guibot Documentation

Release 0.11

Plamen Dimitrov and Thomas Jarosch

Oct 30, 2020

Contents

CHAPTER 1

guibot package

1.1 Submodules

1.1.1 guibot.calibrator module

guibot.calibrator.**benchmark_blacklist = [('mixed', 'normal', 'mixed', 'east', 'hmm', 'adapt**
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.

__init__(*needle=None*, *haystack=None*, *config=None*)
Build a calibrator object for a given match case.

Parameters

- **haystack** (target.Image or None) – image to look in
- **needle** (target.Target or None) – target to look for

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

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 = None
time interval between mouse down and up in a click

click_delay = None
time interval after a click (in a double or n-click)

delay_after_drag = None
timeout before drag operation

delay_before_drop = None
timeout before drop operation

delay_before_keys = None
timeout before key press operation

delay_between_keys = None
time interval between two consecutively typed keys

rescan_speed_on_find = None
time interval between two image matching attempts (used to reduce overhead on the CPU)

smooth_mouse_drag = None
whether to move the mouse cursor to a location instantly or smoothly

screen_autoconnect = None
whether to perform complete initialization of the desktop control backend

preprocess_special_chars = None
whether to preprocess capital and special characters and handle them internally

save_needle_on_error = None
whether to perform an extra needle dump on matching error

image_logging_level = None
logging level similar to the python logging module

image_logging_step_width = None
number of digits when enumerating the image logging steps, e.g. value=3 for 001, 002, etc.

image_logging_destination = None
relative path of the image logging steps

display_control_backend = None
name of the desktop control backend

find_backend = None
name of the computer vision backend

contour_threshold_backend = None
name of the contour threshold backend

template_match_backend = None
name of the template matching backend

feature_detect_backend = None
name of the feature detection backend

feature_extract_backend = None
name of the feature extraction backend

feature_match_backend = None
name of the feature matching backend

text_detect_backend = None
name of the text detection backend

text_ocr_backend = None
name of the optical character recognition backend

deep_learn_backend = None
name of the deep learning backend

hybrid_match_backend = None
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
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

__getattribute__(*name*)
Return getattr(self, name).

__setattr__(*name*, *value*)
Implement setattr(self, name, value).

__enter__()

__exit__(*_)

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

Container for the configuration of all desktop 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.

__init__(*configure=True*, *synchronize=True*)
Build a container for the entire backend configuration.

Parameters

- **configure** (*bool*) – whether to also generate configuration
- **synchronize** (*bool*) – whether to also apply configuration

Available algorithms can be seen in the *algorithms* attribute whose keys are the algorithm types and values are the members of these types. The algorithm types are shortened as *categories*.

A parameter can be accessed as follows (example):

```
print(self.params["control"]["vnc_hostname"])
```

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.desktopcontrol module

1.1.4 guibot.errors module

exception guibot.errors.**GuiBotError**
Bases: Exception

GuiBot exception base class

exception guibot.errors.**FileNotFoundError**
Bases: *guibot.errors.GuiBotError*

Exception raised when a picture file cannot be found on disc

exception guibot.errors.**IncompatibleTargetError**
Bases: *guibot.errors.GuiBotError*

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

exception guibot.errors.**IncompatibleTargetFileError**
Bases: *guibot.errors.GuiBotError*

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

exception guibot.errors.**FindError**(*failed_target=None*)
Bases: *guibot.errors.GuiBotError*

Exception raised when an Image cannot be found on the screen

__init__(*failed_target=None*)
Build the exception possibly providing the failed target.

Parameters **failed_target** (target.Target or None) – the target that wasn't found

exception guibot.errors.**NotFindError**(*failed_target=None*)
Bases: *guibot.errors.GuiBotError*

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

__init__(*failed_target=None*)
Build the exception possibly providing the failed target.

Parameters **failed_target** (target.Target or None) – the target that was found

exception guibot.errors.**UnsupportedBackendError**
Bases: *guibot.errors.GuiBotError*

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

exception guibot.errors.**MissingHotmapError**
Bases: *guibot.errors.GuiBotError*

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

exception guibot.errors.**UninitializedBackendError**
Bases: *guibot.errors.GuiBotError*

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

1.1.5 guibot.finder module

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.

__init__(*value*, *min_val=None*, *max_val=None*, *delta=10.0*, *tolerance=1.0*, *fixed=True*, *enumerated=False*)
Build a computer vision parameter.

Parameters

- **value** (*bool or int or float or str or None*) – value of the parameter
- **min_val** (*int or float or None*) – lower boundary for the parameter range
- **max_val** (*int or float or None*) – upper boundary for the parameter range
- **delta** (*float*) – delta for the calibration and random value (no calibration if *delta* < *tolerance*)
- **tolerance** (*float*) – tolerance of calibration

- **fixed** (*bool*) – whether the parameter is prevented from calibration
- **enumerated** (*bool*) – whether the parameter value belongs to an enumeration or to a range (distance matters)

As a rule of thumb a good choice for the parameter delta is one fourth of the range since the delta will be used as standard deviation when generating a random value for the parameter from a normal distribution. The delta to tolerance ratio is basically the number of failing trials before the parameter converges and is usually set to ten.

__repr__()
Provide a representation of the parameter for storing and reporting.

Returns special syntax representation of the parameter

Return type str

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: *guibot.config.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

__init__(*configure=True*, *synchronize=True*)
Build a finder and its CV backend settings.

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: *guibot.finder.Finder*

Simple matching backend provided by AutoPy.

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using 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: *guibot.finder.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.

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's contour matching.

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: *guibot.finder.Finder*

Template matching backend provided by OpenCV.

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's template matching.

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: *guibot.finder.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).

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's feature matching.

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: *guibot.finder.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.

__init__(*classifier_datapath='.'*, *configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's cascade matching options.

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: *guibot.finder.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

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's text matching options.

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: *guibot.finder.TemplateFinder*, *guibot.finder.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.

__init__(*configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's template and 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: *guibot.finder.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.

__init__(*classifier_datapath='.'*, *configure=True*, *synchronize=True*)
Build a CV backend using OpenCV's text matching options.

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: *guibot.finder.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.

__init__(*configure=True*, *synchronize=True*)
Build a hybrid matcher.

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.

1.1.6 guibot.guibot module

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

__init__(*dc=None*, *cv=None*)
Build a guibot object.

Parameters

- **dc** (controller.Controller or None) – DC backend used for any desktop control
- **cv** (finder.Finder or None) – CV backend used for any target finding

We will initialize with default region of full screen and default desktop control and computer vision backends if none are provided.

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

1.1.7 guibot.guibot_proxy module

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`.

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.guibot.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.

__init__ (*dc=None*, *cv=None*)
Build a guibot object.

Parameters

- **dc** (`controller.Controller` or None) – DC backend used for any desktop control
- **cv** (`finder.Finder` or None) – CV backend used for any target finding

We will initialize with default region of full screen and default desktop control and computer vision backends if none are provided.

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 (image, text, etc.) 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`

Raises `errors.FindError` if no match is found

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 whether the target disappeared from the region

Return type bool

Raises errors.NotFindError if match is still found

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()`.

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_image_or_location=None*, *modifiers=None*, *timeout=60*)
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_image_or_location** (*Image or Location or None*) – image or location to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timout** (*int*) – time in seconds to wait for

Returns match obtained from finding the second target within the region

Return type `match.Match`

click_vanish(*click_image_or_location*, *expect_image_or_location=None*, *modifiers=None*, *timeout=60*)
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_image_or_location** (*Image or Location or None*) – image or location to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timout** (*int*) – time in seconds to wait for

Returns whether the second target disappeared from the region

Return type bool

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(*target_or_location=None*, *keys=None*)
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()`.

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(*target_or_location=None*, *text=''*, *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()`.

fill_at(*anchor*, *text*, *dx*, *dy*, *del_flag=True*, *esc_flag=True*, *mark_clicks=1*)
Fills a new text at a text box with variable content using an anchor image and a displacement from that image.

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*)

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

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.

1.1.8 guibot.guibot_simple module

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.

guibot.guibot_simple.**initialize**()

guibot.guibot_simple.**check_initialized**()

guibot.guibot_simple.**add_path**(*directory*)

guibot.guibot_simple.**remove_path**(*directory*)

guibot.guibot_simple.**find**(*target*, *timeout=10*)

guibot.guibot_simple.**find_all**(*target*, *timeout=10*, *allow_zero=False*)

guibot.guibot_simple.**sample**(*target*)

guibot.guibot_simple.**exists**(*target*, *timeout=0*)

guibot.guibot_simple.**wait**(*target*, *timeout=30*)

guibot.guibot_simple.**wait_vanish**(*target*, *timeout=30*)

guibot.guibot_simple.**get_mouse_location**()

guibot.guibot_simple.**hover**(*target_or_location*)

guibot.guibot_simple.**click**(*target_or_location*, *modifiers=None*)

guibot.guibot_simple.**right_click**(*target_or_location*, *modifiers=None*)

guibot.guibot_simple.**double_click**(*target_or_location*, *modifiers=None*)

guibot.guibot_simple.**multi_click**(*target_or_location*, *count=3*, *modifiers=None*)

guibot.guibot_simple.**click_expect**(*click_image_or_location*, *expect_image_or_location=None*, *modifiers=None*, *timeout=60*)

guibot.guibot_simple.**click_vanish**(*click_image_or_location*, *expect_image_or_location=None*, *modifiers=None*, *timeout=60*)

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

guibot.guibot_simple.**mouse_down**(*target_or_location*, *button=None*)

guibot.guibot_simple.**mouse_up**(*target_or_location*, *button=None*)

guibot.guibot_simple.**mouse_scroll**(*target_or_location*, *clicks=10*, *horizontal=False*)

guibot.guibot_simple.**drag_drop**(*src_target_or_location*, *dst_target_or_location*, *modifiers=None*)

guibot.guibot_simple.**drag_from**(*target_or_location*, *modifiers=None*)

guibot.guibot_simple.**drop_at**(*target_or_location*, *modifiers=None*)

guibot.guibot_simple.**press_keys**(*keys*)

guibot.guibot_simple.**press_at**(*target_or_location=None*, *keys=None*)

guibot.guibot_simple.**type_text**(*text*, *modifiers=None*)

guibot.guibot_simple.**type_at**(*target_or_location=None*, *text=''*, *modifiers=None*)

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

guibot.guibot_simple.**select_at**(*anchor*, *image_or_index*, *dx*, *dy*, *dw=0*, *dh=0*, *ret_flag=True*, *mark_clicks=1*)

1.1.9 guibot.imagelogger module

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

__init__()
Build an imagelogger object.

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.

1.1.10 guibot.inputmap module

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

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

__init__()
Build an instance containing an empty key map.

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: *guibot.inputmap.Key*

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

__init__()
Build an instance containing the key map for the AutoPy backend.

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

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

__init__()
Build an instance containing the key map for the xdotool backend.

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

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

__init__()
Build an instance containing the key map for the VNCDoTool backend.

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

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

__init__()
Build an instance containing the key map for the PyAutoGUI backend.

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

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

__init__()
Build an instance containing an empty modifier key map.

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: *guibot.inputmap.KeyModifier*

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

__init__()
Build an instance containing the modifier key map for the AutoPy backend.

class guibot.inputmap.**XDoToolKeyModifier**
Bases: *guibot.inputmap.KeyModifier*

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

__init__()
Build an instance containing the modifier key map for the xdotool backend.

class guibot.inputmap.**VNCDoToolKeyModifier**
Bases: *guibot.inputmap.KeyModifier*

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

__init__()
Build an instance containing the modifier key map for the VNCDoTool backend.

class guibot.inputmap.**PyAutoGUIKeyModifier**
Bases: *guibot.inputmap.KeyModifier*

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

__init__()
Build an instance containing the modifier key map for the PyAutoGUI backend.

class guibot.inputmap.**MouseButton**
Bases: `object`

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

__init__()
Build an instance containing an empty mouse button map.

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: *guibot.inputmap.MouseButton*

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

__init__()
Build an instance containing the mouse button map for the AutoPy backend.

class guibot.inputmap.**XDoToolMouseButton**
Bases: *guibot.inputmap.MouseButton*

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

__init__()
Build an instance containing the mouse button map for the xdotool backend.

class guibot.inputmap.**VNCDoToolMouseButton**
Bases: *guibot.inputmap.MouseButton*

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

__init__()
Build an instance containing the mouse button map for the VNCDoTool backend.

class guibot.inputmap.**PyAutoGUIMouseButton**
Bases: *guibot.inputmap.MouseButton*

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

__init__()
Build an instance containing the mouse button map for the PyAutoGUI backend.

1.1.11 guibot.location module

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

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

__init__(*xpos=0*, *ypos=0*)
Build a location object.

Parameters

- **xpos** (*int*) – x coordinate of the location
- **ypos** (*int*) – y coordinate of the location

__str__()
Provide a compact form for the location.

x
Getter for readonly attribute.

Returns x coordinate of the location

Return type int

y
Getter for readonly attribute.

Returns y coordinate of the location

Return type int

1.1.12 guibot.match module

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

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

__init__ (*xpos*, *ypos*, *width*, *height*, *dx=0*, *dy=0*, *similarity=0.0*, *dc=None*, *cv=None*)
Build a match object.

Parameters

- **xpos** (*int*) – x coordinate of the upleft vertex of the match region
- **ypos** (*int*) – y coordinate of the upleft vertex of the match region
- **width** (*int*) – x distance from upleft to downright vertex of the match region
- **height** (*int*) – y distance from upleft to downright vertex of the match region
- **dx** (*int*) – x offset from the center of the match region
- **dy** (*int*) – y offset from the center of the match region
- **similarity** (*float*) – attained similarity of the match region

__str__()
Provide the target location of the match distinguishing it from any location.

x
Getter for readonly attribute.

Returns x coordinate of the upleft vertex of the region

Return type int

y
Getter for readonly attribute.

Returns y coordinate of the upleft vertex of the region

Return type int

dx
Getter for readonly attribute.

Returns x offset from the center of the match region

Return type int

dy
Getter for readonly attribute.

Returns y offset from the center of the match region

Return type int

similarity
Getter for readonly attribute.

Returns similarity the match was obtained with

Return type float

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`

1.1.13 guibot.path module

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

1.1.14 guibot.region module

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.

__init__(*xpos=0*, *ypos=0*, *width=0*, *height=0*, *dc=None*, *cv=None*)
Build a region object from upleft to downright vertex coordinates.

Parameters

- **xpos** (*int*) – x coordinate of the upleft vertex of the region
- **ypos** (*int*) – y coordinate of the upleft vertex of the region
- **width** (*int*) – width of the region (xpos+width for downright vertex x)
- **height** (*int*) – height of the region (ypos+height for downright vertex y)
- **dc** (`controller.Controller` or None) – DC backend used for any desktop control
- **cv** (`finder.Finder` or None) – CV backend used for any target finding

Raises `UninitializedBackendError` if the region is empty

If any of the backends is not defined a new one will be initiated using the parameters defined in `config.GlobalConfig`. If *width* or *height* remains zero, it will be set to the maximum available within the screen space.

x
Getter for readonly attribute.

Returns x coordinate of the upleft vertex of the region

Return type int

y
Getter for readonly attribute.

Returns y coordinate of the upleft vertex of the region

Return type int

width
Getter for readonly attribute.

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

Return type int

height
Getter for readonly attribute.

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

Return type int

center
Getter for readonly attribute.

Returns center of the region

Return type `location.Location`

top_left
Getter for readonly attribute.

Returns upleft vertex of the region

Return type `location.Location`

top_right
Getter for readonly attribute.

Returns upright vertex of the region

Return type `location.Location`

bottom_left
Getter for readonly attribute.

Returns downleft vertex of the region

Return type `location.Location`

bottom_right
Getter for readonly attribute.

Returns downright vertex of the region

Return type `location.Location`

is_empty
Getter for readonly attribute.

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

Return type bool

last_match
Getter for readonly attribute.

Returns last match obtained from finding a target within the region

Return type `match.Match`

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 (image, text, etc.) 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`

Raises `errors.FindError` if no match is found

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 whether the target disappeared from the region

Return type bool

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()*.

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_image_or_location=None*, *modifiers=None*, *timeout=60*)
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_image_or_location** (*Image or Location or None*) – image or location to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timout** (*int*) – time in seconds to wait for

Returns match obtained from finding the second target within the region

Return type match.Match

click_vanish(*click_image_or_location*, *expect_image_or_location=None*, *modifiers=None*, *timeout=60*)

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_image_or_location** (*Image or Location or None*) – image or location to wait for
- **modifiers** (*[Key] or None*) – key modifiers when clicking
- **timout** (*int*) – time in seconds to wait for

Returns whether the second target disappeared from the region

Return type bool

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()*.

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()`.

fill_at(*anchor*, *text*, *dx*, *dy*, *del_flag=True*, *esc_flag=True*, *mark_clicks=1*)
Fills a new text at a text box with variable content using an anchor image and a displacement from that image.

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*)
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

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.

1.1.15 guibot.target module

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

__init__(*match_settings=None*)
Build a target object.

Parameters **match_settings** (finder.Finder or None) – predefined configuration for the CV backend if any

__str__()
Provide a constant name 'target'.

similarity
Getter for readonly attribute.

Returns similarity required for the image to be matched

Return type float

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: *guibot.target.Target*

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

__init__(*image_filename=None*, *pil_image=None*, *match_settings=None*, *use_cache=True*)
Build an image object.

Parameters

- **image_filename** (*str or None*) – name of the image file if any
- **pil_image** (PIL.Image or None) – image data - use cache or recreate if none
- **match_settings** (finder.Finder or None) – predefined configuration for the CV backend if any
- **use_cache** (*bool*) – whether to cache image data for better performance

__str__()
Provide the image filename.

filename
Getter for readonly attribute.

Returns filename of the image

Return type str

width
Getter for readonly attribute.

Returns width of the image

Return type int

height
Getter for readonly attribute.

Returns height of the image

Return type int

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*, *match_settings=None*)
Bases: *guibot.target.Target*

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

__init__(*value*, *match_settings=None*)
Build a text object.

Parameters

- **value** (*str*) – text value to search for
- **match_settings** (finder.Finder or None) – predefined configuration for the CV backend if any

__str__()
Provide a part of the text value.

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: *guibot.target.Target*

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

__init__(*id*, *match_settings=None*)
Build a pattern object.

Parameters

- **id** (*str*) – alphanumeric id of logit or label for the given pattern
- **match_settings** (finder.Finder or None) – predefined configuration for the CV backend if any

__str__()
Provide the data filename.

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: *guibot.target.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.

__init__(*target_name*, *match_settings=None*)
Build an chain object.

Parameters

- **target_name** (*str*) – name of the target for all steps
- **match_settings** (finder.Finder or None) – predefined configuration for the CV backend if any

__str__()
Provide the target name.

__iter__()
Provide an interator over the steps.

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

Python Module Index

g

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

W

X

Y