bug when finding multiple matches so it will currently only return a single match.

class guibot.finder.**ContourFinder**(*configure=True*, *synchronize=True*)

Bases: *Finder*

Contour matching backend provided by OpenCV.

Essentially, we will find all countours in a binary image, preprocessed with Gaussian blur and adaptive threshold and return the ones with area (size) similar to the searched image.

configure_backend(*backend=None*, *category='contour'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

configure(*threshold_filter=None*, *reset=True*, ***kwargs*)

Custom implementation of the base method.

Parameters
threshold_filter (*str or None*) – name of a preselected backend

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Image) – target iamge to search for

See base method for details.

First extract all contours from a binary (boolean, threshold) version of the needle and haystack and then match the needle contours with one or more sets of contours in the haystack image. The number of needle matches depends on the set similarity and can be improved by requiring minimal area for the contours to be considered.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**TemplateFinder**(*configure=True*, *synchronize=True*)

Bases: *Finder*

Template matching backend provided by OpenCV.

configure_backend(*backend=None*, *category='template'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Image) – target iamge to search for

Raises
UnsupportedBackendError if the choice of template matches is not among the supported ones

See base method for details.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**FeatureFinder**(*configure=True*, *synchronize=True*)

Bases: *Finder*

Feature matching backend provided by OpenCV.

Note: SURF and SIFT are proprietary algorithms and are not available by default in newer OpenCV versions (>3.0).

---

configure_backend(*backend=None*, *category='feature'*, *reset=False*)

Custom implementation of the base method.

Some relevant parameters are:

- **detect filter - works for certain detectors and**
  determines how many initial features are detected in an image (e.g. hessian threshold for SURF detector)

- **match filter - determines what part of all matches**
  returned by feature matcher remain good matches
- **project filter - determines what part of the good**
  matches are considered inliers
- ratio test - boolean for whether to perform a ratio test
- symmetry test - boolean for whether to perform a symmetry test

See base method for details.

configure(*feature_detect=None*, *feature_extract=None*, *feature_match=None*, *reset=True*, ***kwargs*)

Custom implementation of the base method.

Parameters

- **feature_detect** (*str or None*) – name of a preselected backend
- **feature_extract** (*str or None*) – name of a preselected backend
- **feature_match** (*str or None*) – name of a preselected backend

synchronize_backend(*backend=None*, *category='feature'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize(*feature_detect=None*, *feature_extract=None*, *feature_match=None*, *reset=True*)

Custom implementation of the base method.

Parameters

- **feature_detect** (*str or None*) – name of a preselected backend
- **feature_extract** (*str or None*) – name of a preselected backend
- **feature_match** (*str or None*) – name of a preselected backend

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Image) – target iamge to search for

See base method for details.

> **Warning:** Finding multiple matches is currently not supported and this will currently only return a single match.

Available methods are: a combination of feature detector, extractor, and matcher.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**CascadeFinder**(*classifier_datapath='.'*, *configure=True*, *synchronize=True*)

Bases: *Finder*

Cascade matching backend provided by OpenCV.

This matcher uses Haar cascade for object detection. It is the most advanced method for object detection excluding convolutional neural networks. However, it requires the generation of a Haar cascade (if such is not already provided) of the needle to be found.

TODO: Currently no similarity requirement can be applied due to the cascade classifier API.

configure_backend(*backend=None*, *category='cascade'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Pattern) – target pattern (cascade) to search for

See base method for details.

class guibot.finder.**TextFinder**(*configure=True*, *synchronize=True*)

Bases: *ContourFinder*

Text matching backend provided by OpenCV.

This matcher will find a text (string) needle in the haystack, eventually relying on Tesseract or simpler kNN-based OCR, using extremal regions or contours before recognition, and returning a match if the string is among the recognized strings using string metric similar to Hamming distance.

Extremal Region Filter algorithm described in: Neumann L., Matas J.: Real-Time Scene Text Localization and Recognition, CVPR 2012

configure_backend(*backend=None*, *category='text'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

configure(*text_detector=None*, *text_recognizer=None*, *threshold_filter=None*, *threshold_filter2=None*, *threshold_filter3=None*, *reset=True*, ***kwargs*)

Custom implementation of the base method.

Parameters

- **text_detector** (*str or None*) – name of a preselected backend
- **text_recognizer** (*str or None*) – name of a preselected backend
- **threshold_filter** (*str or None*) – threshold filter for the text detection stage
- **threshold_filter2** (*str or None*) – additional threshold filter for the OCR stage
- **threshold_filter3** (*str or None*) – additional threshold filter for distance transformation

synchronize_backend(*backend=None*, *category='text'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize(*text_detector=None*, *text_recognizer=None*, *threshold_filter=None*, *threshold_filter2=None*, *threshold_filter3=None*, *reset=True*)

Custom implementation of the base method.

Parameters

- **text_detector** (*str or None*) – name of a preselected backend
- **text_recognizer** (*str or None*) – name of a preselected backend
- **threshold_filter** (*str or None*) – threshold filter for the text detection stage
- **threshold_filter2** (*str or None*) – additional threshold filter for the OCR stage
- **threshold_filter3** (*str or None*) – additional threshold filter for distance transformation

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Text) – target text to search for

See base method for details.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**TemplateFeatureFinder**(*configure=True*, *synchronize=True*)

Bases: *TemplateFinder*, *FeatureFinder*

Hybrid matcher using both OpenCV's template and feature matching.

Feature matching is robust at small regions not too abundant of features where template matching is too picky. Template matching is good at large feature abundant regions and can be used as a heuristic for the feature matching. The current matcher will perform template matching first and then feature matching on the survived template matches to select among them one more time.

A separate (usually lower) front similarity is used for the first stage template matching in order to remove a lot of noise that would otherwise be distracting for the second stage feature matching.

configure_backend(*backend=None*, *category='tempfeat'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

configure(*template_match=None*, *feature_detect=None*, *feature_extract=None*, *feature_match=None*, *reset=True*, ***kwargs*)

Custom implementation of the base methods.

See base methods for details.

synchronize(*feature_detect=None*, *feature_extract=None*, *feature_match=None*, *reset=True*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

See base method for details.

Use template matching to deal with feature dense regions and guide a final feature matching stage.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**DeepFinder**(*classifier_datapath='.'*, *configure=True*, *synchronize=True*)

Bases: *Finder*

Deep learning matching backend provided by PyTorch.

The current implementation contains a basic convolutional neural network which can be trained to produce needle locations from a haystack image.

configure_backend(*backend=None*, *category='deep'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='deep'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

Parameters
needle (Pattern) – target pattern (cascade) to search for

See base method for details.

log(*lvl*)

Custom implementation of the base method.

See base method for details.

class guibot.finder.**HybridFinder**(*configure=True*, *synchronize=True*)

Bases: *Finder*

Match a target through a sequence of differently configured attempts.

This matcher can work with any other matcher in the background and with unique or repeating matchers for each step. If a step fails, the matcher tries the next available along the fallback chain or fails if the end of the chain is reached.

configure_backend(*backend=None*, *category='hybrid'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

synchronize_backend(*backend=None*, *category='hybrid'*, *reset=False*)

Custom implementation of the base method.

See base method for details.

find(*needle*, *haystack*)

Custom implementation of the base method.

See base method for details.

1.1.8 guibot.guibot module

1.1.8.1 SUMMARY

Main guibot interface for GUI automation.

This frontend is recommended for use in most normal cases.

1.1.8.2 INTERFACE

class guibot.guibot.**GuiBot**(*dc=None*, *cv=None*)

Bases: *Region*

The main guibot object is the root (first and screen wide) region with some convenience functions added.

See also:

Real API is inherited from `region.Region`.

add_path(*directory*)

Add a path to the list of currently accessible paths if it wasn't already added.

Parameters
directory (*str*) – path to add

remove_path(*directory*)

Remove a path from the list of currently accessible paths.

Parameters
directory (*str*) – path to add

1.1.9 guibot.guibot_proxy module

1.1.9.1 SUMMARY

Remote guibot interface for proxy operations using remote visual objects.

Frontend with serialization compatible API allowing the use of PyRO modified `guibot.GuiBot` object (creating and running the same object remotely and manipulating it locally). All the methods delegate their calls to this object with some additional postprocessing to make the execution remote so for information about the API please refer to it and `region.Region`.

1.1.9.2 INTERFACE

guibot.guibot_proxy.**serialize_custom_error**(*class_obj*)

Serialization method for the `errors.UnsupportedBackendError` which was chosen just as a sample.

Parameters
class_obj (*classobj*) – class object for the serialized error class

Returns
serialization dictionary with the class name, arguments, and attributes

Return type
{str, str or getset_descriptor or dictproxy}

guibot.guibot_proxy.**register_exception_serialization**()

We put here any exceptions that are too complicated for the default serialization and define their serialization methods.

Note: This would not be needed if we were using the Pickle serializer but its security problems at the moment made us prefer the serpent serializer paying for it with some extra setup steps and functions below.

---

class guibot.guibot_proxy.**GuiBotProxy**(*dc=None*, *cv=None*)

Bases: *GuiBot*

The proxy guibot object is just a wrapper around the actual guibot object that takes care of returning easily serializable PyRO proxy objects instead of the real ones or their serialized copies.

It allows you to move the mouse, type text and do any other GuiBot action from code which is executed on another machine somewhere on the network.

nearby(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

above(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

below(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

left(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

right(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

find(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

find_all(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

sample(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

exists(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

wait(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

wait_vanish(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

idle(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

hover(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

click(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

right_click(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

middle_click(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

double_click(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

multi_click(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

click_expect(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

click_vanish(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

click_at_index(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

mouse_down(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

mouse_up(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

mouse_scroll(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

drag_drop(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

drag_from(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

drop_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

press_keys(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

press_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

press_expect(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

press_vanish(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

type_text(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

type_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

click_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

fill_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

select_at(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

1.1.10 guibot.guibot_simple module

1.1.10.1 SUMMARY

Simple guibot interface for short scripts, examples, and basic GUI automation.

Frontend with simple procedural API allowing the use of a module instead of the `guibot.GuiBot` object (creating and running this same object internally). All the methods delegate their calls to this object so for information about the API please refer to it and `region.Region`.

1.1.10.2 INTERFACE

guibot.guibot_simple.**buttons**

alias of `Buttons`

guibot.guibot_simple.**initialize**()

Initialize the simple API.

guibot.guibot_simple.**check_initialized**()

Make sure the simple API is initialized.

guibot.guibot_simple.**add_path**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**remove_path**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**find**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**find_all**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**sample**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**exists**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**wait**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**wait_vanish**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**get_mouse_location**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**idle**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**hover**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**click**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**right_click**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**middle_click**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**double_click**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**multi_click**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**click_expect**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**click_vanish**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**click_at_index**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**mouse_down**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**mouse_up**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**mouse_scroll**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**drag_drop**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**drag_from**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**drop_at**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**press_keys**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**press_at**(**args*, ***kwargs*)
See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**press_expect**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**press_vanish**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**type_text**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**type_at**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**click_at**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**fill_at**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

guibot.guibot_simple.**select_at**(**args*, ***kwargs*)

See *guibot.guibot.GuiBot* and its inherited *guibot.region.Region* for details.

1.1.11 guibot.imagelogger module

1.1.11.1 SUMMARY

Image logging for enhanced debugging and verbosity of guibot's operation.

1.1.11.2 INTERFACE

class guibot.imagelogger.**ImageLogger**

Bases: `object`

Logger for the image matching process with the help of images.

It always contains the current match case: the needle and haystack images/targets being matched and the hotmap (an image with additional drawn information on it), the matched similarity and the matched coordinates.

Generally, each finder class takes care of its own image logging, performing drawing or similar operations on the spot and deciding which hotmaps (also their names and order) to dump.

step = 1

number of the current step

accumulate_logging = False

switch to stop logging and later on log all accumulated dumps at once

logging_level = 40

level for the image logging

logging_destination = 'imglog'

destination for the image logging in order to dump images (the executing code decides when to clean this directory)

step_width = 3

number of digits for the counter of logged steps

property printable_step
Getter for readonly attribute.

Returns
step number prepended with zeroes to obtain a fixed length enumeration

Return type
str

debug()
Log images with a DEBUG logging level.

info()
Log images with an INFO logging level.

warning()
Log images with a WARNING logging level.

error()
Log images with an ERROR logging level.

critical()
Log images with a CRITICAL logging level.

dump_matched_images()
Write file with the current needle and haystack.

The current needle and haystack (matched images) are stored as *needle* and *haystack* attributes.

dump_hotmap(*name*, *hotmap*)
Write a file the given hotmap.

Parameters

- **name** (*str*) – filename to use for the image
- **hotmap** (PIL.Image or numpy.ndarray) – image (with matching results) to write

clear()
Clear all accumulated logging including hotmaps, similarities, and locations.

1.1.12 guibot.inputmap module

1.1.12.1 SUMMARY

Key mappings, modifiers, and mouse buttons.

1.1.12.2 INTERFACE

class guibot.inputmap.**Key**
Bases: object

Helper to contain all key mappings for a custom display control backend.

to_string(*key*)

Provide with a text representation of a desired key according to the custom BC backend.

Parameters
key (*str*) – selected key name according to the custom backend

Returns
text representation of the selected key

Return type
str

Raises
ValueError if *key* is not found in the current key map

class guibot.inputmap.**AutoPyKey**

Bases: *Key*

Helper to contain all key mappings for the AutoPy DC backend.

class guibot.inputmap.**XDoToolKey**

Bases: *Key*

Helper to contain all key mappings for the xdotool DC backend.

class guibot.inputmap.**VNCDoToolKey**

Bases: *Key*

Helper to contain all key mappings for the VNCDoTool DC backend.

class guibot.inputmap.**PyAutoGUIKey**

Bases: *Key*

Helper to contain all key mappings for the PyAutoGUI DC backend.

class guibot.inputmap.**KeyModifier**

Bases: object

Helper to contain all modifier key mappings for a custom display control backend.

to_string(*key*)

Provide with a text representation of a desired modifier key according to the custom BC backend.

Parameters
key (*str*) – selected modifier name according to the current backend

Returns
text representation of the selected modifier

Return type
str

Raises
ValueError if *key* is not found in the current modifier map

class guibot.inputmap.**AutoPyKeyModifier**

Bases: *KeyModifier*

Helper to contain all modifier key mappings for the AutoPy DC backend.

class guibot.inputmap.**XDoToolKeyModifier**

Bases: *KeyModifier*

Helper to contain all modifier key mappings for the xdotool DC backend.

class guibot.inputmap.**VNCDoToolKeyModifier**

Bases: *KeyModifier*

Helper to contain all modifier key mappings for the VNCDoTool DC backend.

class guibot.inputmap.**PyAutoGUIKeyModifier**

Bases: *KeyModifier*

Helper to contain all modifier key mappings for the PyAutoGUI DC backend.

class guibot.inputmap.**MouseButton**

Bases: `object`

Helper to contain all mouse button mappings for a custom display control backend.

to_string(*key*)

Provide with a text representation of a desired mouse button according to the custom BC backend.

Parameters
key (*str*) – selected mouse button according to the current backend

Returns
text representation of the selected mouse button

Return type
str

Raises
`ValueError` if *key* is not found in the current mouse map

class guibot.inputmap.**AutoPyMouseButton**

Bases: *MouseButton*

Helper to contain all mouse button mappings for the AutoPy DC backend.

class guibot.inputmap.**XDoToolMouseButton**

Bases: *MouseButton*

Helper to contain all mouse button mappings for the xdotool DC backend.

class guibot.inputmap.**VNCDoToolMouseButton**

Bases: *MouseButton*

Helper to contain all mouse button mappings for the VNCDoTool DC backend.

class guibot.inputmap.**PyAutoGUIMouseButton**

Bases: *MouseButton*

Helper to contain all mouse button mappings for the PyAutoGUI DC backend.

1.1.13 guibot.location module

1.1.13.1 SUMMARY

Simple class to hold screen location data.

..note:: Unless this class becomes more useful for the extra OOP abstraction it might get deprecated in favor of a simple (x, y) tuple.

1.1.13.2 INTERFACE

class guibot.location.**Location**(*xpos=0*, *ypos=0*)

Bases: object

Simple location on a 2D surface, region, or screen.

property x

Getter for readonly attribute.

Returns
x coordinate of the location

Return type
int

property y

Getter for readonly attribute.

Returns
y coordinate of the location

Return type
int

1.1.14 guibot.match module

1.1.14.1 SUMMARY

Class and functionality related to target matches on screen.

1.1.14.2 INTERFACE

class guibot.match.**Match**(*xpos*, *ypos*, *width*, *height*, *dx=0*, *dy=0*, *similarity=0.0*, *dc=None*, *cv=None*)

Bases: *Region*

Wrapper around image which adds data necessary for manipulation of matches on a screen.

property x

Getter for readonly attribute.

Returns
x coordinate of the upleft vertex of the region

Return type
int

property y

Getter for readonly attribute.

Returns
y coordinate of the upleft vertex of the region

Return type
int

property dx

Getter for readonly attribute.

Returns
x offset from the center of the match region

Return type
int

property dy

Getter for readonly attribute.

Returns
y offset from the center of the match region

Return type
int

property similarity

Getter for readonly attribute.

Returns
similarity the match was obtained with

Return type
float

property target

Getter for readonly attribute.

Returns
target location to click on if clicking on the match

Return type
location.Location

calc_click_point(*xpos*, *ypos*, *width*, *height*, *offset*)

Calculate target location to click on if clicking on the match.

Parameters

- **xpos** (*int*) – x coordinate of upleft vertex of the match region
- **ypos** (*int*) – y coordinate of upleft vertex of the match region
- **width** (*int*) – width of the match region
- **height** (*int*) – height of the match region
- **offset** (location.Location) – offset from the match region center for the final target

Returns
target location to click on if clicking on the match

Return type
location.Location

1.1.15 guibot.path module

1.1.15.1 SUMMARY

Old module for path resolution - to be deprecated.

1.1.15.2 INTERFACE

guibot.path.**Path**
alias of *FileResolver*

1.1.16 guibot.region module

1.1.16.1 SUMMARY

Secondary (and more advanced) interface for generic screen regions.

The main guibot interface is just a specialized region where we could match and work with subregions. Any region instance can also be a complete screen, hence the increased generality of using this as an interface and calling it directly.

1.1.16.2 INTERFACE

class guibot.region.**Region**(*xpos=0*, *ypos=0*, *width=0*, *height=0*, *dc=None*, *cv=None*)

Bases: object

Region of the screen supporting vertex and nearby region selection, validation of expected images, and mouse and keyboard control.

property x

Getter for readonly attribute.

Returns
x coordinate of the upleft vertex of the region

Return type
int

property y

Getter for readonly attribute.

Returns
y coordinate of the upleft vertex of the region

Return type
int

property width

Getter for readonly attribute.

Returns
width of the region (xpos+width for downright vertex x)

Return type
int

property height

Getter for readonly attribute.

Returns
height of the region (ypos+height for downright vertex y)

Return type
int

property center

Getter for readonly attribute.

Returns
center of the region

Return type
location.Location

property top_left

Getter for readonly attribute.

Returns
upleft vertex of the region

Return type
location.Location

property top_right

Getter for readonly attribute.

Returns
upright vertex of the region

Return type
location.Location

property bottom_left

Getter for readonly attribute.

Returns
downleft vertex of the region

Return type
location.Location

property bottom_right

Getter for readonly attribute.

Returns
downright vertex of the region

Return type
location.Location

property is_empty

Getter for readonly attribute.

Returns
whether the region is empty, i.e. has zero size

Return type
bool

property last_match

Getter for readonly attribute.

Returns
last match obtained from finding a target within the region

Return type
match.Match

property mouse_location

Main region methods

nearby(*rrange=50*)

Obtain a region containing the previous one but enlarged by a number of pixels on each side.

Parameters
rrange (*int*) – number of pixels to add

Returns
new region enlarged by *rrange* on all sides

Return type
Region

above(*rrange=0*)

Obtain a region containing the previous one but enlarged by a number of pixels on the upper side.

Parameters
rrange (*int*) – number of pixels to add

Returns
new region enlarged by *rrange* on upper side

Return type
Region

below(*rrange=0*)

Obtain a region containing the previous one but enlarged by a number of pixels on the lower side.

Parameters
rrange (*int*) – number of pixels to add

Returns
new region enlarged by *rrange* on lower side

Return type
Region

left(*rrange=0*)

Obtain a region containing the previous one but enlarged by a number of pixels on the left side.

Parameters
rrange (*int*) – number of pixels to add

Returns
new region enlarged by *rrange* on left side

Return type
Region

right(*rrange=0*)

Obtain a region containing the previous one but enlarged by a number of pixels on the right side.

Parameters
rrange (*int*) – number of pixels to add

Returns
new region enlarged by *rrange* on right side

Return type
Region

find(*target*, *timeout=10*)

Find a target on the screen.

Parameters

- **target** (str or target.Target) – target to look for
- **timeout** (*int*) – timeout before giving up

Returns
match obtained from finding the target within the region

Return type
match.Match

This method is the main entrance to all our target finding capabilities and is the milestone for all target expect methods.

find_all(*target*, *timeout=10*, *allow_zero=False*)

Find multiples of a target on the screen.

Parameters

- **target** (str or target.Target) – target to look for
- **timeout** (*int*) – timeout before giving up
- **allow_zero** (*bool*) – whether to allow zero matches or raise error

Returns
matches obtained from finding the target within the region

Return type
[match.Match]

Raises
errors.FindError if no matches are found and zero matches are not allowed

This method is similar the one above but allows for more than one match.

sample(*target*)

Sample the similarity between a target and the screen, i.e. an empirical probability that the target is on the screen.

Parameters
target (str or target.Target) – target to look for

Returns
similarity with best match on the screen

Return type
float

Note: Not all matchers support a 'similarity' value. The ones that don't will return zero similarity (similarly to the target logging case).

exists(*target*, *timeout=0*)

Check if a target exists on the screen using the matching success as a threshold for the existence.

Parameters

- **target** (str or `target.Target`) – target to look for
- **timeout** (*int*) – timeout before giving up

Returns
match obtained from finding the target within the region or nothing if no match is found

Return type
`match.Match` or None

wait(*target*, *timeout=30*)

Wait for a target to appear (be matched) with a given timeout as failing tolerance.

Parameters

- **target** (str or `target.Target`) – target to look for
- **timeout** (*int*) – timeout before giving up

Returns
match obtained from finding the target within the region

Return type
`match.Match`

Raises
`errors.FindError` if no match is found

wait_vanish(*target*, *timeout=30*)

Wait for a target to disappear (be unmatched, i.e. matched without success) with a given timeout as failing tolerance.

Parameters

- **target** (str or `target.Target`) – target to look for
- **timeout** (*int*) – timeout before giving up

Returns
self

Return type
Region

Raises
`errors.NotFindError` if match is still found

idle(*timeout*)

Wait for a number of seconds and continue the nested call chain.

Parameters
timeout (*int*) – timeout to wait for

Returns
self

Return type
Region

This method can be used as both a way to compactly wait for some time while not breaking the call chain. e.g.:

```
aregion.hover('abox').idle(1).click('aboxwithinthebox')
```

and as a way to conveniently perform timeout in between actions.

hover(*target_or_location*)

Hover the mouse over a target or location.

Parameters
target_or_location (match.Match or location.Location or str or target.Target) – target or location to hover to

Returns
match from finding the target or nothing if hovering over a known location

Return type
match.Match or None

click(*target_or_location*, *modifiers=None*)

Click on a target or location using the left mouse button and optionally holding special keys.

Parameters

- **target_or_location** (match.Match or location.Location or str or target.Target) – target or location to click on
- **modifiers** (*[str]*) – special keys to hold during clicking (see inputmap.KeyModifier for extensive list)

Returns
match from finding the target or nothing if clicking on a known location

Return type
match.Match or None

The special keys refer to a list of key modifiers, e.g.:

```
self.click('my_target', [KeyModifier.MOD_CTRL, 'x']).
```

right_click(*target_or_location*, *modifiers=None*)

Click on a target or location using the right mouse button and optionally holding special keys.

Arguments and return values are analogical to *Region.click()*.

middle_click(*target_or_location*, *modifiers=None*)

Click on a target or location using the middle mouse button and optionally holding special keys.

Arguments and return values are analogical to *Region.click()*.

double_click(*target_or_location*, *modifiers=None*)

Double click on a target or location using the left mouse button and optionally holding special keys.

Arguments and return values are analogical to *Region.click()*.

multi_click(*target_or_location*, *count=3*, *modifiers=None*)

Click N times on a target or location using the left mouse button and optionally holding special keys.

Arguments and return values are analogical to *Region.click()*.

click_expect(*click_image_or_location*, *expect_target*, *modifiers=None*, *timeout=60*, *retries=3*)

Click on an image or location and wait for another one to appear.

Parameters

- **click_image_or_location** (*Image or Location*) – image or location to click on
- **expect_target** – target to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timeout** (*int*) – time in seconds to wait for
- **retries** (*int*) – number of retries to reach expected target behavior

Returns

match obtained from finding the second target within the region

Return type

match.Match

click_vanish(*click_image_or_location*, *expect_target*, *modifiers=None*, *timeout=60*, *retries=3*)

Click on an image or location and wait for another one to disappear.

Parameters

- **click_image_or_location** (*Image or Location*) – image or location to click on
- **expect_target** – target to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timeout** (*int*) – time in seconds to wait for
- **retries** (*int*) – number of retries to reach expected target behavior

Returns

self

Return type

Region

click_at_index(*anchor*, *index=0*, *find_number=3*, *timeout=10*)

Find all instances of an anchor image and click on the one with the desired index given that they are horizontally then vertically sorted.

Parameters

- **anchor** (str or target.Target) – image to find all matches of
- **index** (*int*) – index of the match to click on (assuming >=1 matches), sorted according to their (x,y) coordinates
- **find_number** (*int*) – expected number of matches which is necessary for fast failure in case some elements are not visualized and/or proper matching result
- **timeout** (*int*) – timeout before which the number of matches should be found

Returns

match from finding the target of the desired index

Return type

match.Match

Note: This method is a good replacement of a number of coincident limitations regarding the Windows version of autopy and PyRO and therefore the (Windows) virtual user:

- autopy has an old BUG regarding capturing the screen at a region with boundaries, different than the entire screen -> subregioning which is the main way to deal with any kind of highly repeating and homogeneous interface, is totally unavailable here.
- PyRO cannot serialize generators, so this is an implementation of a "generator step" involving clicking on consecutive matches.
- The serialized virtual user now returns a list of proxified matches when calling find_all, but they are all essentially useless as they don't proxify their returned objects and cannot be sent back as arguments. The special proxy interface of the virtual user was implemented only to handle the most basic case - serialize the objects returned by the main shared class by proxifying them (turning them into remote objects as well, which already have a well-defined serialization method) and nothing more.

mouse_down(*target_or_location*, *button=None*)

Hold down an arbitrary mouse button on a target or location.

Parameters

- **target_or_location** (match.Match or location.Location or str or target.Target) – target or location to toggle on
- **button** (*int or None*) – button index depending on backend (default is left button) (see inputmap.MouseButton for extensive list)

Returns
match from finding the target or nothing if toggling on a known location

Return type
match.Match or None

mouse_up(*target_or_location*, *button=None*)

Release an arbitrary mouse button on a target or location.

Parameters

- **target_or_location** (match.Match or location.Location or str or target.Target) – target or location to toggle on
- **button** (*int or None*) – button index depending on backend (default is left button) (see inputmap.MouseButton for extensive list)

Returns
match from finding the target or nothing if toggling on a known location

Return type
match.Match or None

mouse_scroll(*target_or_location*, *clicks=10*, *horizontal=False*)

Scroll the mouse for a number of clicks.

Parameters

- **target_or_location** (match.Match or location.Location or str or target.Target) – target or location to scroll on
- **clicks** (*int*) – number of clicks to scroll up (positive) or down (negative)

- **horizontal** (*bool*) – whether to perform a horizontal scroll instead (only available on some platforms)

Returns
match from finding the target or nothing if scrolling on a known location

Return type
match.Match or None

drag_drop(*src_target_or_location*, *dst_target_or_location*, *modifiers=None*)

Drag from and drop at a target or location optionally holding special keys.

Parameters

- **src_target_or_location** (match.Match or location.Location or str or target.Target) – target or location to drag from
- **dst_target_or_location** (match.Match or location.Location or str or target.Target) – target or location to drop at
- **modifiers** (*[str]*) – special keys to hold during dragging and dropping (see inputmap.KeyModifier for extensive list)

Returns
match from finding the target or nothing if dropping at a known location

Return type
match.Match or None

drag_from(*target_or_location*, *modifiers=None*)

Drag from a target or location optionally holding special keys.

Arguments and return values are analogical to *Region.drag_drop()* but with *target_or_location* as *src_target_or_location*.

drop_at(*target_or_location*, *modifiers=None*)

Drop at a target or location optionally holding special keys.

Arguments and return values are analogical to *Region.drag_drop()* but with *target_or_location* as *dst_target_or_location*.

press_keys(*keys*)

Press a single key or a list of keys simultaneously.

Parameters
keys (*[str] or str (possibly special keys in both cases)*) – characters or special keys depending on the backend (see inputmap.Key for extensive list)

Returns
self

Return type
Region

Thus, the line `self.press_keys([Key.ENTER])` is equivalent to the line `self.press_keys(Key.ENTER)`. Other examples are:

```
self.press_keys([Key.CTRL, 'X'])
self.press_keys(['a', 'b', 3])
```

press_at(*keys*, *target_or_location*)

Press a single key or a list of keys simultaneously at a specified target or location.

This method is similar to *Region.press_keys()* but with an extra argument like *Region.click()*.

press_expect(*keys*, *expect_target*, *timeout=60*, *retries=3*)

Press a key and wait for a target to appear.

Parameters

- **keys** (*[str] or str (possibly special keys in both cases)*) – characters or special keys depending on the backend (see `inputmap.Key` for extensive list)
- **expect_target** – target to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timeout** (*int*) – time in seconds to wait for
- **retries** (*int*) – number of retries to reach expected target behavior

Returns
match obtained from finding the second target within the region

Return type
`match.Match`

press_vanish(*keys*, *expect_target*, *timeout=60*, *retries=3*)

Press a key and wait for a target to disappear.

Parameters

- **keys** (*[str] or str (possibly special keys in both cases)*) – characters or special keys depending on the backend (see `inputmap.Key` for extensive list)
- **expect_target** – target to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timeout** (*int*) – time in seconds to wait for
- **retries** (*int*) – number of retries to reach expected target behavior

Returns
self

Return type
Region

type_text(*text*, *modifiers=None*)

Type a list of consecutive character keys (without special keys).

Parameters

- **text** (*[str] or str (no special keys in both cases)*) – characters or strings (independent of the backend)
- **modifiers** (*[str]*) – special keys to hold during typing (see `inputmap.KeyModifier` for extensive list)

Returns
self

Return type
Region

Thus, the line `self.type_text(['hello'])` is equivalent to the line `self.type_text('hello')`. Other examples are:

```
self.type_text('ab3') # compare with press_keys()
self.type_text(['Hello', ' ', 'user3614']) # in cases with appending
```

Special keys are only allowed as modifiers here - simply call *Region.press_keys()* multiple times for consecutively typing special keys.

type_at(*text*, *target_or_location*, *modifiers=None*)

Type a list of consecutive character keys (without special keys) at a specified target or location.

This method is similar to *Region.type_text()* but with an extra argument like *Region.click()*.

click_at(*anchor*, *dx*, *dy*, *count=1*)

Clicks on a relative location using a displacement from an anchor.

Parameters

- **anchor** (`Match` or `Location` or `Target` or str) – target of reference for relative location
- **dx** (*int*) – displacement from the anchor in the x direction
- **dy** (*int*) – displacement from the anchor in the y direction
- **count** (*int*) – 0, 1, 2, ... clicks on the relative location

Returns
self

Return type
Region

Raises
`exceptions.ValueError` if *count* is not acceptable value

fill_at(*anchor*, *text*, *dx*, *dy*, *del_flag=True*, *esc_flag=True*, *mark_clicks=1*)

Fills a new text at a text box using a displacement from an anchor.

Parameters

- **anchor** (`Match` or `Location` or `Target` or str) – target of reference for the input field
- **text** (*str*) – text to fill in
- **dx** (*int*) – displacement from the anchor in the x direction
- **dy** (*int*) – displacement from the anchor in the y direction
- **del_flag** (*bool*) – whether to delete the highlighted text
- **esc_flag** (*bool*) – whether to escape any possible fill suggestions
- **mark_clicks** (*int*) – 0, 1, 2, ... clicks to highlight previous text

Returns
self

Return type
Region

Raises
`exceptions.ValueError` if *mark_click* is not acceptable value

If the delete flag is set the previous content will be deleted or otherwise the new text will be added in the end of the current text. If the escape flag is set an escape will be pressed after typing in order to avoid any entry suggestions from a dropdown list that could cover important image matching areas.

Since different interfaces behave differently, one might need a single, double or triple click to mark the already present text that has to be replaced.

select_at(*anchor*, *image_or_index*, *dx*, *dy*, *dw=0*, *dh=0*, *ret_flag=True*, *mark_clicks=1*, *tries=3*)

Select an option at a dropdown list using either an integer index or an option image if the order cannot be easily inferred.

Parameters

- **anchor** (Match or Location or Target or str) – target of reference for the input dropdown menu
- **image_or_index** (*str or int*) – item image or item index
- **dx** (*int*) – displacement from the anchor in the x direction
- **dy** (*int*) – displacement from the anchor in the y direction
- **dw** (*int*) – width to add to the displacement for an image search area
- **dh** (*int*) – height to add to the displacement for an image search area
- **ret_flag** (*bool*) – whether to press Enter after selecting
- **mark_clicks** (*int*) – 0, 1, 2, ... clicks to highlight previous text
- **tries** (*int*) – retries if the dropdown menu doesn't open after the initial click

Returns
self

Return type
Region

It uses an anchor image which is rather constant and a displacement to locate the dropdown location. It moves down to the option if index is used where index 0 represents the current selection.

To avoid the limitations of the index method, an image of the option can be provided and will be matched in the area with and under the dropdown list. This also handles cases where the option coincides with the previously selected option. For more details see the really cool note in the end of this method.

1.1.17 guibot.target module

1.1.17.1 SUMMARY

Classes and functionality related to sought targets on screen.

1.1.17.2 INTERFACE

class guibot.target.**Target**(*match_settings=None*)

Bases: object

Target used to obtain screen location for clicking, typing, validation of expected visual output, etc.

static from_data_file(*filename*)

Read the target type from the extension of the target filename.

Parameters
filename (*str*) – data filename for the target

Returns
target of type determined from its data filename extension

Return type
target.Target

Raises
errors.IncompatibleTargetFileError if the data file if of unknown type

static from_match_file(*filename*)

Read the target type and configuration from a match file with the given filename.

Parameters
filename (*str*) – match filename for the configuration

Returns
target of type determined from its parsed (and generated) settings

Return type
target.Target

property similarity

Getter for readonly attribute.

Returns
similarity required for the image to be matched

Return type
float

property center_offset

Getter for readonly attribute.

Returns
offset with respect to the target center (used for clicking)

Return type
location.Location

This clicking location is set in the target in order to be customizable, it is then taken when matching to produce a clicking target for a match.

load(*filename*, ***kwargs*)

Load target from a file.

Parameters
filename (*str*) – name for the target file

If no local file is found, we will perform search in the previously added paths.

save(*filename*)

Save target to a file.

Parameters
filename (*str*) – name for the target file

copy()

Perform a copy of the target data and match settings.

Returns
copy of the current target (with settings)

Return type
target.Target

with_center_offset(*xpos*, *ypos*)

Perform a copy of the target data with new match settings and with a newly defined center offset.

Parameters

- **xpos** (*int*) – new offset in the x direction
- **ypos** (*int*) – new offset in the y direction

Returns
copy of the current target with new center offset

Return type
target.Target

with_similarity(*new_similarity*)

Perform a copy of the target data with new match settings and with a newly defined required similarity.

Parameters
new_similarity (*float*) – new required similarity

Returns
copy of the current target with new similarity

Return type
target.Target

class guibot.target.**Image**(*image_filename=None*, *pil_image=None*, *match_settings=None*, *use_cache=True*)

Bases: *Target*

Container for image data supporting caching, clicking target, file operations, and preprocessing.

property filename

Getter for readonly attribute.

Returns
filename of the image

Return type
str

property width

Getter for readonly attribute.

Returns
width of the image

Return type
int

property height

Getter for readonly attribute.

Returns
height of the image

Return type
int

property pil_image

Getter for readonly attribute.

Returns
image data of the image

Return type
PIL.Image

load(*filename*, *use_cache=True*, ***kwargs*)

Load image from a file.

Parameters

- **filename** (*str*) – name for the target file
- **use_cache** (*bool*) – whether to cache image data for better performance

save(*filename*)

Save image to a file.

Parameters
filename (*str*) – name for the target file

Returns
copy of the current image with the new filename

Return type
target.Image

The image is compressed upon saving with a PNG compression setting specified by config.GlobalConfig.image_quality().

class guibot.target.**Text**(*value=None*, *text_filename=None*, *match_settings=None*)

Bases: *Target*

Container for text data which is visually identified using OCR or general text detection methods.

load(*filename*, ***kwargs*)

Load text from a file.

Parameters
filename (*str*) – name for the target file

save(*filename*)

Save text to a file.

Parameters
filename (*str*) – name for the target file

distance_to(*str2*)

Approximate Hungarian distance.

Parameters
str2 (*str*) – string to compare to

Returns
string distance value

Return type
float

class guibot.target.**Pattern**(*id*, *match_settings=None*)

Bases: *Target*

Container for abstracted data which is obtained from training of a classifier in order to recognize a target.

load(*filename*, ***kwargs*)

Load pattern from a file.

Parameters
filename (*str*) – name for the target file

save(*filename*)

Save pattern to a file.

Parameters
filename (*str*) – name for the target file

class guibot.target.**Chain**(*target_name*, *match_settings=None*)

Bases: *Target*

Container for multiple configurations representing the same target.

The simplest version of a chain is a sequence of the same match configuration steps performed on a sequence of images until one of them succeeds. Every next step in this chain is a fallback case if the previous step did not succeed.

load(*steps_filename*, ***kwargs*)

Load steps from a sequence definition file.

Parameters
steps_filename (*str*) – names for the sequence definition file

Raises
`errors.UnsupportedBackendError` if a chain step is of unknown type

Raises
`IOError` if an chain step line cannot be parsed

save(*steps_filename*)

Save steps to a sequence definition file.

Parameters
steps_filename (*str*) – names for the sequence definition file

1.2 Module contents

1.2.1 SUMMARY

Package with the complete guibot modules and functionality.

1.2.2 INTERFACE

PYTHON MODULE INDEX

g

INDEX

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

W

X

Y