Проект: бинаризация объёма (томограммы).

Исполнитель: Степан Корней

Версия: 22-12-2024-c

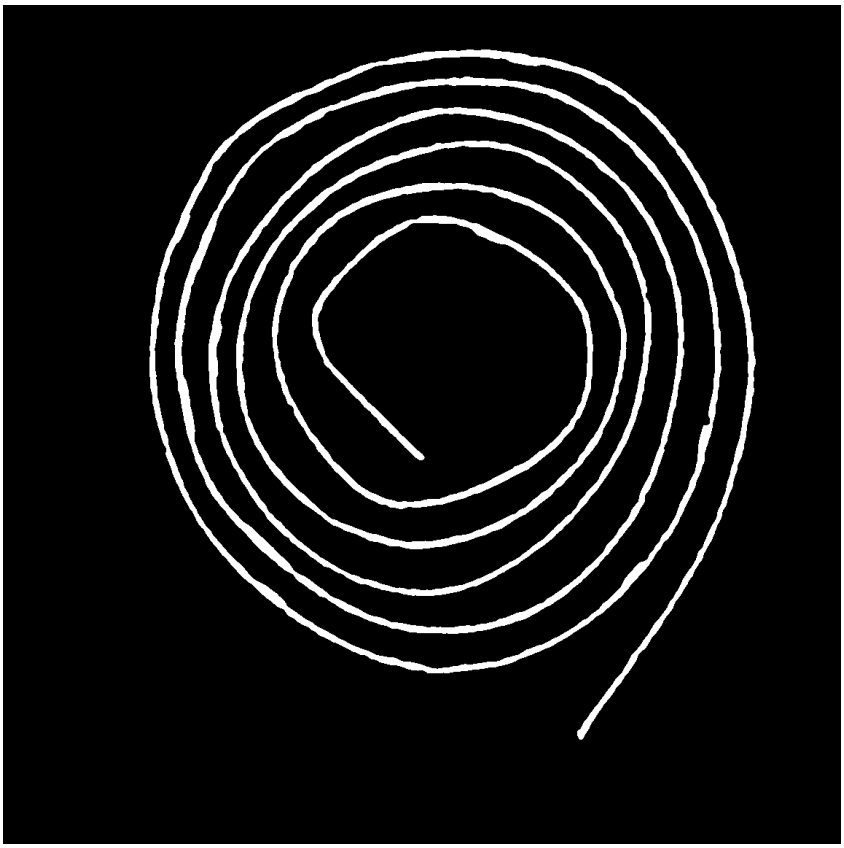
ТЗ:

1. Программа-бинаризатор.
    - Входные данные: 3d массив скаляров. Может быть задан в виде:
        a. Последовательности двумерных одноцветных картинок одинакового пиксельного размера в формате tiff. Каждый файл содержит информацию о слое 3d массива скаляров.
        b. Переменных типа ndarray из библиотеки numpy. Размерность и другие требования уточняйте в документации к каждой функции.

      В каждом элементе записана “плотность вещества” в соотв. точке пр-ва. В этом массиве всегда результат томографии свитка. Близкие к 1 значения - свиток есть, близкие к 0 - воздух. Возможен шум. Посторонних предметов в массиве не запечатлено.

      Между частями свитка много пустого пространства (не менее примерно 1-2 толщины свитка, на каждом слое может быть не более 1 точки касания свитка с самим собой). Свиток не разорван, имеет примерно постоянную толщину (примерно от ½ до ⅔ средней толщины). Свиток скручен и расположен так, что ось скручивания примерно перпендикулярна плоскости каждой картинки, как на

иллюстрации ниже.



Типичная картинка, подаваемая на вход бинаризатора.

В случае numpy.ndarray, ожидается, что эта ось примерно параллельна оси массива, имеющей индекс 0.

- Выходные данные: имеют формат с таким же описанием, как входные данные, но элементы могут принимать только 2 значения: 0 и 1. 0 - система приняла решение, что свиток не проходит через эту точку, 1 - решила, что проходит.

2. Демонстрационный датасет из четырёх свитков. (4 многотомных архива) Так как датасет занимает огромное количество места на диске, на некоторую его часть будет дана только ссылка.
3. Программа, запускающая программу из пункта 1 и делающая вычисление функций оценки качества. Значения функций оценки качества выводятся в консоль и записываются в файл.
    - Формат входных данных:
        a. Без входа. Система может работать без входных данных, тестируя алгоритмы на данных по умолчанию.

    b. Имена файлов. Последовательность строк, каждая из которых - путь к файлу, содержащему вход или Ground Truth.
    c. Имя файла для записи результатов, если это требуется.
- Формат выходных данных:
    a. Текст в консоль. Программа может автоматически вывести в консоль результаты тестов.
    b. Текст в файл. Аналогично, программа может вывести результаты тестов в файл, название которого можно указать на входе.

4. Инструкция по пользованию программами из пунктов 1 и 3.
5. Примерные оценки того, какое качество должна давать программа из пункта 1 на валидных данных.
6. config-файл с настройками, подобранными для работы с демонстрационными данными. На их основе можно подобрать настройки для обработки пользовательских данных.

Инструкция

Код расположен на этом репозитории:
https://github.com/stevegamer1/scrolls-2024
Чтобы получить исходный код, этот репозиторий следует склонировать на целевой компьютер, используя git clone.

Установка требуемых пакетов

Проект написан на языке Python. Для запуска требуется интерпретатор Python и менеджер пакетов Python - pip.
Когда код проекта получен, нужно установить необходимые проекту пакеты. Для этого есть файл requirements.txt, в котором содержится список необходимых пакетов. Можно установить их вручную или с помощью, например, команды pip с флагом -r.
Рекомендуется использовать механизм виртуальных окружений, чтобы изолировать окружение, нужное проекту.

Как запустить бинаризацию

Когда код проекта получен, установив необходимые пакеты и активировав виртуальное окружение, если вы его используете, нужно сделать хотя бы одно из двух:

1. Вызвать интересующие функции и передав необходимые аргументы (подробности уточняйте в разделе API).
2. Запустить с помощью интерпретатора Python скрипт default_quality_measurements.py. Это приведёт к вычислению функций оценки качества, выводу результатов на экран, и записи результатов в файл “algorithms results.txt” в той же папке, что и скрипт.

Как запустить тесты

Тесты функций оценки качества запускаются автоматически при подключении модуля quality_funcs. Можно также запустить этот файл с помощью интерпретатора Python.

Использованные алгоритмы и функции оценки качества

Какие алгоритмы использовались

- Порог (одноцветные пиксели классифицируются по признаку превышения некоторого значения цвета).
- Раздутие и эрозия (минимум и максимум по окну в бинарном изображении).
- Фильтр Гаусса.
- Фильтр Кэнни.
- Скелетонизация (thinning). Используется Zhang-Suen thinning algorithm.

Какие функции оценки качества использовались

- IoU (intersection over union). Эта функция склонна поощрять не вполне релевантные особенности сравниваемых изображений, но тем не менее используется, так как является характеристикой схожести изображений. В этой задаче много внимания на неё обращать не рекомендуется.
- RAD - относительная разность площадей:
  $\frac{|B|-|A|}{|B|}$, то есть отношение разности площадей к площади Ground Truth. Это грубая функция и иногда позволяет выявить недостаточную или чрезмерную сегментацию. A и B здесь - результат работы алгоритма и Ground Truth.

- Абсолютное среднее расстояние (ASD):
  $\frac{D_1(A,B)+D_1(B,A)}{|A|+|B|}$, где $D_n(U,\ V)\ =\ \sum_{p\in S(U)}(d(p,\ V))^n$, $d(p,\ V)$ - расстояние между точкой p и множеством V, а $S(U)$ - поверхность, то есть множество пикселей, у каждого из которых есть не включённый в U сосед, а соседями считаются пиксели с общей стороной или вершиной.

- Корень среднего квадрата расстояния (RMSD):
  В тех же обозначениях, $\sqrt{\frac{D_2(A,B)+D_2(B,A)}{|A|+|B|}}$

- Хаусдорфово расстояние.

- Хаусдорфово-95 расстояние, где вместо максимума берётся 95-й перцентиль:
  $max(P_{95}(D_1(A)),\ P_{95}(D_1(B)))$, где $D_1(U)$ - функция,
  $D_1(U)(p)\ =\ d(p,\ U)$

RAD и IoU - наиболее грубые функции. Если остальные функции показывают высокое качество, а эти две - нет, то, скорее всего, есть какая-то большая разница между именно внутренними частями белых регионов на изображениях.
ASD, RMSD, Хаусдорфово-95 и Хаусдорфово расстояния оценивают близость именно поверхностей белых регионов. Основное отличие в чувствительности к выбросам, именно в указанной последовательности они расставлены по нарастанию этой чувствительности, то есть ASD менее чувствительно, чем RMSD, и т. д.

Эксперименты над функциями оценки качества
С помощью функции do_quality_functions_experiments() из модуля default_quality_measurements были проведены эксперименты для подтверждения этих теоретических закономерностей.

Было выбрано 6 модификаций:

1. Сдвиг на 1 пиксель.
2. Сдвиг на 10 пикселей.
3. Добавление шума на границе. Под границей понимается разность dilated и eroded версий Ground Truth. Для каждого пикселя 5% вероятности, что он будет инвертирован.
4. Добавление шума внутри. Отличие от предыдущего эксперимента в том, что шум применялся внутри, на eroded-версии региона, и инвертировалось 70% пикселей.
5. Раздутие (dilate) на 3 пикселя.
6. Эрозия (erode) на 3 пикселя.

Были отобраны слои ground truth для демонстрационного свитка-1 (см. раздел “Данные”), а именно слои начиная с нулевого через каждые 400 слоёв, то есть 0, 400, и т. д.
Для каждого слоя применялись вышеуказанные 6 модификаций, и вычислялись функции оценки качества между ним и результатом модификации. Затем каждый результат усреднялся по слоям, и для каждой модификации получался профиль: как какие модификации на неё влияют.
В таблице ниже приведены результаты:

	Shift-X-1	Shift-X-10	Noise 5%, border	Noise 70%, inside	Erode 3 pixels	Dilate 3 pixels
IoU	0.91	0.35	0.97	0.5	0.87	0.88
RAD	0	0	0	1.01	0.15	-0.12
ASD	0.51	4.79	0.21	1.24	0.7	0.7
RMSD	0.72	5.5	0.56	2.17	0.84	0.84
Хаусдорф	1	10	2.83	8.38	1.55	1.89
Хаусдорф-95	1	9.49	1	0	1	1

Первое, что можно заметить - две самые правые строки сильно похожи. Значимое различие видно лишь у относительной разности площадей (RAD) и хаусдорфова расстояния. Это значит, что данные функции качества следует использовать при оценке качества бинаризации, ведь без них было бы трудно различить две подобные ситуации.

Хаусдорфова-95 функция не превосходит хаусдорфово расстояние, но иногда равна ему, что означает, что последнее действительно склонно к большему увеличению с увеличением неодинаковости изображений. При переходе от Shift-1 к Shift-10 это менее заметно, но в случае эрозии и раздутия видно, что возрастание происходит резче.

В целом, таблица подтверждает высказывание о том, что ASD, RMSD, и варианты хаусдорфова расстояния примерно упорядочены по чувствительности к выбросам.

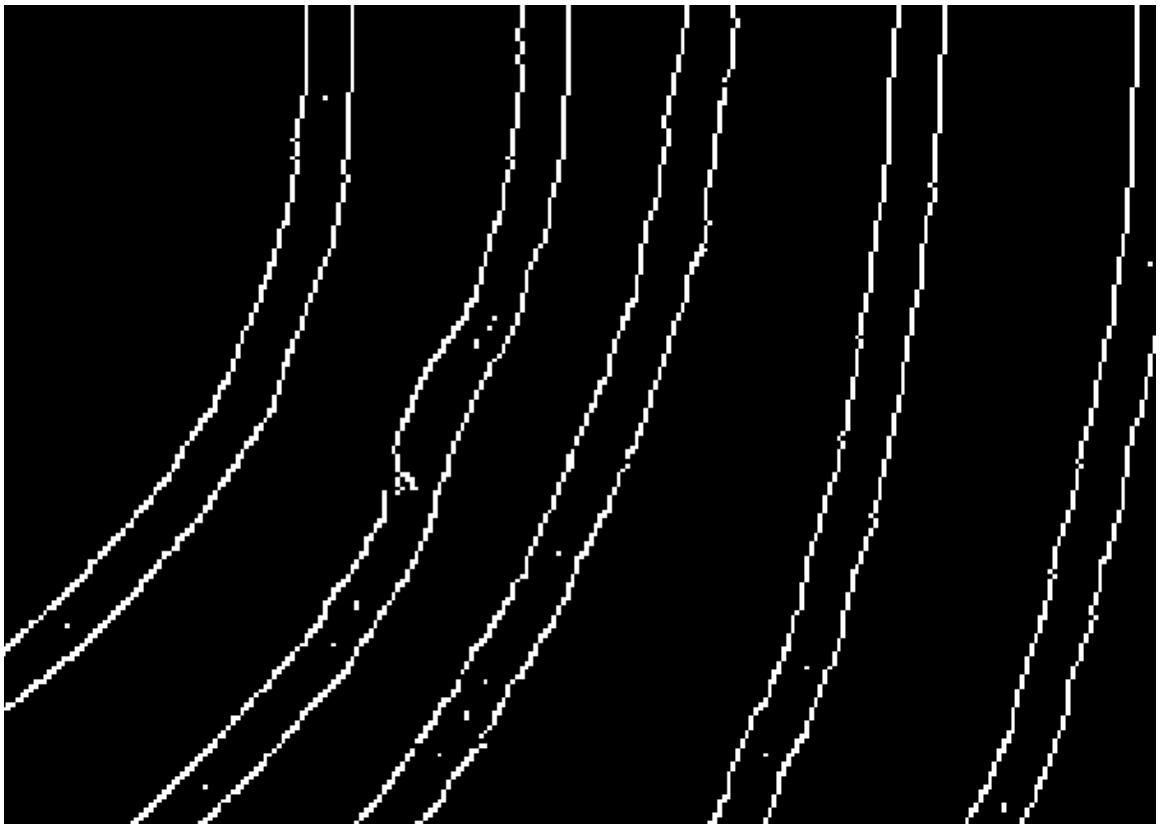
IoU сильно падает всего лишь от смещения в примерно 1 толщину свитка (10 пикселей), но практически не обращает внимания на поверхностный шум. Это значит, что на данную функцию оценки обращать большого внимания не стоит.

Оценка качества

Алгоритмы бинаризации

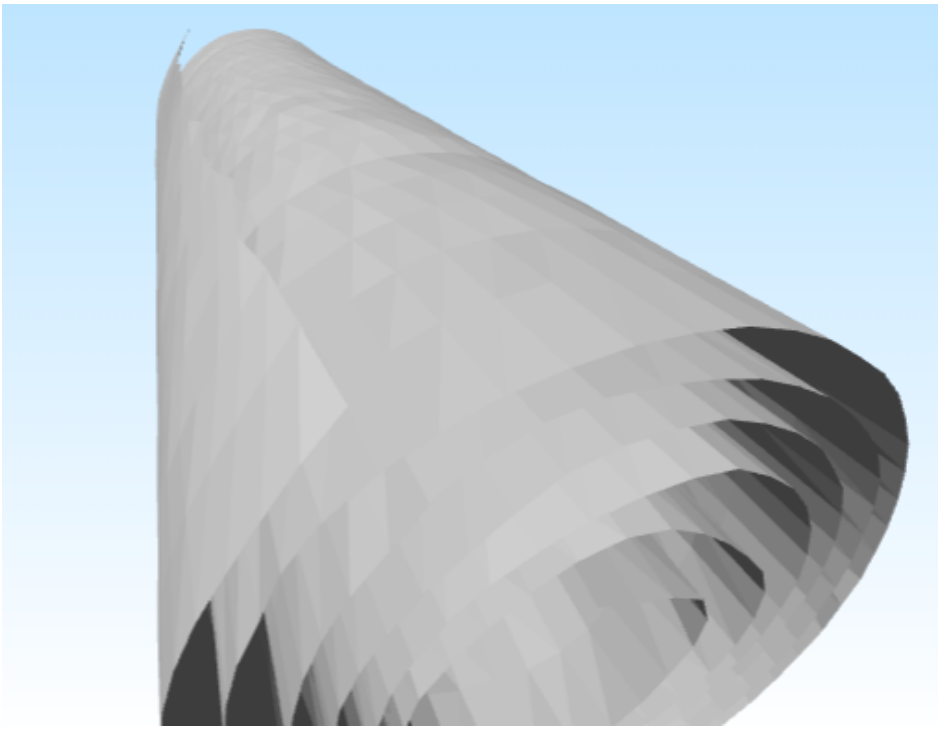
Было реализовано и настроено 3 алгоритма бинаризации:

1. smooth1: сначала с помощью порога удаляется шум, применяется фильтр Гаусса, снова берётся порог, и, наконец, производится эрозия.
   Результат выглядит обычно примерно как на иллюстрации:
   
2. Canny: бинаризация на фильтре Кэнни. Суть сосредоточена в фильтре Кэнни и алгоритме скелетонизации (thinning). Сечение свитка - фигура простая, поэтому разные алгоритмы давали практически один и тот же результат. Используемая библиотека OpenCV производит скелетонизацию по алгоритму (Zhang-Suen).
   Промежуточный результат выглядит примерно как на иллюстрации:



От этих границ алгоритм стремится добиться неразрывности и заполнить внутренние чёрные регионы белым цветом.

3. mesh: над результатом работы smooth1 производится thinning по тому же алгоритму, а затем на получившейся кривой через равные расстояния помещаются точки. Это происходит лишь на некоторых слоях (назовём их **опорными**). В итоге получается меш, который интерполируется между слоями, на которых произведена процедура. Интерполированный меш утолщается, результат выдаётся в ответ.
   Меш обычно выглядит примерно как на иллюстрации:



Анализ на демонстрационных данных

Для оценки качества работы алгоритмов были проделаны эксперименты, похожие на те, что были выполнены для анализа самих функций оценки качества.

Для этого использовалась функция assess_algorithms() из модуля default_quality_measurements.

Вместо искусственных модификаций брались результаты работы алгоритмов.

Измерения проводились на тех же слоях, то есть начиная с нулевого через каждые 400.

Итак, результаты:

	smooth1	Canny filter-based	mesh
IoU	0.65	0.83	0.85
RAD	0.53	0.03	-0.08
ASD	2.13	0.97	0.98
RMSD	2.22	3.92	1.31
Хаусдорф-95	3.05	3.09	2.41
Хаусдорф	6.18	63.46	9.84

Эти результаты, получились, конечно, не сразу. Функция assess_algorithms() помогала понять, какие алгоритмы в каком смысле ошибаются. Затем алгоритмы совершенствовались.
Среднее расстояние Хаусдорфа однозначно говорит, что ещё есть куда совершенствовать алгоритм бинаризации, основанный на фильтре Кэнни. Действия на пути к этому предпринимались, и алгоритм был улучшен, но он ещё не совершенен.
Видно, что алгоритм mesh, основанный на построении сетки и её раздутии, превосходит по качеству алгоритм smooth1, результаты которого использует, и алгоритм, основанный на фильтре Кэнни. Алгоритм на фильтре Кэнни стремится заполнить границы сечения свитка, которые иногда получаются разрывными.

Данные

Данные можно загрузить по ссылкам:
https://zenodo.org/records/13640029
https://zenodo.org/records/7221350
https://zenodo.org/records/7324316
https://zenodo.org/records/7324321
Первая из ссылок ведёт на страницу, где можно скачать данные, относящиеся к уже упомянутому свитку-1.
Файлы, содержащие необходимые данные, называются scroll00{номер свитка}.rec{сторона одного изображения в пикселях}.zip.{номер тома многотомного архива}
Нужно скачать все тома архива (рекомендуется делать это с помощью инструментов скачивания больших файлов, например curl или JDownloader), распаковать их, и поместить файлы в папку, из которой их будет читать вызываемая функция. По умолчанию, это папка scroll001.rec_1150, находящаяся на одном уровне с папкой, в которой расположены файлы модулей. Можно указать другой путь в файле config.py или передать в параметр вызываемой функции.
Ground truth функции по умолчанию ожидают в папке ground_truth,

находящейся на одном уровне с файлами модулей, как это уже и есть на репозитории.
На репозитории в папке ground_truth можно найти файл scroll001_ground_truth.nrrd. Это ground truth, использованная для изучения свойств алгоритмов и функций оценки качества. Она была получена с использованием Slicer3D путём применения порога, сглаживания и ручной доработки. Её можно открыть в Slicer3D или считать из файла с помощью пакета pynrrd.

API

Начиная со следующей страницы, начинается документация.

scroll-binarization

Stepan Korney

Dec 22, 2024

ONE

CONFIG MODULE

Конфигурационный файл для свитка 1. Содержит параметры для алгоритмов бинаризации, с которыми алгоритмы хорошо бинаризируют этот свиток.

TWO

DEFAULT_QUALITY_MEASUREMENTS MODULE

Примеры предполагаемого использования функций оценки качества и алгоритмов бинаризации.

default_quality_measurements.**assess_algorithms**(*output_filename: str | None*, *should_print: bool = True*)

Вычислить функции оценки качества для каждого алгоритма, результаты вывести на экран и записать в файл.

output_filename - путь к файлу, в который нужно записать результаты.

should_print - True, если нужно выводить результаты в консоль.

default_quality_measurements.**do_quality_functions_experiments**(*output_filename: str | None*, *should_print: bool = True*)

Вычислить функции оценки качества для ground truth и её искусственно искажённых версий для изучения свойств функций оценки качества.

output_filename - путь к файлу, в который нужно записать результаты, или None, если не нужно писать результаты в файл.

should_print - True, если нужно выводить результаты в консоль.

default_quality_measurements.**get_scroll1_ground_truth**()

Получить ground truth для свитка-1 как numpy-массив.

THREE

QUALITY_FUNCS MODULE

Функции оценки качества и тесты для них.

quality_funcs.calc_ASD(*arr1_surface: ndarray*, *arr1_kdtree: cKDTree*, *arr2_surface: ndarray*, *arr2_kdtree: cKDTree*)

Вычислить Average Symmetric Distance. Если оно больше, то массивы более ‘разные’.

arr1_surface, arr2_surface - двумерные массивы. По оси 0 - номер точки, по оси 1 - координаты точки.

arr1_kdtree, arr2_kdtree - scipy.spatial.cKDtree, построенные на arr1_surface и arr2_surface.

quality_funcs.calc_Hausdorff(*arr1_surface: ndarray*, *arr1_kdtree: cKDTree*, *arr2_surface: ndarray*, *arr2_kdtree: cKDTree*)

Хаусдорфово расстояние. Maximum Symmetric Distance. Если оно больше, то массивы более ‘разные’.

arr1_surface, arr2_surface - двумерные массивы. По оси 0 - номер точки, по оси 1 - координаты точки.

arr1_kdtree, arr2_kdtree - scipy.spatial.cKDtree, построенные на arr1_surface и arr2_surface.

quality_funcs.calc_Hausdorff95(*arr1_surface: ndarray*, *arr1_kdtree: cKDTree*, *arr2_surface: ndarray*, *arr2_kdtree: cKDTree*)

Вычислить почти Хаусдорфово расстояние, но вместо максимума берётся 95-й перцентиль. Менее чувствительна к выбросам, чем Хаусдорфово расстояние.

arr1_surface, arr2_surface - двумерные массивы. По оси 0 - номер точки, по оси 1 - координаты точки.

arr1_kdtree, arr2_kdtree - scipy.spatial.cKDtree, построенные на arr1_surface и arr2_surface.

quality_funcs.calc_IoU(*arr1: ndarray*, *arr2: ndarray*)

Вычислить intersection over union двух булевых массивов.

quality_funcs.calc_RMSD(*arr1_surface: ndarray*, *arr1_kdtree: cKDTree*, *arr2_surface: ndarray*, *arr2_kdtree: cKDTree*)

Вычислить Root Mean Symmetric Distance. Если оно больше, то массивы более ‘разные’. Более чувствительна к выбросам, чем ASD.

arr1_surface, arr2_surface - двумерные массивы. По оси 0 - номер точки, по оси 1 - координаты точки.

arr1_kdtree, arr2_kdtree - scipy.spatial.cKDtree, построенные на arr1_surface и arr2_surface.

quality_funcs.**calc_relative_area_diff**(*arr1: ndarray*, *arr2: ndarray*)

Вычислить relative area difference двух булевых массивов. Функция асимметрична. arr2 обычно ground truth.

quality_funcs.**get_surface_cKDtree**(*image: ndarray*)

Получить координаты пикселей, образующих границы (8-соседство) белых регионов на булевом изображении. Возвращает массив с координатами и scipy.spatial.cKDtree, построенное на нём.

image - изображение, на котором надо найти поверхностные пиксели.

quality_funcs.**quality_functions_tests**()

Проверка правильной работы функций оценки качества.

FOUR

ALGORITHMS MODULE

Алгоритмы бинаризации свитков.

class algorithms.**Point**(*x=0*, *y=0*)

Bases: object

Информация о точке на плоскости. Заменитель 2-тьюплов. Предпочтителен там, где используется, потому что поля имеют имена.

algorithms.**build_broken_line**(*image: ndarray*, *start:* Point, *distance_threshold: int*)

Построить ломаную, проходящую вдоль линии, нарисованной на бинарном изображении. Ожидается, что на изображении линия присутствует, и что она не касается краёв изображения.

image - 2d массив, изображение тонкой (~3 пикселя - идеальная толщина) белой линии без разрывов и самопересечений.
start - Point, семя алгоритма. Построение ломаной начинается в этой точке.
distance_threshold - расстояние, которое выдерживается между вершинами ломаной.

algorithms.**canny_based_segmentation**(*data: ndarray*, *gauss_sigma: float = 5.0*, *canny_thresh1: float = 0.1*, *canny_thresh2: float = 10.0*, ***smooth_from_canny_args*)

Получить бинаризацию сечения свитка, запечатлённого на изображении, с помощью алгоритма, основанного на фильтре Кэнни.

data - 2d массив, сырое (только что из файла) изображение сечения свитка.
gauss_sigma - какой параметр сигма брать для сглаживающего фильтра Гаусса. Размер окна вильтра вычисляется автоматически.
canny_thresh1, canny_thresh2 - пороги для фильтра Кэнни.
smooth_from_canny_args - какие параметры передать при вызове smooth_from_canny. Смотрите описание smooth_from_canny.

algorithms.**fill_small_black_regions**(*bw_image: ndarray*)

Заполнить все регионы, состоящие из нулей, кроме самого большого, числом 255.

bw_image - 2d массив, чёрно-белое (0 и 255) восьмибитное изображение.

algorithms.find_starting_point(*image: ndarray*)

Найти какую-нибудь белую точку на чёрно-белом изображении.

algorithms.get_line(*data: ndarray*, *distance_threshold: int*, *should_make_copy_with_line: bool = False*, ***smooth1_args*)

Построить линию, описывающую форму сечения свитка, запечатлённого на изображении.

data - 2d массив, сырое изображение сечения свитка.

distance_threshold - расстояние, которое выдерживается между вершинами ломаной.

should_make_copy_with_line - bool, если True то вторым аргументом возвращается изображение с нарисованной на нём результирующей линией.

smooth1_args - какие параметры передать при вызове smooth1. Смотрите описание smooth1.

algorithms.get_line_for(*path: str*, *distance_threshold: int*, *should_make_copy_with_line: bool = False*, ***smooth1_args*)

Применить алгоритм get_line к изображению, записанному в файл.

path - путь к файлу изображения.

distance_threshold - расстояние, которое выдерживается между вершинами ломаной.

should_make_copy_with_line - bool, если True то вторым аргументом возвращается изображение с нарисованной на нём результирующей линией.

smooth1_args - какие параметры передать при вызове smooth1. Смотрите описание smooth1.

algorithms.get_scroll_shape_as_mesh(*filenames: list | None = None*, *delta_z: int | None = None*, *distance_threshold: int | None = None*, ***smooth1_args*)

Создать 3d-модель, описывающую форму свитка. Возвращает 2 списка - точек и треугольников. Список точек - список списков из 3 элементов. 3 элемента - координаты точки в 3d. Список треугольников - список списков из 3 элементов. 3 элемента - индексы (начинаются с 0) точек, образующих треугольник.

filenames - список строк, пути к tiff-файлам, из которых следует читать слои томографии свитка. Если None, то значение берётся из модуля config.

delta_z - каким считать росстояние между слоями. Если None, то значение берётся из модуля config.

distance_threshold - расстояние, которое выдерживается между вершинами ломаной.

smooth1_args - какие параметры передать при вызове smooth1. Смотрите описание smooth1.

algorithms.point_distance(*p1:* Point, *p2:* Point)

Евклидово расстояние между двумя точками на плоскости.

algorithms.save_as_obj(*points: list*, *triangles: list*, *filename: str*)

Сохранить 3d модель, описанную точками и треугольниками, в формате obj. Предполагаемое использование: сохранять результат работы get_scroll_shape_as_mesh.

points - список списков из 3 элементов. 3 элемента - координаты точки в 3d.

triangles - список списков из 3 элементов. 3 элемента - индексы (начинаются с 0) точек, образующих треугольник.

filename - путь к результирующему файлу, например “my mesh.obj”

`algorithms.smooth1`(*image: ndarray*, *denoise_threshold: float | None = None*, *aftergauss_threshold: float | None = None*, *gauss_sigma: float | None = None*, *erosion_size: int | None = None*)

Грубо бинаризировать чёрно-белое изображение сечения свитка. Основа более продвинутых алгоритмов. В модуле config содержатся хорошие значения параметров. Если какой-то параметр указан как None, то значение берётся из модуля config.

denoise_threshold - ниже какого порога следует отсечь значения в начале, чтобы удалить шум.

gauss_sigma - параметр сигма для фильтра Гаусса, применяемого для сглаживания. Размер окна вычисляется по этой сигме.

aftergauss_threshold - по какому порогу бинаризировать изображение после применения фильтра Гаусса.

erosion_size - какого размера взять фильтр эрозии в конце.

`algorithms.smooth_from_canny`(*image_after_canny: ndarray*, *dilate1_size: int = 5*, *noise_area_threshold: int = 200*, *dilate2_size: int = 4*)

Получить сглаженную бинаризацию, используя результат работы фильтра Кэнни.

image_after_canny - 2d массив, результат применения фильтра Кэнни к исходному изображению.

dilate1_size - какого размера брать первое раздутие, до первого утоньшения.

noise_area_threshold - какая площадь считается максимальной площадью кусочков шума, которые будут удалены после первого утоньшения.

dilate2_size - какого размера брать второе раздутие, после удаления шума.

PYTHON MODULE INDEX

INDEX