Part-Structured Spotting Documentation

Release 1.0

Maximilian Klingmann

March 21, 2016

Contents

sphinx-quickstart on Mon Jan 18 19:52:56 2016. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

CHAPTER 1

pss package

1.1 Submodules

1.2 pss.gui module

class pss.gui.**DistanceTransformPlot**(*target*)
Bases: object

create_distance_transform_figure(*name*, *dt*, *title*)

static draw_distance_transform(*ax*, *dt*, *vmin*, *vmax*)

class pss.gui.**GUIHandler**
Bases: object

This class starts up all the gui wanted.

static display_distance_transform(*target*)
Plots the summed up distance transform for the target image :param target: TargetDistanceTransform-object, which holds summed up distance transform

static display_query(*query*)
Plots the query entered in different forms :param query: The query image to plot

static display_target(*target*)

static show()

class pss.gui.**PrintNodes**(*symbol_group*)
Bases: object

Prints the Nodes locations to the log

print_positions()
Prints the Nodes

class pss.gui.**QueryPlot**(*query*)
Bases: object

This class is responsible for plotting the Query.

create_original_image_figure(*query*)

create_skeleton_figure(*query*)

create_tree_figure(*query*)

draw_skeleton_image(*ax*, *array*)
Draws skeletonized version of the original array :param ax: AxisSubplot to draw on :param array: Skeletonized-Array to draw onto AxisSubplot

draw_tree_image(*ax*, *root_node*, *center_of_mass*)
Draws tree :param ax: AxisSubplot to draw on :param root_node: Root-Node (is drawn green) :param center_of_mass: Center-Of-Mass Node (is drawn red)

static plot_center_of_mass(*ax*, *center_of_mass*)
Plots the center of mass into a given subplot :param ax: Subplot to draw the center of mass into :param center_of_mass: Center of mass to draw to the subplot

static plot_edge(*ax*, *parent*, *child*)
This plots edge between parent-child relation :param ax: Subplot to draw the edge into :param parent: Parent of the relation :param child: Child of the Relation

static plot_nodes(*ax*, *nodes*)
Plots the Nodes onto a subplot :param ax: Subplot to plot onto :param nodes: Nodes-Array to draw the nodes for

static plot_root_node(*ax*, *root_node*)
Highlights the root node, which is the node closest to the center of mass :param ax: Subplot to draw the root node into :param root_node: Root-Node to highlight

class pss.gui.**TargetPlot**(*target*)
Bases: object

create_target_figure(*name*, *original_array*, *title*)

static draw_target_image(*ax*, *target*)

pss.gui.**draw_array**(*ax*, *array*)
Fills the AxisSubplot with the array to draw :param ax: AxisSubplot to draw on :param array: Array to draw onto AxisSubplot

pss.gui.**setup_figure**(*name*)
This sets up the plot-figures, respectively the window. :param name: The name to set the windows title :return: The two empty subplots for the original image and the skeletonized one

pss.gui.**setup_plot**(*ax*, *array*, *title*)

Parameters

- **ax** – Empty subplot for the original image or the skeletonized image
- **array** – The array to fill up the subplot
- **title** – Title of the subplot

This fills up the subplots

1.3 pss.model module

This module is the core module of this application. It defines a SymbolGroup as the query object and builds up a parent-child tree by using the query image, skeletonizing it down to one pixel and placing nodes over the skeleton The nodes within the tree are represented by the custom class Node.

class pss.model.**DistanceTransform**(*query*, *target*)
Bases: object

This class calculates the distance transforms for all possible nodes of a query and sums them up to get the energy minimization for a given target.

build_distance_transform_to_parent(*child*)
Calculates the sum of the current total distance transform and a new distance transform by the child's position :param child: Child to calculate the new distance transform for :return: Added up distance transform from sum_distance_transform and newest child distance transform

calculate_distance_transform()
This starts the calculation of the all over distance transform and energy minimization.

calculate_height_and_width_of_distance_transform()
Calculates the maximum dimensions needed for adding up all distance transforms :return: the height, the width and the absolute starting point (root does not have to be and will very likely not be the most top-left pixel, since it's the center of mass)

recursively_add_up_all_distance_transforms(*node*)
Iterates over all nodes and their potential children adding up all distance transform created on the run. :param node: current node to add to the total distance transform

class pss.model.**Node**(*parent=None*, *position=None*, *offset=None*)
Bases: object

This class represents a node, defined by Howe et. al. It contains the reference to a parent-node and a list of its children-nodes, as well as the 2d-representation of that node. Additionally an offset is stored within the node. This shows the offset relative to its parent-node

add_child(*child*)
Adds a child to the children list. Added to avoid a breach of the law of demeter. :param child: The child to add.

calculate_offset()
This calculates the offset from the parent_node. The offset itself is to be read from the parent_node's location. eg. an offset of (1, 1) means the child is (1, 1) from the parents point of view.

set_parent(*parent*)
Setter for the Parent-Node :param parent: A Node, which should be the parent of the current node

class pss.model.**Query**(*query*, *index=0*, *bin=False*, *scale=1*)
Bases: object

This class represents one symbol group as defined in the given svg or the already rasterzized QImage if QueryPng is used. When using QuerySvg: It uses these svg-paths to create a QImage using QPainterPaths. This "original_image" is converted to a Numpy-array to get skeletonized by skimage. The array holding the skeletonized version of the QImage is then used to find joints and corners, also using skimage. These vectors are used to create the first nodes representing the initial symbol group paths given by their svg.

add_nodes_greedily()
Adds nodes between junctions and corners greedily. All corners and junctions are given as starting points. For each of these a neighborhood of DISTANCE is checked. If "True"'s within the enlarged_skeleton are found closer than DISTANCE away they get deleted from the true_list. If they are exactly DISTANCE away they are added to additional_nodes. This is done for every new "good" neighbor found until a corner_node is hit. If there is more than one "good" neighbors for a given node they get saved into a rest_list. This leads to a path getting finished before switching to a new path. The node creation therefor first checks one path until it hits a corner_node. When this happens the algorithm switches to the next path (starting point in rest_list). :return: The list of all nodes representing the QImage

add_relation(*real_child*, *real_parent*)
Adding the child to the parent and vice versa :param real_child: Child-Node to add :param real_parent: Parent-Node where Child-Node should be added

add_remaining_nodes()
This calls sub-functions to add the remaining nodes between junctions and corners. :return: The list of all nodes representing the QImage

build_up_tree()
Builds up the tree structure starting from the root node.

calculate_center_of_mass()
Calculates the center of mass by using the arithmetic mean of all node positions :return: The arithmetic mean of all node positions

create_bounding_box()
Creates the Bounding Box given all the paths given to the constructor :return: Bounding Box that surrounds all paths of this symbol group

create_copy_of_open_list_and_clear()
Creates a copy of the open_list and saves it inside a rest_list. After that the open_list is cleared :return: A list of all neighbors

create_original_image()
Creates a QImage and sets its background to COLOR_BG. After that the QImage is tried to get filled with the paths :return: QImage created out of QPainterPaths, which were given to the constructor

create_true_list()
Creates a List, which holds all indices of a "True"-appearance within the enlarged_skeleton. Only Non-Corner-/Non-Junction-Nodes are added to this list. :return: A list of 2D-Numpy-Arrays. The Arrays represent the appearance of a "True" in the enlarged_skeleton.

creating_all_relations_for_tree()
This starts the creation of all relations for the nodes starting from the root node.

decide_good_or_bad_neighbor(*node*)
Decides if a neighbor is seen as good neighbor, meaning a new node should be build there or as a bad neighbor, meaning it should get deleted, since it is too close to existing nodes. :param node: Current node which neighbors should get checked :return: List of "Good neighbors" and list of "Bad neighbors"

delete_bad_neighbors_from_true_list(*bad_neighbors*)
Deletes bad neighbors from the true_list, so they don't get checked ever again. They can't get "Good neighbors" any longer. :param bad_neighbors: List of "Bad neighbors", which should get deleted out of the true_list

enlarge_skeleton()
Adds two columns and two rows at the top, bottom, left and right to ensure the neighbor-search doesn't hit a border. :return: Numpy-Array, which is a column and row larger on each side.

fill_image_with_paths(*qpainter*, *image*)
Fills the QImage with the QPainterPaths :param qpainter: QPainter to let Qt draw into the QImage :param image: The empty QImage :return: The QImage, filled with QPainterPaths in a binary representation

find_closest_node(*current_tree*)
Finds node with closest distance to the already created tree with a greedy approach :param current_tree: Already created tree :return: Child-Node to add to the tree and Parent-Node, where child should get added

find_corners_and_junctions()
Finds all junctions and corners of the enlarged_skeleton, which represents a QImage :return: List of all junctions and corners found within the enlarged_skeleton Numpy-Array

find_root_node()
Finds the closes node to the center of mass :return: The node closed to the center od mass. This becomes the root node for the upcoming parent-child tree

static get_euclidean_addend(*a*, *b*, *i*)
Calculates one addend of the euclidean distance :param a: first node :param b: second node :param i: Index of the axis/dimension :return: One calculated addend of form (a.pos.item(i) - b.pos.item(i))^2

get_euclidean_distance(*a*, *b*)
Finds the euclidean distance between two given nodes :param a: first node :param b: second node :return: euclidean distance between node a and node b

has_node_more_than_one_neighbor()
Checks if a node has more than one neighbor :return: True if more than one neighbor

static is_neighbor_the_right_distance_away(*node*, *true_position*)
Checks if a neighbor is the right distance to be seen as a "Good neighbor" :param node: Current node :param true_position: Position of potential neighbor :return: True if distance is just right to be seen as a "Good neighbor"

static is_neighbor_too_close(*node*, *true_position*)
Checks if a neighbor is too close to its node (Too close meaning less than DISTANCE away) :param node: Current node :param true_position: Position of potential neighbor :return: True if distance is too close

is_node_completely_unknown(*new_node*)
Checks if a given node is completely unknown, meaning it is neither in the closed_list nor in the corner_nodes :param new_node: Node to be checked :return: True if completely unknown (not in self.closed_list and not in self.corner_nodes)

still_neighbors_to_check(*rest_list*)

Parameters **rest_list** – List of other Neighbors found when started to populate from corner_node

Returns True nodes still in open_list or rest_list

try_to_fill_image_with_paths(*image*)
Tries filling up the QImage with QPainterPaths :param image: The empty QImage :return: The QImage, filled with QPainterPaths in a binary representation

static update_tree(*current_tree*, *real_child*)
Adds new child to the current tree and removes it from the nodes list :param current_tree: Already created tree :param real_child: Child to add to tree and remove from nodes list

class pss.model.**Target**(*target*, *bin=False*, *scale=1*)
Bases: object

This class represents the target image, which is usually the tablet to scan. It is responsible for rasterizing a given svg to a QImage and converting this QImage to a valid ndarray, which can be used by skimage, scipy and matplotlib. In case TargetBin was used to load in the target-image the conversion to a QImage is not needed, since TargetBin already holds a QImage with the image loaded into it. In that case this step is skipped and only the conversion to a valid ndarray is done.

create_image(*renderer*)
Uses a QSvgRenderer to return a QImage from it :param renderer: the QSvgRenderer object given by TargetSvg :return: QImage representation of the QSvgRenderer

pss.model.**convert_qimage_to_ndarray**(*image*)
Converts a given QImage into a Numpy-Array :param image: :return: Boolean Numpy-Array representing the QImage. "True" = Foreground, "False" = Background

pss.model.**create_skeleton**(*name*, *original_array*)
Skeletonizes a given Numpy-Array to a 1px width. See skimage-documentation for skeletonize :param original_array: :param name: Name of the QImage (For Logging purposes) :return: Skeletonized Numpy-Array

pss.model.**print_time**(*end_time*, *start_time*)
This method just prints how long a certain function took :param end_time: Timestamp taken after finished building the tree :param start_time: Timestamp taken before started building the tree

pss.model.**set_background**(*color*, *image*)
Sets the background color of the QImage :param image: QImage which background should be changed :param color: Color to change background to

1.4 pss.svg module

class pss.svg.**QuerySvg**(*path*, *infix=''*)
Bases: object

This class opens and manages as the query image.

static get_symbol_group_size(*symbol_group*)
Returns the number of QPainterPaths within a given SymbolGroup :param symbol_group: The SymbolGroup to find the size of :return: The number of Paths within that SymbolGroup

static load_svg(*infix*, *path*)
Opens an SVG-file :param infix: This infix is used by the external library elka_svg.py to check for specific groups within the svg. :param path: Path, where the SVG file is located :return: names and lists of QPainterPaths of the SymbolGroups read from the SVG

class pss.svg.**TargetSvg**(*path*)
Bases: object

This class opens and manages a given svg file as the target image.

pss.svg.**handle_file_not_existing**(*path*)
Checks whether the given file is existing :param path: Path of file to check for existence :raises: FileNotFoundError if path not found

1.5 Module contents

p

I

L

N

P

Q

R

S

T

U