

## scikit-image: Image processing in Python

scikit-image is an image processing library that implements algorithms and utilities for use in research, education and industry applications. It is released under the liberal "Modified BSD" open source license, provides a well-documented API in the Python programming language, and is developed by an active, international team of collaborators. In this paper we highlight the advantages of open source to achieve the goals of the scikit-image library, and we showcase several real-world image processing applications that use scikit-image.

# scikit-image: Image processing in Python

Stéfan van der Walt<sup>1,2</sup>, Johannes L. Schönberger<sup>3</sup>, Juan Nunez-Iglesias<sup>4</sup>,  
François Boulogne<sup>5</sup>, Joshua D. Warner<sup>6</sup>, Neil Yager<sup>7</sup>, Emmanuelle  
Gouillart<sup>8</sup>, Tony Yu<sup>9</sup>, and the scikit-image contributors<sup>10</sup>

<sup>1</sup>Corresponding author: [stefan@sun.ac.za](mailto:stefan@sun.ac.za)

<sup>2</sup>Stellenbosch University, Stellenbosch, South Africa

<sup>3</sup>Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599, USA

<sup>4</sup>Victorian Life Sciences Computation Initiative, Carlton, VIC, 3010, Australia

<sup>5</sup>Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, New Jersey 08544, USA

<sup>6</sup>Department of Biomedical Engineering, Mayo Clinic, Rochester, Minnesota 55905, USA

<sup>7</sup>AICBT Ltd, Oxford, UK

<sup>8</sup>Joint Unit CNRS / Saint-Gobain, Cavaillon, France

<sup>9</sup>Enthought Inc., Austin, TX, USA

<sup>10</sup><https://github.com/scikit-image/scikit-image/graphs/contributors>

## ABSTRACT

scikit-image is an image processing library that implements algorithms and utilities for use in research, education and industry applications. It is released under the liberal “Modified BSD” open source license, provides a well-documented API in the Python programming language, and is developed by an active, international team of collaborators. In this paper we highlight the advantages of open source to achieve the goals of the scikit-image library, and we showcase several real-world image processing applications that use scikit-image.

Keywords: image processing, reproducible research, education, visualization

## INTRODUCTION

In our data-rich world, images represent a significant subset of all measurements made. Examples include DNA microarrays, microscopy slides, astronomical observations, satellite maps, robotic vision capture, synthetic aperture radar images, and higher-dimensional images such as 3-D magnetic resonance or computed tomography imaging. Exploring these rich data sources requires sophisticated software tools that should be easy to use, free of charge and restrictions, and able to address all the challenges posed by such a diverse field of analysis.

This paper describes scikit-image, a collection of image processing algorithms implemented in the Python programming language by an active community of volunteers and available under the liberal BSD Open Source license. The rising popularity of Python as a scientific programming language, together with the increasing availability of a large eco-system of complementary tools, make it an ideal environment in which to produce an image processing toolkit.

The project aims are:

1. *To provide high quality, well-documented and easy-to-use implementations of*

36 *common image processing algorithms.*

37 Such algorithms are essential building blocks in many areas of scientific research,  
38 algorithmic comparisons and data exploration. In the context of reproducible  
39 science, it is important to be able to inspect any source code used for algorithmic  
40 flaws or mistakes. Additionally, scientific research often requires custom  
41 modification of standard algorithms, further emphasizing the importance of open  
42 source.

43 2. *To facilitate education in image processing.*

44 The library allows students in image processing to learn algorithms in a hands-on  
45 fashion by adjusting parameters and modifying code. In addition, a `novice`  
46 module is provided, not only for teaching programming in the “turtle graphics”  
47 paradigm, but also to familiarize users with image concepts such as color and  
48 dimensionality. Furthermore, the project takes part in the yearly Google Summer  
49 of Code program (Google, 2004), where students learn about image processing  
50 and software engineering through contributing to the project.

51 3. *To address industry challenges.*

52 High quality reference implementations of trusted algorithms provide industry  
53 with a reliable way of attacking problems, without having to expend significant  
54 energy in re-implementing algorithms already available in commercial packages.  
55 Companies may use the library entirely free of charge, and have the option of  
56 contributing changes back, should they so wish.

## 57 GETTING STARTED

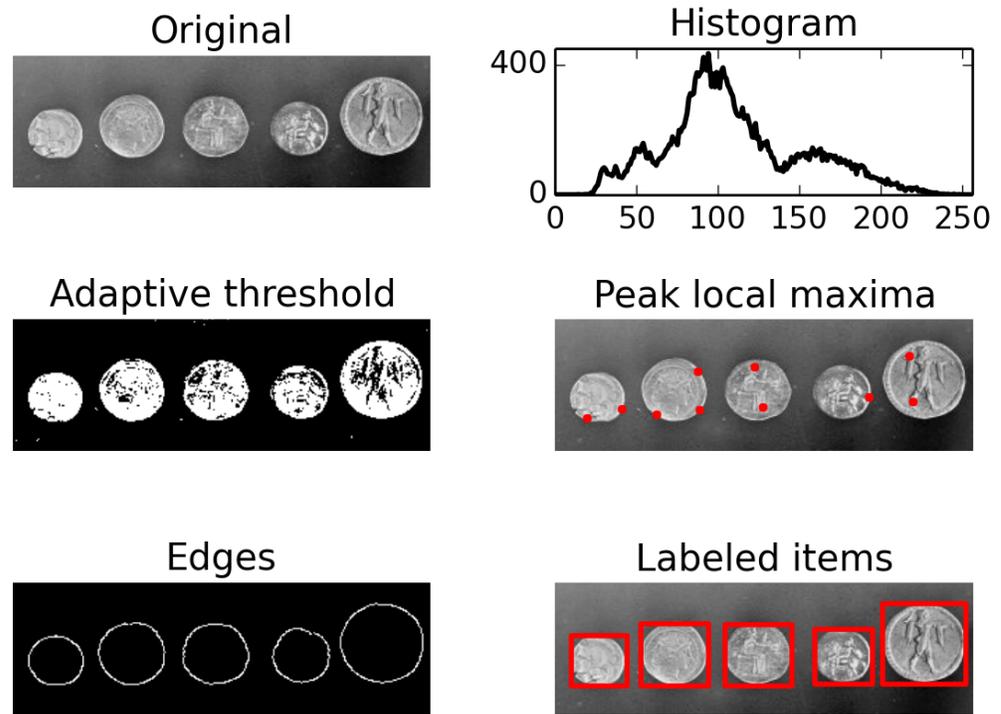
58 One of the main goals of scikit-image is to make it easy for any user to get started  
59 quickly—especially users already familiar with Python’s scientific tools. To that end, the  
60 basic image is just a standard NumPy array, which exposes pixel data directly to the  
61 user. A new user can simply load an image from disk (or use one of scikit-image’s  
62 sample images), process that image with one or more image filters, and quickly display  
63 the results:

```
from skimage import data, io, filter

image = data.coins() # or any NumPy array!
edges = filter.sobel(image)
io.imshow(edges)
```

64 The above demonstration loads `data.coins`, an example image shipped with  
65 scikit-image. For a more complete example, we import NumPy for array manipulation  
66 and matplotlib for plotting. At each step, we add the picture or the plot to a matplotlib  
67 figure shown in Figure 1.

```
import numpy as np
import matplotlib.pyplot as plt
```



**Figure 1.** Illustration of several functions available in scikit-image: adaptive threshold, local maxima, edge detection and labels. The use of NumPy arrays as our data container also enables the use of NumPy's built-in `histogram` function.

```
# Load a small section of the image.
image = data.coins()[0:95, 70:370]

fig, axes = plt.subplots(ncols=2, nrows=3,
                          figsize=(8, 4))
ax0, ax1, ax2, ax3, ax4, ax5 = axes.flat
ax0.imshow(image, cmap=plt.cm.gray)
ax0.set_title('Original', fontsize=24)
ax0.axis('off')
```

68        Since the image is represented by a NumPy array, we can easily perform operations  
69        such as building an histogram of the intensity values.

```
# Histogram.
values, bins = np.histogram(image,
                             bins=np.arange(256))

ax1.plot(bins[:-1], values, lw=2, c='k')
ax1.set_xlim(xmax=256)
ax1.set_yticks([0, 400])
ax1.set_aspect(.2)
ax1.set_title('Histogram', fontsize=24)
```

70        To divide the foreground and background, we threshold the image to produce a binary  
71        image. Several threshold algorithms are available. Here, we employ  
72        `filter.threshold_adaptive` where the threshold value is the weighted mean  
73        for the local neighborhood of a pixel.

```
# Apply threshold.
from skimage.filter import threshold_adaptive

bw = threshold_adaptive(image, 95, offset=-15)

ax2.imshow(bw, cmap=plt.cm.gray)
ax2.set_title('Adaptive threshold', fontsize=24)
ax2.axis('off')
```

74        We can easily detect interesting features, such as local maxima and edges. The  
75        function `feature.peak_local_max` can be used to return the coordinates of local  
76        maxima in an image.

```
# Find maxima.
from skimage.feature import peak_local_max

coordinates = peak_local_max(image, min_distance=20)
```

```
ax3.imshow(image, cmap=plt.cm.gray)
ax3.autoscale(False)
ax3.plot(coordinates[:, 1],
          coordinates[:, 0], c='r.')
ax3.set_title('Peak local maxima', fontsize=24)
ax3.axis('off')
```

77 Next, a Canny filter (`filter.canny`) (Canny, 1986) detects the edge of each  
78 coin.

```
# Detect edges.
from skimage import filter

edges = filter.canny(image, sigma=3,
                    low_threshold=10,
                    high_threshold=80)

ax4.imshow(edges, cmap=plt.cm.gray)
ax4.set_title('Edges', fontsize=24)
ax4.axis('off')
```

79 Then, we attribute to each coin a label (`morphology.label`) that can be used to  
80 extract a sub-picture. Finally, physical information such as the position, area, eccentricity,  
81 perimeter, and moments can be extracted using `measure.regionprops`.

```
# Label image regions.
from skimage.measure import regionprops
import matplotlib.patches as mpatches
from skimage.morphology import label

label_image = label(edges)

ax5.imshow(image, cmap=plt.cm.gray)
ax5.set_title('Labeled items', fontsize=24)
ax5.axis('off')

for region in regionprops(label_image):
    # Draw rectangle around segmented coins.
    minr, minc, maxr, maxc = region.bbox
    rect = mpatches.Rectangle((minc, minr),
                              maxc - minc,
                              maxr - minr,
                              fill=False,
                              edgecolor='red',
                              linewidth=2)

    ax5.add_patch(rect)
```

```
plt.tight_layout()
plt.show()
```

82 scikit-image thus makes it possible to perform sophisticated image processing tasks  
83 with only a few function calls.

## 84 LIBRARY CONTENTS

85 The scikit-image project started in August of 2009 and has received contributions  
86 from more than 100 individuals (Ohloh, 2014). The package can be installed from,  
87 amongst other sources, the Python Package Index, Continuum Anaconda (Continuum  
88 Analytics, 2012), Enthought Canopy (Enthought, Inc, 2014), Python(x,y) (Raybaut,  
89 2014), NeuroDebian (Halchenko and Hanke, 2012) and GNU/Linux distributions such  
90 as Ubuntu (Canonical, Ltd., 2004). In March 2014 alone, the package was downloaded  
91 more than 5000 times from the Python Package Index (PyPI, 2014).

92 The package currently contains the following sub-modules:

- 93 • color: Color space conversion.
- 94 • data: Test images and example data.
- 95 • draw: Drawing primitives (lines, text, etc.) that operate on NumPy arrays.
- 96 • exposure: Image intensity adjustment, e.g., histogram equalization, etc.
- 97 • feature: Feature detection and extraction, e.g., texture analysis, corners, etc.
- 98 • filter: Sharpening, edge finding, rank filters, thresholding, etc.
- 99 • graph: Graph-theoretic operations, e.g., shortest paths.
- 100 • io: Reading, saving, and displaying images and video.
- 101 • measure: Measurement of image properties, e.g., similarity and contours.
- 102 • morphology: Morphological operations, e.g., opening or skeletonization.
- 103 • novice: Simplified interface for teaching purposes.
- 104 • restoration: Restoration algorithms, e.g., deconvolution algorithms, denoising,  
105 etc.
- 106 • segmentation: Partitioning an image into multiple regions.
- 107 • transform: Geometric and other transforms, e.g., rotation or the Radon transform.
- 108 • viewer: A simple graphical user interface for visualizing results and exploring  
109 parameters.

## 110 DATA FORMAT AND PIPELINING

111 scikit-image represents images as NumPy arrays (van der Walt et al., 2011), the de facto  
112 standard for storage of multi-dimensional data in scientific Python. Each array has a  
113 dimensionality, such as 2 for a 2-D grayscale image, 3 for a 2-D multi-channel image,  
114 or 4 for a 3-D multi-channel image; a shape, such as  $(M, N, 3)$  for an RGB color image  
115 with  $M$  vertical and  $N$  horizontal pixels; and a numeric data type, such as `float` for  
116 continuous-valued pixels and `uint8` for 8-bit pixels. Our use of NumPy arrays as the  
117 fundamental data structure maximizes compatibility with the rest of the scientific Python  
118 ecosystem. Data can be passed as-is to other tools such as NumPy, SciPy, matplotlib,  
119 scikit-learn (Pedregosa et al., 2011), Mahotas (Coelho, 2013), OpenCV, and more.

120 Images of differing data-types can complicate the construction of pipelines. scikit-  
121 image follows an "Anything In, Anything Out" approach, whereby all functions are  
122 expected to allow input of an arbitrary data-type but, for efficiency, also get to choose  
123 their own output format. Data-type ranges are clearly defined. Floating point images are  
124 expected to have values between 0 and 1 (unsigned images) or -1 and 1 (signed images),  
125 while 8-bit images are expected to have values in  $\{0, 1, 2, \dots, 255\}$ . We provide utility  
126 functions, such as `img_as_float`, to easily convert between data-types.

## 127 DEVELOPMENT PRACTICES

128 The purpose of scikit-image is to provide a high-quality library of powerful, diverse im-  
129 age processing tools free of charge and restrictions. These principles are the foundation  
130 for the development practices in the scikit-image community.

131 The library is licensed under the *Modified BSD license*, which allows unrestricted  
132 redistribution for any purpose as long as copyright notices and disclaimers of warranty  
133 are maintained (Regents of the University of California, 1999). It is compatible with  
134 GPL licenses, so users of scikit-image can choose to make their code available under  
135 the GPL. However, unlike the GPL, it does not require users to open-source derivative  
136 work (BSD is not a so-called copyleft license). Thus, scikit-image can also be used in  
137 closed-source, commercial environments.

138 The development team of scikit-image is an open community that collaborates on  
139 the *GitHub* (the scikit-image team, 2010a) platform for issue tracking, code review, and  
140 release management. *Google Groups* (the scikit-image team, 2010b) is used as a public  
141 discussion forum for user support, community development, and announcements.

142 scikit-image complies with the PEP8 coding style standard (van Rossum et al., 2001)  
143 and the NumPy documentation format (Gommers and the NumPy developers, 2010)  
144 in order to provide a consistent, familiar user experience across the library similar to  
145 other scientific Python packages. As mentioned earlier, the data representation used  
146 is  $n$ -dimensional NumPy arrays, which guarantees universal interoperability within  
147 the scientific Python ecosystem. The majority of the scikit-image API is intentionally  
148 designed as a functional interface which allows one to simply apply one function to  
149 the output of another. This modular approach also lowers the barrier of entry for new  
150 contributors, since one only needs to master a small part of the entire library in order to  
151 make an addition.

152 We ensure high code quality by a thorough review process using the pull request  
153 interface on GitHub. The source code is mainly written in Python, although certain

154 performance critical sections are implemented in Cython, an optimising static compiler  
155 for Python (Behnel et al., 2011). scikit-image aims to achieve full unit test coverage,  
156 which is above 85% as of release 0.10 and continues to rise. A continuous integration  
157 system (the Travis-CI community, 2012; LEMUR Heavy Industries, 2013) automatically  
158 checks each commit for unit test coverage and failures on both Python 2 and Python 3.  
159 Additionally, the code is analyzed by flake8 (Cordasco, 2010) to ensure compliance with  
160 the PEP8 coding style standards (van Rossum et al., 2001). Finally, the properties of  
161 each public function are documented thoroughly in an API reference guide, embedded as  
162 Python docstrings and accessible through the official project homepage or an interactive  
163 Python console. Short usage examples are typically included inside the docstrings,  
164 and new features are accompanied by longer, self-contained example scripts added  
165 to the narrative documentation and compiled to a gallery on the project website. We  
166 use Sphinx (Brandl and the Sphinx team, 2007) to automatically generate both library  
167 documentation and the website.

168 The development master branch is fully functional at all times and can be obtained  
169 from GitHub (the scikit-image developers, 2010). The community releases major  
170 updates as stable versions approximately every six months (Wikipedia, 2014). Major  
171 releases include new features, while minor releases typically contain only bug fixes.  
172 Users are notified about API-breaking changes by deprecation warnings one full major  
173 release before they are applied.

## 174 USAGE EXAMPLES

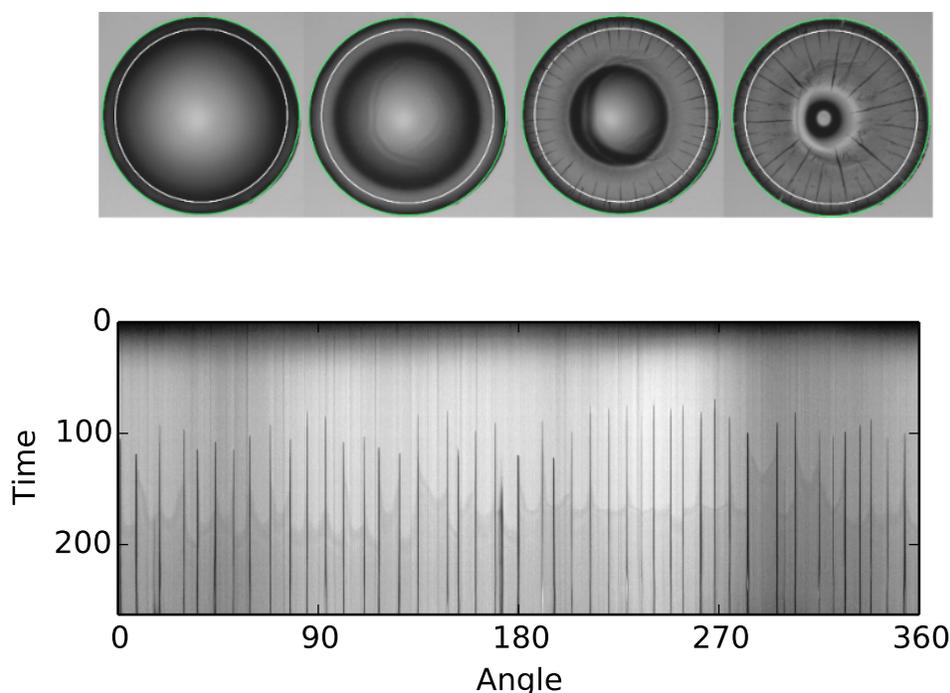
### 175 Research

176 Often, a disproportionately large component of research involves dealing with various  
177 image data-types, color representations, and file format conversion. scikit-image offers  
178 robust tools for converting between image data-types (Microsoft, 1995; the Khronos  
179 Group, 2004; Paeth, 1990) and to do file input/output (I/O) operations. Our purpose is  
180 to allow investigators to focus their time on research, instead of expending effort on  
181 mundane low-level tasks.

182 The package includes a number of algorithms with broad applications across image  
183 processing research, from computer vision to medical image analysis. We refer the  
184 reader to the current API documentation for a full listing of current capabilities (the  
185 scikit-image team, 2014). In this section we illustrate two real-world usage examples of  
186 scikit-image in scientific research.

187 First, we consider the analysis of a large stack of images, each representing drying  
188 droplets containing nanoparticles (see Figure 2). As the drying proceeds, cracks propa-  
189 gate from the edge of the drop to its center. The aim is to understand crack patterns by  
190 collecting statistical information about their positions, as well as their time and order of  
191 appearance. To improve the speed at which data is processed, each experiment, consti-  
192 tuting an image stack, is automatically analysed without human intervention. The con-  
193 tact line is detected by a circular Hough transform (`transform.hough_circle`)  
194 providing the drop radius and its center. Then, a smaller concentric circle is drawn  
195 (`draw.circle_perimeter`) and used as a mask to extract intensity values from  
196 the image. Repeating the process on each image in the stack, collected pixels can be  
197 assembled to make a space-time diagram. As a result, a complex stack of images is

198 reduced to a single image summarizing the underlying dynamic process.



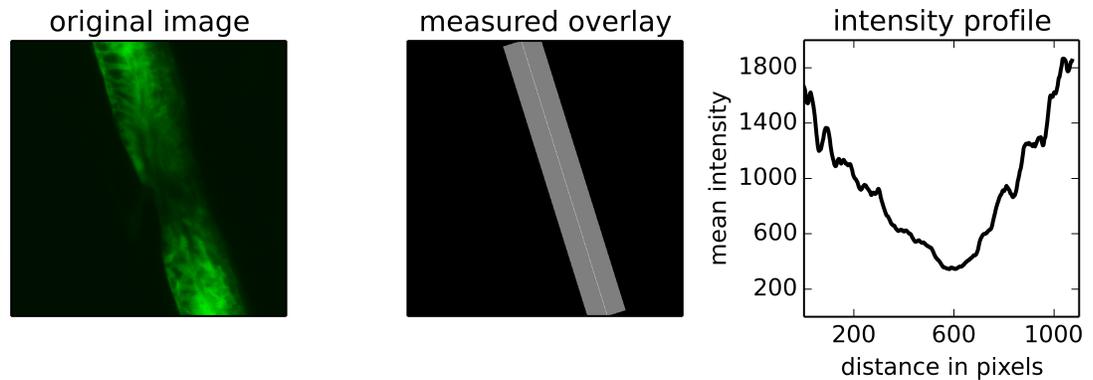
**Figure 2.** Scikit-image is used to track the propagation of cracks (black lines) in a drying colloidal droplet. The sequence of pictures shows the temporal evolution of the system with the drop contact line, in green, detected by the Hough transform and the circle, in white, used to extract an annulus of pixel intensities. The result shown illustrates the angular position of cracks and their time of appearance.

199 Next, in regenerative medicine research, scikit-image is used to monitor the regener-  
200 ation of spinal cord cells in zebrafish embryos (Figure 3). This process has important  
201 implications for the treatment of spinal cord injuries in humans (Bhatt et al., 2004;  
202 Thuret et al., 2006).

203 To understand how spinal cords regenerate in these animals, injured cords are  
204 subjected to different treatments. Neuronal precursor cells (labeled green in Figure 3,  
205 left panel) are normally uniformly distributed across the spinal cord. At the wound site,  
206 they have been removed. We wish to monitor the arrival of new cells at the wound site  
207 over time. In Figure 3, we see an embryo two days after wounding, with precursor  
208 cells beginning to move back into the wound site (the site of minimum fluorescence).  
209 The `measure_profile_line` function measures the fluorescence along the cord,  
210 directly proportional to the number of cells. We can thus monitor the recovery process  
211 and determine which treatments prevent or accelerate recovery.

## 212 Education

213 scikit-image's simple, well-documented application programming interface (API) makes  
214 it ideal for educational use, via self-taught exploration or formal training sessions.



**Figure 3.** The `measure.profile_line` function being used to track recovery in spinal cord injuries. *Left:* an image of fluorescently-labeled nerve cells in an injured zebrafish embryo. *Middle:* the automatically determined region of interest. The SciPy library was used to determine the region extent, and functions from the `scikit-image` `draw` module were used to draw it. *Right:* the image intensity along the line of interest, averaged over the displayed width.

215 The online gallery of examples not only provides an overview of the functionality  
 216 available in the package but also introduces many of the algorithms commonly used  
 217 in image processing. This visual index also helps beginners overcome a common  
 218 entry barrier: locating the class (denoising, segmentation, etc.) and name of operation  
 219 desired, without being proficient with image processing jargon. For many functions, the  
 220 documentation includes links to research papers or Wikipedia pages to further guide the  
 221 user.

222 Demonstrating the utility and ease-of-use of `scikit-image`, thirteen-year-old Rishab  
 223 Gargeya of the Harker School won the Synopsys Silicon Valley Science and Technology  
 224 Championship using `scikit-image` in his project, “A software based approach for auto-  
 225 mated pathology diagnosis of diabetic retinopathy in the human retina” ([science-fair.org](http://science-fair.org),  
 226 2014).

227 We have also delivered image processing tutorials using `scikit-image` at various  
 228 annual scientific Python conferences, such as EuroSciPy ([de Buyl and Pettiaux, 2013](#)),  
 229 PyData 2012 ([PyData organizers, 2012](#)) and SciPy India 2012. Course materials for  
 230 some of these sessions are found in [Haenel et al. \(2014\)](#) and are licensed under the  
 231 permissive CC-BY license ([Creative Commons, 2013](#)). These typically include an intro-  
 232 duction to the package and provide intuitive, hands-on introductions to image processing  
 233 concepts. The well documented application programming interface (API) along with  
 234 tools that facilitate visualization contribute to the learning experience, and make it easy  
 235 to investigate the effect of different algorithms and parameters. For example, when inves-  
 236 tigating denoising, it is easy to observe the difference between applying a median filter  
 237 (`filter.rank.median`) and a Gaussian filter (`filter.gaussian_filter`),  
 238 demonstrating that a median filter preserves straight lines much better.

239 Finally, easy access to readable source code gives users an opportunity to learn how  
 240 algorithms are implemented and gives further insight into some of the intricacies of a  
 241 fast Python implementation, such as indexing tricks and look-up tables.

## 242 Industry

243 Due to the breadth and maturity of its code base, as well as the its commercial-friendly  
244 license, scikit-image is well suited for industrial applications.

245 BT Imaging (BT Imaging, 2014) designs and builds tools that use photoluminescence  
246 (PL) imaging for photovoltaic applications. PL imaging can characterize the quality of  
247 multicrystalline silicon wafers by illuminating defects that are not visible under standard  
248 viewing conditions. The left panel of Figure 4 shows an optical image of a silicon  
249 wafer, and the center panel shows the same wafer using PL imaging. In the right panel,  
250 the wafer defects and impurities have been detected through automated image analysis.  
251 scikit-image plays a key role in the image processing pipeline. For example, a Hough  
252 transform (`transform.hough_line`) finds the wafer edges in order to segment the  
253 wafer from the background. scikit-image is also used for feature extraction. Crystal  
254 defects (dislocations) are detected using a band-pass filter, which is implemented as a  
255 Difference of Gaussians (`filter.gaussian_filter`).

256 The image processing results are input to machine learning algorithms, which assess  
257 intrinsic wafer quality. Solar cell manufacturers can use this information to reject poor  
258 quality wafers and charge more for cells that are expected to have high efficiency.



**Figure 4.** *Left:* An image of an as-cut silicon wafer before it has been processed into a solar cell. *Center:* A PL image of the same wafer. Wafer defects, which have a negative impact solar cell efficiency, are visible as dark regions. *Right:* Image processing results. Defects in the crystal growth (dislocations) are colored blue, while red indicates the presence of impurities.

259 scikit-image is also applied in a commercial setting for biometric security applica-  
260 tions. AICBT Ltd uses multispectral imaging to detect when a person attempts to conceal  
261 their identity using a facial mask (AICBT, Ltd., 2014). scikit-image performs file I/O  
262 (`io.imread`), histogram equalization (`exposure.equalize_hist`), and aligns a  
263 visible wavelength image with a thermal image (`transform.AffineTransform`).  
264 The system determines the surface temperature of a subject's skin and detects situations  
265 where the face is being obscured.

## 266 EXAMPLE: IMAGE REGISTRATION AND STITCHING

267 This section gives a step-by-step outline of how to perform panorama stitching using the  
268 primitives found in scikit-image. The full source code is at <https://github.com/>

269 `scikit-image/scikit-image-demos.`

## 270 **Data loading**

271 The “ImageCollection” class provides an easy way of representing multiple images on  
272 disk. For efficiency, images are not read until accessed.

```
from skimage import io
ic = io.ImageCollection('data/*')
```

273 Figure 5a shows the Petra dataset, which displays the same facade from two different  
274 angles. For this demonstration, we will estimate a projective transformation that relates  
275 the two images. Since the outer parts of these photographs do not conform well to such  
276 a model, we select only the central parts. To further speed up the demonstration, images  
277 are downscaled to 25% of their original size.

```
from skimage.color import rgb2gray
from skimage import transform

image0 = rgb2gray(ic[0][:, 500:500+1987, :])
image1 = rgb2gray(ic[1][:, 500:500+1987, :])

transform = transform.ProjectiveTransform()
image0 = transform.rescale(image0, 0.25)
image1 = transform.rescale(image1, 0.25)
```

## 278 **Feature detection and matching**

279 “Oriented FAST and rotated BRIEF” (ORB) features (Rublee et al., 2011) are detected in  
280 both images. Each feature yields a binary descriptor; those are used to find the putative  
281 matches shown in Figure 5b.

```
from skimage.feature import ORB, match_descriptors

orb = ORB(n_keypoints=1000, fast_threshold=0.05)

orb.detect_and_extract(image0)
keypoints1 = orb.keypoints
descriptors1 = orb.descriptors

orb.detect_and_extract(image1)
keypoints2 = orb.keypoints
descriptors2 = orb.descriptors

matches12 = match_descriptors(descriptors1,
                              descriptors2,
                              cross_check=True)
```

## 282 Transform estimation

283 To filter the matches, we apply RANdom SAmple Consensus (RANSAC) (Fischler and  
284 Bolles, 1981), a common method for outlier rejection. This iterative process estimates  
285 transformation models based on randomly chosen subsets of matches, finally selecting  
286 the model which corresponds best with the majority of matches. The new matches are  
287 shown in Figure 5c.

```
from skimage.measure import ransac

# Select keypoints from the source (image to be
# registered) and target (reference image).

src = keypoints2[matches12[:, 1]][:, :-1]
dst = keypoints1[matches12[:, 0]][:, :-1]

model_robust, inliers = \
    ransac((src, dst), ProjectiveTransform,
          min_samples=4, residual_threshold=2)
```

## 288 Warping

289 Next, we produce the panorama itself. The first step is to find the shape of the output  
290 image by considering the extents of all warped images.

```
r, c = image1.shape[:2]

# Note that transformations take coordinates in
# (x, y) format, not (row, column), in order to be
# consistent with most literature.
corners = np.array([[0, 0],
                    [0, r],
                    [c, 0],
                    [c, r]])

# Warp the image corners to their new positions.
warped_corners = model_robust(corners)

# Find the extents of both the reference image and
# the warped target image.
all_corners = np.vstack((warped_corners, corners))

corner_min = np.min(all_corners, axis=0)
corner_max = np.max(all_corners, axis=0)

output_shape = (corner_max - corner_min)
output_shape += np.abs(corner_min)
output_shape = output_shape[:-1]
```

291 The images are now warped according to the estimated transformation model. Values  
292 outside the input images are set to -1 to distinguish the “background”.

293 A shift is added to ensure that both images are visible in their entirety. Note that  
294 warp takes the *inverse* mapping as input.

```

from skimage.color import gray2rgb
from skimage.exposure import rescale_intensity
from skimage.transform import warp
from skimage.transform import SimilarityTransform

offset = SimilarityTransform(translation=-corner_min)

image0_ = warp(image0, offset.inverse,
                output_shape=output_shape, cval=-1)

image1_ = warp(image1, (offset + model_robust).inverse,
                output_shape=output_shape, cval=-1)

```

295 An alpha channel is added to the warped images before merging them into a single  
296 image:

```

def add_alpha(image, background=-1):
    """Add an alpha layer to the image.

    The alpha layer is set to 1 for foreground
    and 0 for background.
    """
    rgb = gray2rgb(image)
    alpha = (image != background)
    return np.dstack((rgb, alpha))

image0_alpha = add_alpha(image0_)
image1_alpha = add_alpha(image1_)

merged = (image0_alpha + image1_alpha)
alpha = merged[..., 3]

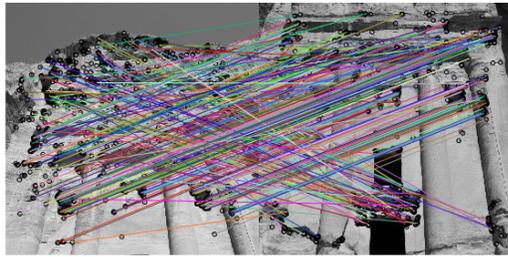
# The summed alpha layers give us an indication of
# how many images were combined to make up each
# pixel. Divide by the number of images to get
# an average.
merged /= np.maximum(alpha, 1)[..., np.newaxis]

```

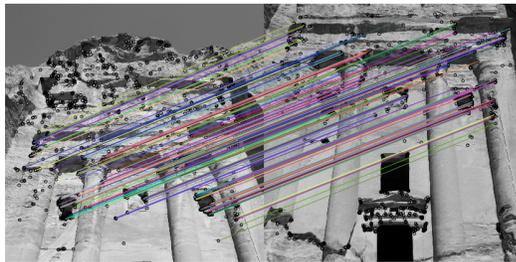
297 The merged image is shown in Figure 5d. Note that, while the columns are well  
298 aligned, the color intensities at the boundaries are not well matched.



(a) Petra images



(b) ORB binary features



(c) RANSAC-filtered features



(e) Final result, combined with Enblend



(d) Warped & positioned

**Figure 5.** An example application of scikit-image: image registration and warping to combine overlapping images. (a): Photographs taken in Petra, Jordan by François Malan. License: CC-BY. (b): Putative matches computed from ORB binary features. (c): Matches filtered using RANSAC. (d): The second input frame (*middle*) is warped to align with the first input frame (*left*), yielding the averaged image shown on the right. (e): The final panorama image, registered and warped using scikit-image, blended with Enblend.

## 299 **Blending**

300 To blend images smoothly we make use of the open source package Enblend (Dersch  
301 and PanoTools contributors, 2010), which in turn employs multi-resolution splines and  
302 Laplacian pyramids (Burt and Adelson, 1983b), (Burt and Adelson, 1983a). The final  
303 panorama is shown in Figure 5e.

## 304 **CONCLUSION**

305 scikit-image provides easy access to a powerful array of image processing functionality.  
306 Over the past few years, it has seen significant growth in both adoption and contribution,  
307 and the team is excited to collaborate with others to see it grow even further, and to  
308 establish it the de facto library for image processing in Python.

## 309 **ACKNOWLEDGEMENTS**

310 We thank Timo Friedrich and Jan Kaslin for providing the zebrafish lesion data. Portions  
311 of the research reported in this publication was supported by the National Institute of  
312 Diabetes and Digestive and Kidney Diseases of the National Institutes of Health under  
313 award number F30DK098832. Portions of the research reported in this publication were  
314 supported by the Victorian Life Sciences Computation Initiative. The content is solely  
315 the responsibility of the authors and does not necessarily represent the official views of  
316 the National Institutes of Health.

## 317 **REFERENCES**

- 318 AICBT, Ltd. (2014). Disguise detection. [http://www.aicbt.com/  
319 disguise-detection/](http://www.aicbt.com/disguise-detection/) Accessed: 2014-03-30.
- 320 Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D., and Smith, K. (2011).  
321 Cython: The best of both worlds. *Computing in Science and Engineering*, 13(2):31–  
322 39.
- 323 Bhatt, D., Otto, S., Depoister, B., and JR., F. (2004). Cyclic amp-induced repair of  
324 zebrafish spinal circuits. *Science*, 305:254–258.
- 325 Brandl, G. and the Sphinx team (2007). <http://sphinx-doc.org/> Accessed:  
326 2014-03-30.
- 327 BT Imaging (2014). BT Imaging. <http://www.btimaging.com> Accessed: 2014-  
328 03-30.
- 329 Burt, P. and Adelson, E. (1983a). The laplacian pyramid as a compact image code. *IEEE*  
330 *Transactions on Communications*.
- 331 Burt, P. and Adelson, E. (1983b). A multiresolution spline with application to image  
332 mosaics. *ACM Transactions on Graphics*, 2(4):217–236.
- 333 Canny, J. (1986). A computational approach to edge detection. *IEEE Trans. Pattern*  
334 *Analysis and Machine Intelligence.*, 8:679–714.
- 335 Canonical, Ltd. (2004). python-skimage in Ubuntu. [http://packages.ubuntu.  
336 com/search?keywords=python-skimage](http://packages.ubuntu.com/search?keywords=python-skimage) Accessed: 2014-03-30.
- 337 Coelho, L. (2013). Mahotas: Open source software for scriptable computer vision.  
338 *Journal of Open Research Software*, 1(1).

- 339 Continuum Analytics (2012). The Anaconda Scientific Python Distribution. <https://store.continuum.io/cshop/anaconda/> Accessed: 2014-03-30.
- 340
- 341 Cordasco, I. (2010). <https://pypi.python.org/pypi/flake8> Accessed:
- 342 2014-03-30.
- 343 Creative Commons (2013). CC-BY license. <http://creativecommons.org/licenses/by/4.0/> Accessed: 2014-03-30.
- 344
- 345 de Buyl, P. and Pettiaux, N. (2013). Euroscipy 2013. <https://www.euroscipy.org/2013/schedule/presentation/3/> Accessed: 2014-03-30.
- 346
- 347 Dersch, H. and PanoTools contributors (2010). Enblend 4.0 documentation. <http://enblend.sourceforge.net> Accessed: 2014-03-30.
- 348
- 349 Enthought, Inc (2014). Enthought canopy. <https://www.enthought.com/products/canopy/> Accessed: 2014-03-30.
- 350
- 351 Fischler, M. and Bolles, R. (1981). Random sample consensus: A paradigm for model
- 352 fitting with applications to image analysis and automated cartography. *Comm. of the*
- 353 *ACM*, 24(6):381–395.
- 354 Gommers, R. and the NumPy developers (2010). [https://github.com/numpy/numpy/blob/master/doc/HOWTO\\_DOCUMENT.rst.txt](https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt) Accessed: 2014-
- 355 03-30.
- 356
- 357 Google (2004). Google Summer of Code. <https://developers.google.com/open-source/soc> Accessed: 2014-03-30.
- 358
- 359 Haenel, V., Gouillart, E., Varoquaux, G., and scipy-lecture-notes contributors (2014).
- 360 Scipy lecture notes. <http://scipy-lectures.github.io/> Accessed:
- 361 2014-03-30.
- 362 Halchenko, Y. and Hanke, M. (2012). Open is not enough. let’s take the next step:
- 363 an integrated, community-driven computing platform for neuroscience. *Front. Neu-*
- 364 *roinf.*, 6:22. <http://neuro.debian.net/pkg/python-skimage.html>
- 365 Accessed: 2014-03-30.
- 366 LEMUR Heavy Industries (2013). <https://coveralls.io> Accessed: 2014-03-
- 367 30.
- 368 Microsoft (1995). <http://msdn.microsoft.com/en-us/library/windows/desktop/dd607323> Accessed: 2014-03-30.
- 369
- 370 Ohloh (2014). Scikit-image on ohloh. <https://www.ohloh.net/p/scikit-image> Accessed:
- 371 <https://www.ohloh.net/p/scikit-image> Accessed:
- 372 2014-03-30.
- 373 Paeth, A. (1990). *Proper treatment of pixels as integers*. Graphics Gems.
- 374 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel,
- 375 M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,
- 376 D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: machine learning
- 377 in python. *Journal of Machine Learning Research*, 12:2825–2830.
- 378 PyData organizers (2012). Pydata 2012. <https://www.youtube.com/watch?v=Wvvxazwi2IY> Accessed: 2014-03-30.
- 379
- 380 PyPI (2014). scikit-image 0.9.3 at the Python Package Index. <https://pypi.python.org/pypi/scikit-image> Accessed: 2014-03-30.
- 381
- 382 Raybaut, P. (2014). Python(x,y). <https://code.google.com/p/pythonxy/>
- 383 Accessed: 2014-03-30.
- 384 Regents of the University of California (1999). <http://www.gnu.org/>

- 385 [licenses/license-list.html#ModifiedBSD](#) Accessed: 2014-03-30.
- 386 Rublee, E., Rabaud, V., Konolige, K., and G., B. (2011). ORB: An efficient alternative to  
387 SIFT and SURF. In *Proceedings of the 2011 International Conference on Computer*  
388 *Vision (ICCV)*, pages 2564–2571.
- 389 science-fair.org (2014). [http://science-fair.org/database/project\\_](http://science-fair.org/database/project_)  
390 [awards.php?schoolname=Privately+Sponsored+Project&](http://science-fair.org/database/project_)  
391 [school\\_year=2014](http://science-fair.org/database/project_) Accessed: 2014-03-30.
- 392 the Khronos Group (2004). <https://www.khronos.org/registry/gles/>  
393 [specs/2.0/es\\_full\\_spec\\_2.0.25.pdf](https://www.khronos.org/registry/gles/) Accessed: 2014-03-30.
- 394 the scikit-image developers (2010). <https://github.com/scikit-image/>  
395 [scikit-image](https://github.com/scikit-image/) Accessed: 2014-03-30.
- 396 the scikit-image team (2010a). <https://github.com/scikit-image/>  
397 [scikit-image](https://github.com/scikit-image/) Accessed: 2014-03-30.
- 398 the scikit-image team (2010b). <https://groups.google.com/forum/?&>  
399 [fromgroups#!forum/scikit-image](https://groups.google.com/forum/?&) Accessed: 2014-03-30.
- 400 the scikit-image team (2014). <http://scikit-image.org/docs/dev/> Ac-  
401 cessed: 2014-03-30.
- 402 the Travis-CI community (2012). <https://travis-ci.org> Accessed: 2014-03-  
403 30.
- 404 Thuret, S., Moon, L., and Gage, F. (2006). Therapeutic interventions after spinal cord  
405 injury. *Nature Rev Neurosci*, 7:628–643.
- 406 van der Walt, S., Colbert, C., and G, V. (2011). The NumPy array: a structure for efficient  
407 numerical computation. *Computing in Science and Engineering*, 13(2):22–30.
- 408 van Rossum, G., Warsaw, B., and N, C. (2001). <http://www.python.org/dev/>  
409 [peps/pep-0008/](http://www.python.org/dev/) Accessed: 2014-03-30.
- 410 Wikipedia (2014). [http://en.wikipedia.org/wiki/Software\\_](http://en.wikipedia.org/wiki/Software_)  
411 [versioning](http://en.wikipedia.org/wiki/Software_) Accessed: 2014-03-30.