Package ‘visiumStitched’

July 14, 2025

Title Enable downstream analysis of Visium capture areas stitched together with Fiji

Version 1.0.0

Date 2024-11-21

Description This package provides helper functions for working with multiple Visium capture areas that overlap each other. This package was developed along with the companion example use case data available from https://github.com/LieberInstitute/visiumStitched_brain. visiumStitched prepares SpaceRanger (10x Genomics) output files so you can stitch the images from groups of capture areas together with Fiji. Then visiumStitched builds a SpatialExperiment object with the stitched data and makes an artificial hexogonal grid enabling the seamless use of spatial clustering methods that rely on such grid to identify neighboring spots, such as PRECAST and BayesSpace. The SpatialExperiment objects created by visiumStitched are compatible with spatialLIBD, which can be used to build interactive websites for stitched SpatialExperiment objects. visiumStitched also enables casting SpatialExperiment objects as Seurat objects.

License Artistic-2.0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.4), SpatialExperiment

Suggests BiocFileCache, BiocStyle, ggplot2, knitr, RefManageR, rmarkdown, sessioninfo, Seurat, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Imports BiocBaseUtils, BiocGenerics, dplyr, DropletUtils, grDevices, imager, Matrix, methods, pkgcond, readr, rjson, S4Vectors, SingleCellExperiment, spatialLIBD (>= 1.17.8), stringr, SummarizedExperiment, tibble, tidyr, xml2

biocViews Software, Spatial, Transcriptomics, Transcription, GeneExpression, Visualization, DataImport

URL https://github.com/LieberInstitute/visiumStitched

BugReports https://support.bioconductor.org/tag/visiumStitched

git_url https://git.bioconductor.org/packages/visiumStitched

git_branch RELEASE_3_21

git_last_commit ffa8c55

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-07-13

Author Nicholas J. Eagles [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9808-5254>),
Leonardo Collado-Torres [ctb] (ORCID:
<https://orcid.org/0000-0003-2140-308X>)

Maintainer Nicholas J. Eagles <nickeagles77@gmail.com>

Contents

`.add_error_metrics` *Add error metrics related to array-coordinate mapping*

Description

Given `tibble()`s before and after mapping to new array coordinates, calculate metrics related to the suitability of the mapping.

Usage

```
.add_error_metrics(coords, coords_new, inter_spot_dist_px)
```

Arguments

`coords`	A `tibble()` containing `array_row`, `array_col`, `key`, `pxl_col_in_fullres`, `pxl_row_in_fullres`, `pxl_col_in_fullres_rounded`, `pxl_row_in_fullres_rounded`, and `capture_area` columns, representing data before mapping to new array coordinates for one `group`.
`coords_new`	A `tibble()` containing `array_row`, `array_col`, `key`, `pxl_col_in_fullres`, `pxl_row_in_fullres`, `pxl_col_in_fullres_rounded`, and `pxl_row_in_fullres_rounded` columns, representing data after mapping to new array coordinates for one `group`.
`inter_spot_dist_px`	A `numeric(1)` giving the number of pixels between spots for the `group`.

Details

Add column `shared_neighbors`, the fraction of neighbors a spot started with that are retained after mapping; add column `euclidean_error`, the number of multiples of the inter-spot distance a spot must move to be placed in the new array coordinates.

Value

A `tibble()` copy of `coords_new` with additional `shared_neighbors` and `euclidean_error` columns.

Author(s)

Nicholas J. Eagles

`.clean_round` *Round to the nearest integer, always rounding up at 0.5*

Description

This consistent behavior is favorable for our application, where we want to minimize duplicate mappings of spots to new array coordinates.

Usage

```
.clean_round(x)
```

Arguments

`x`	`numeric()` vector.

Value

A `numeric()` vector rounded to the nearest integer.

Author(s)

Nicholas J. Eagles

`.fit_to_array` *Fit spots to a new Visium-like array*

Description

Given transformed pixel coordinates, modify the ’array_row’ and ’array_col’ columns to represent a larger Visium capture area containing all capture areas in a common coordinate system. The number of array rows/cols generally changes from the Visium standards of 78 and 128 (and even may change in ratio between num rows and num cols).

Usage

```
.fit_to_array(coords, inter_spot_dist_px)
```

Arguments

`coords`	A `data.frame()` whose rows represent capture areas of the same group, and containing columns ’array_row’, ’array_col’, ’pxl_row_in_fullres’, and ’pxl_col_in_fullres’.
`inter_spot_dist_px`	`numeric(1)` vector giving the pixel distance between any 2 spots in the new coordinates.

Details

Runtime is O(n) with the number of spots, making it much faster than say, a distance-matrix-based approach running at O(n^2).

Value

A tibble with modified `array_row` + `array_col` columns, as well as new `pxl_row_in_fullres_rounded` and `pxl_col_in_fullres_rounded` columns representing the pixel coordinates rounded to the nearest exact array coordinates.

Author(s)

Nicholas J. Eagles

`.get_neighbors` *Get keys of neighboring spots*

Description

For a given row of a `tibble()` containing array coordinates, find the associated spot's neighbors (belonging to the same capture area) and return their keys.

Usage

```
.get_neighbors(i, coords)
```

Arguments

`i`	An `integer(1)` giving a row index in `coords`.
`coords`	A `tibble()` containing `array_row`, `array_col`, `key`, and `capture_area` columns.

Value

A `character()` of neighboring spot keys.

Author(s)

Nicholas J. Eagles

`.get_shared_neighbors` *Calculate fraction of neighbors retained after mapping to new array coordinates*

Description

Given `tibble()`s before and after mapping to new array coordinates, calculate for each spot the fraction of starting neighboring spots that were retained in the new array-coordinate system. Add this metric and return.

Usage

```
.get_shared_neighbors(coords_new, coords)
```

Arguments

`coords_new`	A `tibble()` containing `array_row`, `array_col`, `key`, and `capture_area` columns, representing data after mapping to new array coordinates.
`coords`	A `tibble()` containing `array_row`, `array_col`, `key`, and `capture_area` columns, representing data before mapping to new array coordinates.

Value

A `tibble()` copy of `coords_new` with additional `shared_neighbors` column.

Author(s)

Nicholas J. Eagles

`.refine_fit` *Return array coordinates fit to nearest spot with associated error*

Description

First, values of `x` are rounded to the nearest integer. Then, values of `y` are rounded to the nearest valid integer under the constraint that coordinates for x and y must be both odd or both even. These rounded values are returned, along with the Euclidean distance needed to move x and y from their original, non-integer values to their rounded values.

Usage

```
.refine_fit(x, y, INTERVAL_X, INTERVAL_Y)
```

Arguments

`x`	`numeric()` vector giving "ideal" array coordinates given every spot's transformed pixel coordinates.
`y`	Same as `x`, though `y` must represent ideal array columns iff `x` represents array rows, and vice versa.
`INTERVAL_X`	`numeric(1)` giving pixel distance between coordinate units used for `x` (e.g. if `x` represents ideal `array_col` values, `INTERVAL_X` represents pixel distance between spot columns).
`INTERVAL_Y`	`numeric(1)` giving pixel distance between coordinate units used for `y`.

Value

A `list` consisting of 3 unnamed `numeric()` vectors: rounded `x`, rounded `y`, and the Euclidean distance in pixels from rounding both `x` and `y`.

Author(s)

Nicholas J. Eagles

`.validate_array` *Check if coordinates are Visium-like*

Description

Sanity check designed to catch unforeseen bugs: halt if the tibble-like `coords`, expected to contain columns 'array_row' and 'array_col', represents an invalid Visium array.

Usage

```
.validate_array(coords)
```

Arguments

`coords`	A `data.frame()` containing `'array_row'` and `'array_col'` columns calculated internally by `add_array_coords()`.

Value

It returns `NULL` if all tests were correct.

Author(s)

Nicholas J. Eagles

`add_array_coords` *Add transformed array and pixel coordinates to a* `SpatialExperiment`

Description

Given a SpatialExperiment-class, sample information, and coordinates produced from the refinement workflow, add array and pixel coordinates appropriate for the linearly transformed capture areas making up each group present in the SpatialExperiment-class.

Usage

```
add_array_coords(spe, sample_info, coords_dir, calc_error_metrics = FALSE)
```

Arguments

`spe`	A SpatialExperiment-class object.
`sample_info`	A `data.frame()` with columns `capture_area`, `group`, `fiji_xml_path`, `fiji_image_path`, `spaceranger_dir`, `intra_group_scalar`, and `group_hires_scalef`. The last two are made by `rescale_fiji_inputs()`.
`coords_dir`	A `character(1)` vector giving the directory containing sample directories each with `tissue_positions.csv`, `scalefactors_json.json`, and `tissue_lowres_image.png` files produced from refinement with prep_fiji_coords() and related functions.
`calc_error_metrics`	A `logical(1)` vector indicating whether to calculate error metrics related to mapping spots to well-defined array coordinates. If `TRUE`, adds `euclidean_error` and `shared_neighbors` spot-level metrics to the `colData()`. The former indicates distance in number of inter-spot distances to "move" a spot to the new array position; the latter indicates the fraction of neighbors for the associated capture area that are retained after mapping, which can be quite time-consuming to compute.

Details

Array coordinates are determined via an algorithm that fits each spot to the nearest spot on a new, imaginary, Visium-like capture area. The imaginary capture area differs from a real capture area only in its extent; array coordinates still start at 0 but may extend arbitrarily beyond the normal maximum indices of 77 and 127 to fit every capture area in each group defined in the SpatialExperiment-class. The goal is to return well-defined array coordinates in a consistent spatial orientation for each group, such that downstream applications, such as clustering with `BayesSpace`, can process each group as if it really were one capture area in the first place. See `https://research.libd.org/visiumStitched/articles/visiumStitched.html#defining-array-coordinates` for more details.

Value

A SpatialExperiment-class object with additional `colData` columns `pxl_row_in_fullres_[suffix]` and `pxl_col_in_fullres_[suffix]` with `[suffix]` values `original` and `rounded`; `array_row_original` and `array_col_original` columns; and modified `colData()` columns `array_row` and `array_col` and `spatialCoords()` with their transformed values.

Author(s)

Nicholas J. Eagles

Examples

```
if (!exists("spe")) {
    spe <- spatialLIBD::fetch_data(type = "visiumStitched_brain_spe")
}

########################################################################
#   Prepare sample_info
########################################################################
```

```
sample_info <- dplyr::tibble(
    group = "Br2719",
    capture_area = c("V13B23-283_A1", "V13B23-283_C1", "V13B23-283_D1")
)
#   Add 'spaceranger_dir' column
sr_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_spaceranger"),
    exdir = sr_dir
)
sample_info$spaceranger_dir <- file.path(
    sr_dir, sample_info$capture_area, "outs", "spatial"
)

#   Add Fiji-output-related columns
fiji_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_Fiji_out"),
    exdir = fiji_dir
)
sample_info$fiji_xml_path <- temp[grep("xml$", temp)]
sample_info$fiji_image_path <- temp[grep("png$", temp)]

## Re-size images and add more information to the sample_info
sample_info <- rescale_fiji_inputs(sample_info, out_dir = tempdir())

## Preparing Fiji coordinates and images for build_SpatialExperiment()
spe_input_dir <- tempdir()
prep_fiji_coords(sample_info, out_dir = spe_input_dir)
prep_fiji_image(sample_info, out_dir = spe_input_dir)

########################################################################
#   Add array coordinates
########################################################################

spe_new <- add_array_coords(spe, sample_info, tempdir())

#    Several columns related to spatial coordinates were added
added_cols_regex <- "^(array|pxl)_(row|col)(_in_fullres)?_(original|rounded)$"
colnames(SummarizedExperiment::colData(spe_new))[
    grep(added_cols_regex, colnames(SummarizedExperiment::colData(spe_new)))
]

#    'array_row', 'array_col', and spatialCoords() were overwritten with
#    their transformed values
head(spe$array_row)
head(spe$array_col)
head(SpatialExperiment::spatialCoords(spe_new))
```

`add_overlap_info` *Add info about how spots overlap among capture areas*

Description

Given a SpatialExperiment-class and column name in its `colData`, return a modified copy of the `SpatialExperiment` with additional `colData` columns: `spe$exclude_overlapping` and `spe$overlap_key`.

Usage

```
add_overlap_info(spe, metric_name)
```

Arguments

`spe`	A SpatialExperiment-class with `colData(spe)` columns `array_row`, `array_col`, `key`, and `capture_area`.
`metric_name`	`character(1)` in `colnames(colData(spe))`, where spots belonging to the capture area with highest average value for the metric take precedence over other spots.

Details

`spe$exclude_overlapping` is `TRUE` for spots with a higher-quality overlapping capture area and `FALSE` otherwise. vis_clus onlydisplays `FALSE` spots to prevent overplotting in regions of overlap. `spe$overlap_key` gives comma-separated strings containing the keys of any overlapping spots, and is the empty string otherwise.

Value

A SpatialExperiment object with additional `colData` columns `spe$exclude_overlapping` and `spe$overlap_key`.

Author(s)

Nicholas J. Eagles

Examples

```
if (!exists("spe")) {
    spe <- spatialLIBD::fetch_data(type = "visiumStitched_brain_spe")
}

#    Find the mean of the 'sum_umi' metric by capture area to understand
#    which capture areas will be excluded in regions of overlap
SummarizedExperiment::colData(spe) |>
    dplyr::as_tibble() |>
    dplyr::group_by(capture_area) |>
    dplyr::summarize(mean_sum_umi = mean(sum_umi))

spe <- add_overlap_info(spe, "sum_umi")
```

```
#    See how many spots were excluded by capture area
table(spe$exclude_overlapping, spe$capture_area)

#    Examine how data about overlapping spots is stored (for the first
#    few spots with overlap)
head(spe$overlap_key[spe$overlap_key != ""])
```

`as.Seurat` *Convert a* `SpatialExperiment` *object to a* `Seurat` *object*

Description

Given a SpatialExperiment-class object, first `as.Seurat()` is run, which operates on SingleCellExperiment-class objects. The remaining components (images, spatial coordinates) are added manually. The actual appearance of images are buggy for now.

Usage

```
as.Seurat(
  spe,
  spatial_cols = c(tissue = "in_tissue", row = "array_row", col = "array_col", imagerow =
    "pxl_row_in_fullres", imagecol = "pxl_col_in_fullres"),
  verbose = TRUE
)
```

Arguments

`spe`	A SpatialExperiment-class with `colData()` or `spatialCoords()` columns given by `spatial_cols`. This does not have to be a stitched spe object as this function should work with any type of spe objects.
`spatial_cols`	A `character(5)` named vector mapping which `colData(spe)` or `spatialCoords(spe)` columns contain the `tissue`, `row`, `col`, `imagerow`, and `imagecol` information expected by Seurat.
`verbose`	A `logical(1)` vector. If `TRUE`, print status update about the conversion process. This information can be useful for debugging.

Details

Note that only the `lowres` images from `imgData(spe)` will be used.

Value

A `Seurat` object.

Author(s)

Nicholas J. Eagles

Examples

```
## Download some example data
spe_unstitched <- spatialLIBD::fetch_data(
    type = "spatialDLPFC_Visium_example_subset"
)[seq(100), seq(100)]

## Make the column names unique
colnames(spe_unstitched) <- spatialLIBD::add_key(spe_unstitched)$key

## Convert from a SpatialExperiment to a Seurat object
seur <- as.Seurat(spe_unstitched)
seur

## Example with an stitched SPE object
if (!exists("spe")) {
    spe <- spatialLIBD::fetch_data(type = "visiumStitched_brain_spe")
}
seur_stitched <- as.Seurat(spe[seq(100), seq(100)])

## Let's look at our resulting Seurat object
seur_stitched
```

`build_SpatialExperiment`

Build stitched `SpatialExperiment`

Description

First, read in capture-area-level SpaceRanger `https://www.10xgenomics.com/support/software/space-ranger/latest/analysis/running-pipelines/space-ranger-count` outputs. Then, overwrite spatial coordinates and images to represent group-level samples using `sample_info$group` (though keep original coordinates in `colData` columns ending with the suffix `"_original"`). Next, add info about overlaps (via `spe$exclude_overlapping` and `spe$overlap_key`). Ultimately, return a SpatialExperiment-class ready for visualization or downstream analysis.

Usage

```
build_SpatialExperiment(
  sample_info,
  coords_dir,
  count_type = "sparse",
  reference_gtf = NULL,
 gtf_cols = c("source", "type", "gene_id", "gene_version", "gene_name", "gene_type"),
  calc_error_metrics = FALSE
)
```

Arguments

`sample_info`	A `data.frame()` with columns `capture_area`, `group`, `fiji_xml_path`, `fiji_image_path`, `spaceranger_dir`, `intra_group_scalar`, and `group_hires_scalef`. The last two are made by `rescale_fiji_inputs()`.
`coords_dir`	A `character(1)` vector giving the directory containing sample directories each with `tissue_positions.csv`, `scalefactors_json.json`, and `tissue_lowres_image.png` files produced from refinement with prep_fiji_coords() and related functions.
`count_type`	A `character(1)` vector passed to `type` from `SpatialExperiment::read10xVisium`, defaulting to "sparse".
`reference_gtf`	Passed to `spatialLIBD::read10xVisiumWrapper()`. If working on the same system where SpaceRanger was run, the GTF will be automatically found; otherwise a `character(1)` path may be supplied, pointing to a GTF file of gene annotation to populate `rowData()` with.
`gtf_cols`	Passed to `spatialLIBD::read10xVisiumWrapper()`. Columns in the reference GTF to extract and populate `rowData()`.
`calc_error_metrics`	A `logical(1)` vector indicating whether to calculate error metrics related to mapping spots to well-defined array coordinates. If `TRUE`, adds `euclidean_error` and `shared_neighbors` spot-level metrics to the `colData()`. The former indicates distance in number of inter-spot distances to "move" a spot to the new array position; the latter indicates the fraction of neighbors for the associated capture area that are retained after mapping, which can be quite time-consuming to compute.

Value

A SpatialExperiment-class object with one sample per group specified in `sample_info` using transformed pixel and array coordinates (including in the `spatialCoords()`).

Author(s)

Nicholas J. Eagles

Examples

```
########################################################################
#   Prepare sample_info
########################################################################

sample_info <- dplyr::tibble(
    group = "Br2719",
    capture_area = c("V13B23-283_A1", "V13B23-283_C1", "V13B23-283_D1")
)
#   Add 'spaceranger_dir' column
sr_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_spaceranger"),
    exdir = sr_dir
```

```
)
sample_info$spaceranger_dir <- file.path(
    sr_dir, sample_info$capture_area, "outs", "spatial"
)

#   Add Fiji-output-related columns
fiji_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_Fiji_out"),
    exdir = fiji_dir
)
sample_info$fiji_xml_path <- temp[grep("xml$", temp)]
sample_info$fiji_image_path <- temp[grep("png$", temp)]

## Re-size images and add more information to the sample_info
sample_info <- rescale_fiji_inputs(sample_info, out_dir = tempdir())

## Preparing Fiji coordinates and images for build_SpatialExperiment()
spe_input_dir <- tempdir()
prep_fiji_coords(sample_info, out_dir = spe_input_dir)
prep_fiji_image(sample_info, out_dir = spe_input_dir)

########################################################################
#   Build the SpatialExperiment
########################################################################

#    Since we don't have access to the original GTF used to run SpaceRanger,
#    we must explicitly supply our own GTF to build_SpatialExperiment(). We use
#    GENCODE release 32, intended to be quite close to the actual GTF used,
#    which is available from:
#    https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-GRCh38-2024-A.tar.gz
bfc <- BiocFileCache::BiocFileCache()
gtf_cache <- BiocFileCache::bfcrpath(
    bfc,
    paste0(
        "ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/",
        "release_32/gencode.v32.annotation.gtf.gz"
    )
)

## Now we can build the stitched SpatialExperiment object
spe <- build_SpatialExperiment(
    sample_info,
    coords_dir = spe_input_dir, reference_gtf = gtf_cache
)

## Let's explore the stitched SpatialExperiment object
spe
```

`merge_overlapping` *Merge overlapping spots*

Description

Given a stitched SpatialExperiment-class, merge overlapping (same array coordinates) spots by adding expression (i.e. from `assays(spe)$counts`), returning a `SpatialExperiment` with at most one spot per array location.

Usage

```
merge_overlapping(spe)
```

Arguments

`spe`	A SpatialExperiment-class with `colData(spe)` columns `array_row`, `array_col`, `key`, `group`, and `capture_area`.

Details

`colData(spe)` and `spatialCoords(spe)` of the merged spots are taken from the spots whose `exclude_overlapping` values are `TRUE`.

Value

A SpatialExperiment with at most one spot per array location

Author(s)

Nicholas J. Eagles

Examples

```
if (!exists("spe")) {
    spe <- spatialLIBD::fetch_data(type = "visiumStitched_brain_spe")
}

#   Group colData by group and array coordinates
grouped_coldata <- colData(spe) |>
    dplyr::as_tibble() |>
    dplyr::group_by(group, array_row, array_col)

#   Find the first 100 keys that overlap other spots and don't, respectively
overlapping_keys <- grouped_coldata |>
    dplyr::filter(dplyr::n() > 1) |>
    dplyr::slice_head(n = 2) |>
    dplyr::ungroup() |>
    dplyr::slice_head(n = 100) |>
    dplyr::pull(key)
nonoverlapping_keys <- grouped_coldata |>
    dplyr::filter(dplyr::n() == 1) |>
    dplyr::ungroup() |>
    dplyr::slice_head(n = 100) |>
    dplyr::pull(key)
```

```
#   Built a small SPE containing some overlaps and some non-overlapping spots
small_spe <- spe[, c(overlapping_keys, nonoverlapping_keys)]

#   Merge overlapping spots
small_spe_merged <- merge_overlapping(small_spe)

#   All array coordinates have just one unique spot after merging
colData(small_spe_merged) |>
    dplyr::as_tibble() |>
    dplyr::group_by(group, array_row, array_col) |>
    dplyr::summarize(n = dplyr::n()) |>
    dplyr::pull(n) |>
    table()
```

`prep_fiji` *Prepare Fiji outputs for building a SpatialExperiment*

Description

Together, `prep_fiji_image()` and `prep_fiji_coords()` process Fiji outputs and generate one directory per group resembling Spaceranger's spatial outputs; in particular, `tissue_positions.csv`, `tissue_lowres_image.png`, and `scalefactors_json.json` files are created. These functions are necessary to run in preparation for `build_SpatialExperiment()`.

Usage

```
prep_fiji_image(sample_info, out_dir, lowres_max_size = 1200)

prep_fiji_coords(sample_info, out_dir)
```

Arguments

`sample_info`	A `data.frame()` with columns `capture_area`, `group`, `fiji_xml_path`, `fiji_image_path`, `spaceranger_dir`, `intra_group_scalar`, and `group_hires_scalef`. The last two are made by `rescale_fiji_inputs()`.
`out_dir`	A `character(1)` vector giving a path to a directory to place the output pixel coordinates CSVs. It must exist in advance.
`lowres_max_size`	An `integer(1)` vector: the resolution (number of pixels) of the larger dimension of the output image(s), considered to be "low resolution". The default value of `1200` assumes that you are stitching together at most a 2 by 2 grid of Visium capture areas, where each has at most 600 pixels on the longest dimension (as is the default in SpaceRanger).

Details

Given a `data.frame()` of sample information (`sample_info`) with columns `capture_area`, `group`, and `fiji_xml_path`, expected to have one unique path to Fiji XML output per group, `prep_fiji_coords` reads in the pixel coordinates from each capture area's `tissue_positions.csv` file from SpaceRanger, and transform using the rotation matrix specified by Fiji `https://imagej.net/software/fiji/`. It writes one new `tissue_positions.csv` file per group.

After stitching all groups in `sample_info` with Fiji, images of various resolutions (pixel dimensions) are left. `prep_fiji_image()` creates copies of each image whose largest dimension is `lowres_max_size` pixels. It also creates a corresponding `scalefactors_json.json` file much like SpaceRanger's.

Value

This function returns a `character()` with the file paths to the files it created. For `prep_fiji_coords()`, these are the `tissue_positions.csv` files; for `prep_fiji_image()`, these are the `tissue_lowres_image.png` and `scalefactors_json.json` files.

Functions

- `prep_fiji_image()`: Create low-res images and scale factors from high-res Fiji output images
- `prep_fiji_coords()`: Apply transform info from Fiji XML output

Author(s)

Nicholas J. Eagles

Examples

```
sample_info <- dplyr::tibble(
    group = "Br2719",
    capture_area = c("V13B23-283_A1", "V13B23-283_C1", "V13B23-283_D1")
)
#   Add 'spaceranger_dir' column
sr_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_spaceranger"),
    exdir = sr_dir
)
sample_info$spaceranger_dir <- file.path(
    sr_dir, sample_info$capture_area, "outs", "spatial"
)

#   Add Fiji-output-related columns
fiji_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_Fiji_out"),
    exdir = fiji_dir
)
sample_info$fiji_xml_path <- temp[grep("xml$", temp)]
```

```
sample_info$fiji_image_path <- temp[grep("png$", temp)]

## Re-size images and add more information to the sample_info
sample_info <- rescale_fiji_inputs(sample_info, out_dir = tempdir())

spe_input_dir <- tempdir()
out_paths_image <- prep_fiji_image(
    sample_info,
    out_dir = spe_input_dir, lowres_max_size = 1000
)
out_path_coords <- prep_fiji_coords(sample_info, out_dir = spe_input_dir)

#    A "low resolution" stitched image was produced, which has 1000
#    pixels in its largest dimension
this_image <- imager::load.image(
    file.path(spe_input_dir, "Br2719", "tissue_lowres_image.png")
)
dim(this_image)
library("imager")
plot(this_image)

#    'prep_fiji_image' produced an image and scalefactors
out_paths_image

#    'prep_fiji_coords' produced a file of spatial coordinates for the
#    stitched Br2719
readr::read_csv(out_path_coords)
```

`rescale_fiji_inputs` *Write same-scale hires images for input to Fiji*

Description

Given a `data.frame()` of sample information (`sample_info`) with columns `capture_area`, `group`, and `spaceranger_dir`, Write new high-resolution images for use as input to Fiji `https://imagej.net/software/fiji/`. Particularly when capture areas come from different slides, there is a risk of significant scale differences among SpaceRanger's `tissue_hires_image.png` images; that is, the physical distance represented by a pixel from each capture area may differ nontrivially, leading to a distance-distorted output image, and inconsistent scaling when later transforming pixel coordinates. This function writes approximately high-res images whose pixels are of equal physical size within each group, then adds `intra_group_scalar` and `group_hires_scalef` columns to `sample_info`. `intra_group_scalar` gives the scalar by a which a given capture area's `tissue_hires_image.png` image and pixel coordinates must be multiplied to match the scale of other `group` members; `group_hires_scalef` gives the new `tissue_hires_scalef` (as from SpaceRanger's `scalefactors_json.json` file) appropriate for every capture area from the group.

Usage

```
rescale_fiji_inputs(sample_info, out_dir)
```

Arguments

`sample_info`	A `data.frame()` with columns `capture_area`, `group`, `fiji_xml_path`, `fiji_image_path`, `spaceranger_dir`, `intra_group_scalar`, and `group_hires_scalef`. The last two are made by `rescale_fiji_inputs()`.
`out_dir`	A `character(1)` vector giving a path to a directory to place the output images, which must exist in advance.

Value

A tibble: a copy of `sample_info` with additional columns `intra_group_scalar` and `group_hires_scalef`.

Author(s)

Nicholas J. Eagles

Examples

```
#    Define sample information for the example human brain data
sample_info <- dplyr::tibble(
    group = "Br2719",
    capture_area = c("V13B23-283_A1", "V13B23-283_C1", "V13B23-283_D1")
)
#   Add 'spaceranger_dir' column
sr_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_spaceranger"),
    exdir = sr_dir
)
sample_info$spaceranger_dir <- file.path(
    sr_dir, sample_info$capture_area, "outs", "spatial"
)

#   Add Fiji-output-related columns
fiji_dir <- tempdir()
temp <- unzip(
    spatialLIBD::fetch_data("visiumStitched_brain_Fiji_out"),
    exdir = fiji_dir
)
sample_info$fiji_xml_path <- temp[grep("xml$", temp)]
sample_info$fiji_image_path <- temp[grep("png$", temp)]

## Re-size images and add more information to the sample_info
out_dir <- tempdir()
sample_info_new <- rescale_fiji_inputs(sample_info, out_dir = out_dir)

#    Scale factors are computed that are necessary downstream (i.e. with
#    prep_fiji_*() functions)
sample_info_new[, setdiff(colnames(sample_info_new), colnames(sample_info))]

#    Image are produced that are ready for alignment in Fiji
list.files(out_dir)
```

Index