HoughLines

May 22, 2021

[1]:

```
from ipywidgets import interactive
import ipywidgets as widgets
from skimage import data, io, filters
from skimage.filters import threshold_otsu
from skimage.measure import label, regionprops
from skimage import morphology, draw, filters
import matplotlib.pyplot as plt
import os
import numpy as np
from skimage.transform import hough_line, hough_line_peaks
from scipy.signal import find_peaks_cwt, correlate, find_peaks
from scipy.ndimage import convolve1d
import scipy.signal
from matplotlib.figure import Figure
import scipy.signal
from scipy.ndimage import gaussian_filter1d
import glob
import os
from skimage import data, io, filters, transform
import itertools
%matplotlib widget
```

1 Segmenter

[2]:

```
class PiecewiseHoughSegmenter:
    def __init__(self, fragment, num_strips, rescale = 0.5):
        """
        max_slope: should be greater than zero.
        """
        self.fragment = fragment
        fragment_binary = self.fragment.load(rescale = rescale)
        self.ink = fragment_binary < threshold_otsu(fragment_binary)
        self.width = self.ink.shape[1]
        self.height = self.ink.shape[0]
        self.set_strips(num_strips)
    def set_strips(self, num_strips):
        self.strips = []
        widths = [self.width // num_strips +
                  (1 if x < self.width % num_strips else 0)
                  for x in range(num_strips)]
        horizontal_px = np.arange(self.width)
        strip_px = np.array_split(horizontal_px, num_strips)
        start = 0
        for width in widths:
            self.strips.append(LineStrip(self.ink, start, start + width))
            start += width
    def estimate_pitch(self, min_pitch = 50):
        autocorr = sum(strip.autocorr() for strip in self.strips)
        peaks,_ = find_peaks(autocorr)
        pitch_estimate = np.min(peaks[peaks > min_pitch])
        return pitch_estimate
    def hough(self, angle_resolution = 1001, max_angle = 0.5, conv_window =␣
↪None):
        delta_th = max_angle * np.pi
        hough_angles = np.linspace(np.pi / 2 - delta_th, np.pi / 2 + delta_th,␣
↪angle_resolution)
        for strip in self.strips:
            strip.set_hough(hough_angles, conv_window = conv_window)
    def plot_fragment(self, ax):
        """ Plot the fragmenter and lines
        """
        ax.cla()
        ax.imshow(~self.ink, cmap='gray')
        for strip in self.strips:
            ax.axvline(strip.end)
        ax.set_xlim(0, self.width)
        for strip in self.strips:
            for line in strip.lines:
                line.plot(ax, strip.start, strip.end)
    def plot_hough(self, ax, strip, aggregate = True, conv = False, kind =␣
↪'counts'):
        """ Plot the Hough space
        """
        ax.cla()
        if aggregate:
            hough = np.sum([s.hough[kind] for s in self.strips], axis=0)
            for s in self.strips:
                for l in s.lines:
                    l.plot_hough(ax)
        else:
            s = self.strips[strip]
            hough = s.hough[kind]
            for l in s.lines:
                l.plot_hough(ax)
        # Calculate bounds
        angle_step = 0.5 * np.diff(self.strips[0].theta).mean()
        d_step = 0.5 * np.diff(self.strips[0].d).mean()
        bounds = [np.rad2deg(self.strips[0].theta[0] - angle_step),
                  np.rad2deg(self.strips[0].theta[-1] + angle_step),
                  self.strips[0].d[-1] + d_step, self.strips[0].d[0] + d_step]
        ax.imshow(hough, cmap='gray', aspect='auto', vmin=0, vmax=hough.max(),␣
↪extent=bounds)
    def fit_segments(self, rel_threshold):
        global_threshold = rel_threshold * np.max([s.hough['conv'] for s in␣
↪self.strips])
        for s in self.strips:
            s.fit_hough_peaks(threshold = rel_threshold, global_threshold =␣
↪global_threshold)
```

[3]:

```
class LineStrip:
    """
    Vertical strip of a fragment with detected line segments
    """
    def __init__(self, ink, start, end):
        """
        """
        self.start = start
        self.end = end
        self.mid = (end - start) / 2
        self.ink = ink
        self.ink_strip = np.copy(ink)
        self.ink_strip[:,:start] = False
        self.ink_strip[:,end:] = False
        self.center_profile = np.sum(self.ink_strip, axis=1)
        self.lines = []
        self.hough = dict()
    def set_hough(self, angles, conv_window = None):
        # Compute Hough transform for the slice
        self.hough['counts'], self.theta, self.d = hough_line(self.ink_strip,␣
↪theta=angles)
        if conv_window is None:
            self.hough['conv'] = self.hough['counts']
        else:
            self.hough['conv'] =  convolve1d(
                self.hough['counts'].astype(np.int16),
                conv_window, mode='constant',
                cval=0, axis=0)
    def autocorr(self):
        autocorr = correlate(self.center_profile, self.center_profile)
        # Cut autocorrelation in half
        autocorr = autocorr[len(autocorr) // 2:]
        return autocorr
    def fit_hough_peaks(self, threshold = 0.5, angle_range = 0.1 * np.pi,␣
↪min_distance = 10, global_threshold = 1.0):
        self.lines = []
        angle_bounds = (np.pi / 2 - angle_range, np.pi / 2 + angle_range)
        min_angle = 90
        peaks = hough_line_peaks(self.hough['conv'], self.theta, self.d,␣
↪min_angle = min_angle, min_distance = int(min_distance))
        for height, angle, dist in zip(*peaks):
            if height > global_threshold:
                if (angle > angle_bounds[0]) and (angle < angle_bounds[1]):
                    self.lines.append(HoughLine(angle, dist))
```

[4]:

```
class HoughLine:
    def __init__(self, theta, distance):
        self.theta = theta
        self.distance = distance
        self.show = True
    def intersect(self, x):
        '''
        Get intersection with a horizontal line
        '''
        return self.distance / np.sin(self.theta) - x / np.tan(self.theta)
    def from_intercept(self, x, y, theta):
        # Intercept point
        P = np.array([x, y])
        # Line basis vector
        l = np.array([np.cos(theta), np.sin(theta)])
        # Project (origin,P) onto the line
        projection = np.dot(P, l)
        self.distance = np.dot(P, l)
        self.theta = theta
    def plot(self, ax, start = 0, end=20, ls="-"):
        if self.show:
            ax.plot([start, end], [self.intersect(start), self.intersect(end)],␣
↪ls=ls, c='r', lw=1)
        else:
            mid = end + start / 2.
            ax.plot(mid, self.intersect(mid), 'ro')
    def plot_hough(self, ax):
        ax.plot(np.rad2deg(self.theta), self.distance, 'ro')
```

[5]:

```
class FragmentLoader:
    def __init__(self, base_path, name):
        self.base_path = base_path
        self.name = name
    def load(self, suffix = '-binarized.jpg', rescale = None, binarize = True):
        filename = os.path.join(self.base_path, self.name + suffix)
        image = io.imread(filename)
        if not rescale is None:
            image = transform.rescale(image, rescale, anti_aliasing=False)
        return image
```

[6]:

```
def get_fragments(path = "../image-data", suffix="-binarized.jpg"):
    file_pattern = os.path.join(path, "*" + suffix)
    files = glob.glob(file_pattern)
    fragments = []
    for file in files:
        name = os.path.basename(file)
        name = name[:-len(suffix)]
        fragments.append(FragmentLoader(path, name))
    return fragments
```

[7]:

```
class InteractiveFragments(widgets.Tab):
    """ Encapsulate a set of fragments and show interactive controls for␣
 ↪processing it
    and plotting intermediate outputs
    """
    def __init__(self, fragments):
        super().__init__()
        self.fragments = fragments
        self.fragment = fragments[0]

        # Fragments tab
        self.fragments_tab = FragmentsTab(fragments)
        self.fragments_tab.fragment_selector.observe(self.set_fragment, 'value')

        # Segmenter tab
        self.segmenter_tab = SegmenterTab("a")
        self.segmenter_tab.num_strips.observe(self.update_segmenter, 'value')

        # Hough tab
        self.hough_tab = HoughTab()
        self.hough_tab.selected_strip.observe(self.update_hough, 'value')
        self.hough_tab.hough_angles.observe(self.update_hough, 'value')
        self.hough_tab.aggregate_strips.observe(self.update_hough, 'value')
        self.hough_tab.kind.observe(self.update_hough, 'value')
        self.hough_tab.line_height.observe(self.update_hough, 'value')
        self.hough_tab.taper.observe(self.update_hough, 'value')
        self.hough_tab.threshold.observe(self.fit_segments, 'value')
        self.children = [self.fragments_tab, self.segmenter_tab, self.hough_tab]

        self.set_title(0, "Fragment")
        self.set_title(1, "Line segmentation")
        self.set_title(2, "Hough")
        self.set_fragment(None)
    def set_fragment(self, change):
        self.fragment = self.fragments[self.fragments_tab.fragment_selector.
↪value]
        self.update_segmenter(None)
        self.segmenter_tab.label.value = self.fragment.name
    def update_segmenter(self, change):
        num_strips = self.segmenter_tab.num_strips.value
        self.segmenter = PiecewiseHoughSegmenter(self.fragment, num_strips)
        self.hough_tab.selected_strip.max = num_strips - 1
        self.segmenter.plot_fragment(self.segmenter_tab.ax)
        self.update_hough(None)
    def update_hough(self, change):
        max_angle = self.hough_tab.hough_angles.value
        line_height = self.hough_tab.line_height.value
        window = conv_window(
            self.hough_tab.line_height.value,
            self.hough_tab.taper.value
        )
        self.segmenter.hough(max_angle = max_angle, conv_window = window)
        self.update_hough_plot()
    def update_hough_plot(self):
        aggregate = self.hough_tab.aggregate_strips.value
        strip = self.hough_tab.selected_strip.value
        kind = self.hough_tab.kind.value
        self.segmenter.plot_hough(self.hough_tab.ax, strip,␣
↪aggregate=aggregate, kind=kind)
    def fit_segments(self, change):
        threshold = self.hough_tab.threshold.value
        self.segmenter.fit_segments(threshold)
        self.segmenter.plot_fragment(self.segmenter_tab.ax)
        self.update_hough_plot()
```

[8]:

```
class FragmentsTab(widgets.VBox):
    def __init__(self, fragments):
        super().__init__()
        self.fragments = fragments
        self.loader = fragments[0]
        self.version = 'binarized'
        # Controls
        options = [(f.name, n) for n, f in enumerate(fragments)]
        self.fragment_selector = widgets.Dropdown(options=options)

        display_versions = [('Binarized', 'binarized'),
                            ('Color', 'color'),
                            ("Fused", 'fused')]
        version_selector = widgets.Dropdown(options=display_versions)
        # Plot
        plot_output = widgets.Output()
        with plot_output:
            self.fig, self.ax = plt.subplots(constrained_layout = True)
            self.fig.canvas.header_visible = False
        self.children = [self.fragment_selector, version_selector, plot_output]
        self.plot()

        # Linking controls
        self.fragment_selector.observe(self.select_fragment, 'value')
        version_selector.observe(self.set_version, 'value')
    def plot(self):
        suffix = {
            'binarized'  : '-binarized.jpg',
            'color' : '.jpg',
            'fused' : '-fused.jpg'
        }
        im = self.loader.load(suffix = suffix[self.version])
        self.ax.cla()
        self.ax.imshow(im, cmap='gray')
    def select_fragment(self, change):
        self.loader = self.fragments[change.new]
        self.plot()
    def set_version(self, change):
        self.version = change.new
        self.plot()
```

[9]:
```
class SegmenterTab(widgets.VBox):
    def __init__(self, fragment):
        super().__init__()
        self.num_strips = widgets.IntSlider(min=1, max=15, value=5,␣
 ↪description="Strips:")
        self.label = widgets.Label("A")

        self.controls = widgets.VBox([self.label, self.num_strips])

        self.plot_output = widgets.Output()
        with self.plot_output:
            self.fig, self.ax = plt.subplots(constrained_layout = True)
            self.fig.canvas.header_visible = False
        self.children = [self.controls, self.plot_output]
```

[10]:
```
def conv_window(line_height, slope):
    window = -np.abs(np.linspace(-slope, slope, line_height * 2 + 1))
    window += slope / 2
    window = window.clip(-1, 1)
    return window
```

[11]:
```
class HoughTab(widgets.VBox):
    """ Plot Hough-space
    """
    def __init__(self, strips = 1):
        super().__init__()

        self.plot_output = widgets.Output()
        with self.plot_output:
            self.fig, self.ax = plt.subplots(constrained_layout = True)
            self.fig.canvas.header_visible = False
        self.selected_strip = widgets.IntSlider(min = 0, max = strips,␣
↪description = "Shown strip:")
        self.aggregate_strips = widgets.Checkbox(value=True,␣
↪description="Aggregate strips")
        self.hough_angles = widgets.FloatSlider(min=0., max = 0.5, step=0.025,␣
↪value = 0.3, description = "Max slope / pi: ")
        self.kind = widgets.RadioButtons(
            options=['counts', 'conv', 'adjusted'],
            description="Plot type:"
        )
        self.line_height = widgets.IntSlider(min=1, max=100, value=25,␣
↪description="Line height")
        self.taper = widgets.IntSlider(min=1, max=100, value=25,␣
↪description="Window taper")
        self.threshold = widgets.FloatLogSlider(min = -4, max = 0, step = 0.
↪025, base=10, description="Peak threshold", value = 0.9)
        self.children = [
            widgets.HBox([
                widgets.VBox([self.selected_strip, self.aggregate_strips, self.
↪hough_angles]),
                widgets.VBox([self.kind]),
                widgets.VBox([self.line_height, self.taper, self.threshold]),
            ]),
            self.plot_output
        ]
```

[12]:
```
InteractiveFragments(get_fragments())
```

```
InteractiveFragments(children=(FragmentsTab(children=(Dropdown(options=(('P166-Fg002-R-C01-R01
↪0), ('P423-1-…
```

[ ]:
```

```