Basics of Image Processing in Soft Robotics Module

Qiukai Qi (qiukai.qi@bristol.ac.uk) and Andrew Conn (a.conn@bristol.ac.uk)

March 14, 2023

Abstract

This tutorial provides a guide on some image processing basics required for the detection of deformation in soft robotics. Python is adopted with other essential libraries such as OpenCV and Numpy in PyCharm. It starts from the basics of image processing and OpenCV operations, then demonstrates a simple example of motion detection and tracking of a pneumatic gripper, and finally introduces some more advanced algorithms that are common in soft robotics research. This tutorial serves as a basic introductory document that leads students to more advanced algorithms in the literature.

1 Useful Sources

- A comprehensive book on computer vision by Richard Szeliski, "Computer vision: Algorithms and Applications"

  http://szeliski.org/Book/

- A online lecture notes on "Digital Image Processing" from MIT:

  https://web.stanford.edu/class/ee368/handouts.html

- A github repository for a textbook on opencv "Mastering OpenCV 4 with Python"

  https://github.com/PacktPublishing/Mastering-OpenCV-4-with-Python

- A useful python integrated development environment (IDE) community version:

  https://www.jetbrains.com/pycharm/

- Python: https://www.python.org/
- OpenCV: https://opencv.org/

2 Basics of Image Processing

Image

An image can be seen as a two-dimensional (2D) view of the 3D world [1]. It can be described as a 2D function, $f(x, y)$, where (x, y) is the spatial coordinates and the f is the value of that point which is usually proportional to the brightness or the gray level of the image. Both (x, y) and f can be continuous or discrete. When both are finite discrete quantities, the image is called a digital image.

Pixel

Pixels (Picture Element) are the basic building blocks of a digital image [2]. It is the smallest unit of a digital image that can be displayed and represented on a digital display device. For example, an image with a resolution of 500×500 is a grid or matrix of 500 columns and 500 rows of pixels. Note that knowing the number of pixels in an image does not indicate its physical dimension. It has to be combined with pixels per inch (PPI) in order to know the specific size. In a simple sense, pixels can be the basic unit for the spatial coordinates (x, y) of an image.

Colour Space

Colour space is specific organization of colours. There are different colour models, e.g., RGB (red, green, blue) HSL (hue, saturation, lightness) and HSV (hue, saturation, value), and RGB is the most common one. In RGB model, each of the primary colours (R, G, or B) is represented as an integer in the range of [0, 255] which corresponds to the value of f mentioned above. A given colour at point (x, y) is generated by mixing the three colours additively ($f_R(x, y) + f_G(x, y) + f_B(x, y)$). For example, mixing Red ($f_R = 255$), Green($f_G = 255$) and Blue ($f_B = 255$) yields White colour indicated in the center of Figure 1. Adding Red to Green gives Yellow.

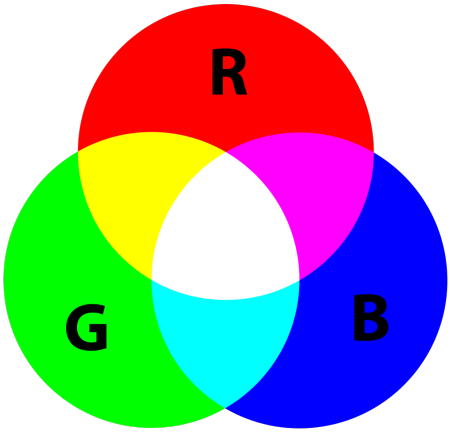

Figure 1: RGB colour space.

3 Basic Operations in OpenCV

An image is stored in OpenCV as data in a matrix form. These data can be read, manipulated (e.g., change colour), and saved as another image. This section introduces some basic operations in OpenCV. Other higher-level operations such as thresholding, contour detection, face recognition etc. are not included.

Coordinate in OpenCV

The origin of the image coordinate system is located at the top left corner pixel of the image, marked as $(0, 0)$. From this point, the x coordinate of (x, y) gets larger as it goes right and y coordinate gets larger as it goes down. However, when accessing pixels in OpenCV using Numpy, the reference format is $[y, x]$. This is a key point that often confuses beginners. A tip that helps to remember is that x is the width and y is the height of the image.

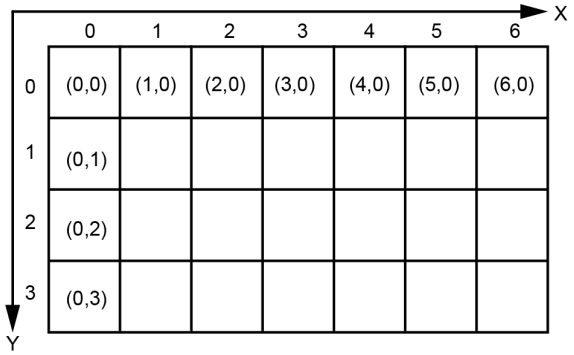

Figure 2: Coordinates of pixels in an image.

Access an image and pixels

Read the image into the workspace with $cv2.imread()$ function.

```
img=cv2.imread(image_directory)
```

This gives access to the properties of an image, such as colour and dimension, that can be called and processed using specific functions.

For example, $dim = img.shape$ returns the tuple (height, width, channels) to the variable dim. To read the pixel value, we would use some function like this, $(b, g, r) = img[6, 40]$, which returns the Blue, Green, and Red values of pixel $(x = 40, y = 6)$. Note the (B, G, R) is the form in OpenCV rather than (R, B, G).

Display an image

Display the image in an image frame with $cv2.imshow()$ function, and $cv2.waitKey()$ function to bind the keyboard function and to wait for a specified number of milliseconds. $cv2.waitKey(0)$ means waiting indefinitely for a keystroke to continue running.

```
cv2.imshow(image_frame, img)
cv2.waitKey(0)
```

Convert between colour spaces

Convert the image from one colour space from one to the other with this function $cv2.cvtColor()$ function.

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

This converts the original rgb image to grayscale, and returns to the variable img_gray. More than 150 colour spaces are supported in OpenCV, and all are convertible accoridng to different purposes. For example, grayscale images are preferred for some image processing operations such as edge detection. This helps to simplify the algorithm and reduce the complexity.

Write an image

Write the image after being processed into a new image with $cv2.write()$ function.

```
cv2.write(image_directory, img_gray)
```

This saves the grayscale image into the image directory.

4 Deformation Detection

Non-invasive image processing and computer vision have been important techniques for soft robotics characterizations such as deformation detection, shape change detection, and dynamics analysis. This section demonstrates a simple deformation detection task of a robotic finger, as indicated in Fig. 3. The pneumatic finger bends and unbends under actuation by a syringe. A web camera is placed in front to detect its deformation by tracking and analyzing the markers on the finger. The sample testing video (output.avi) was recorded while pressurizing and depressurizing the finger, and all analysis here is based on this video. Here the objective is to extract the finger deformation by tracking the locations of markers in the image.

Original images

Here is a sample image extracted from the video file.

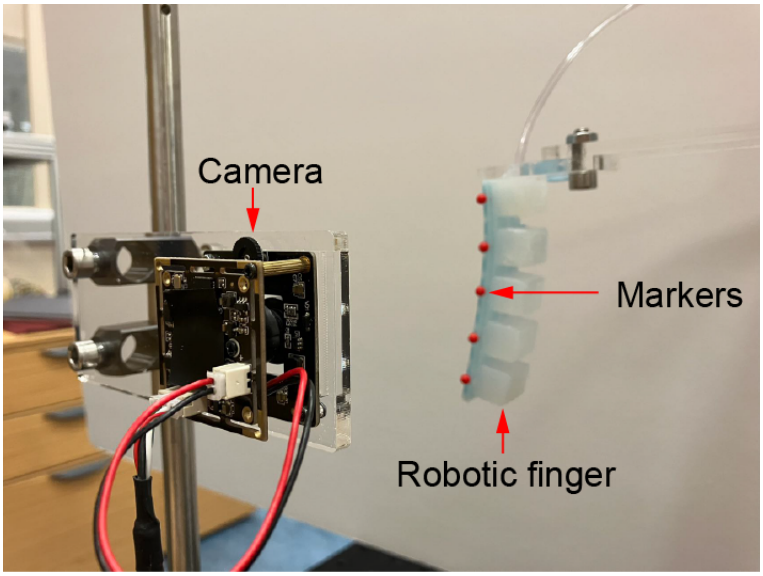

Figure 3: The testing setup of deformation detection.

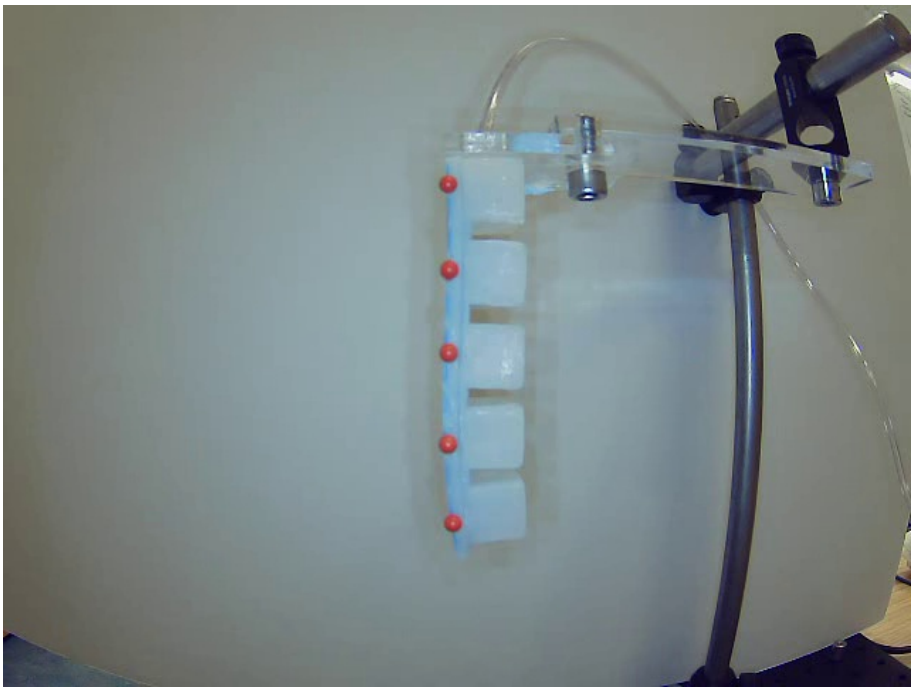

Figure 4: The original image.

Cropping to ROI

Depending on the camera setting, you'll have images of different sizes. In this case, the image dimension is $[480 \times 640]$. Most of the cases it contains much noise information and you'll need to focus on a particular region of interest/image (ROI) in order to simplify the algorithm, reduce the computation time and enhance precision. The cropping is usually done by reselecting the data region from the original matrix. The original image is stored in variable *img_rgb*.

```
img_rgb = img_rgb[row_start:row_end, colunm_start:column_end],
```

where row_start, row_end, colunm_start, column_end are the start and end rows and columns of the ROI, respectively. Very often you'll need to blur the image to further reduce the noise using

*cv*2.*medianBlur*() function.

Figure 5: The region of interest (ROI) cropped from original image.

Converting into hsv colour space

This is to prepare for the colour extraction in the next step.

```
img_hsv = cv.cvtColor(img_rgb, cv.COLOR_BGR2HSV).
```

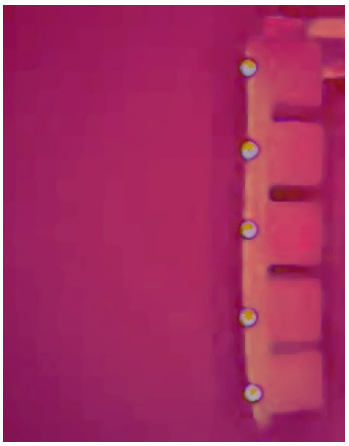

Figure 6: The image in HSV colour space.

Colour detection using inRange function

There are different strategies for marker tracking. A simple algorithm is to use *cv*2.*inRange*() function to extract the color. It returns the binary form of the original image with target colour highlighted as in Fig. 7.

```
mask = cv2.inRange(img_hsv, lowerb=red_hsv_lower, upperb=red_hsv_higher),
```

where red_hsv_lower and red_hsv_upper are the lower and upper hsv thresholds for red colour.

Center detection

Since the *cv*2.*inRange*() function returns the binary image as a matrix, you'll still need to extract the coordinates of the markers by finding the center coordinate of each marker for further analysis. This is done by first detecting the contours of each marker using *cv*2.*findContours*() function, and then finding the center of the enclosing circle of each marker with *cv*2.*minEnclosingCircle*() function.

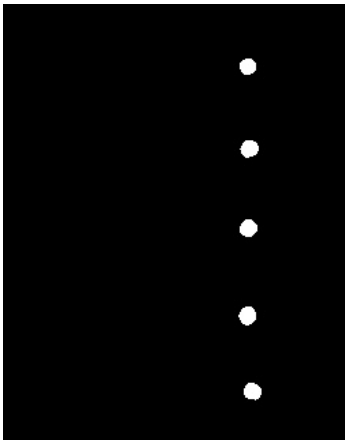

Figure 7: The extracted markers in binary scale.

```
contours, hierarchy = cv.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_NONE),
```

wehre cv.RETR_TREE and cv.CHAIN_APPROX_NONE preset parameters for specific algorithm and contours contain boundary pixels of each marker.

```
(x, y), radius = cv.minEnclosingCircle(cnt),
```

where cnt is the contour list and (x,y) and radius represent the center and radius of the enclosing circle. 8 indicates the image with detected enclosing circles and 9 shows the connecting line segments between markers as an approximation of the boundary of the finger.

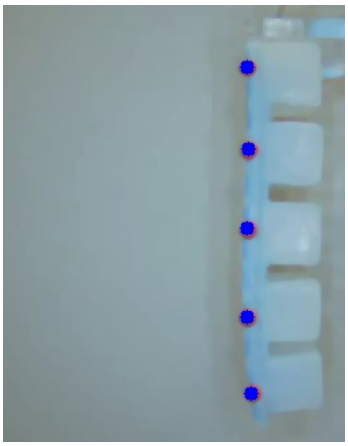

Figure 8: The ROI image with tracked markers.

Sorting algorithm

The sorting algorithm is used to ensure the markers in each frame to be stored in the same order for future use. Check main.py for details.

5 Further Readings on Advanced Algorithms

Quantifying shape changes in soft robotics using Elliptic Fourier descriptors [3]: Algorithms and sample code can be found in the article [4]. By using this method, the shape changes can be successfully detected from video frames, and motion analysis is achieved, as shown in 10.

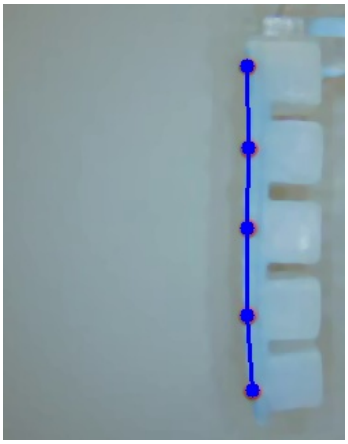

Figure 9: The ROI with approximation curve estimated from markers.

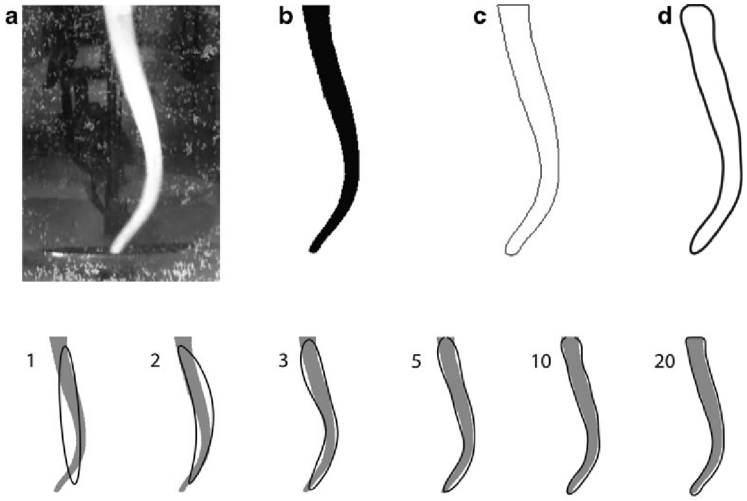

Figure 10: Shape characterization of an octopus arm using elliptic Fourier descriptors.

Extracting object centerline from video: Algorithms and sample code can be found in the article [5]. Fig. 11 outlines the algorithms. Skeletal dynamics can be analyzed further based on this method.

References

[1] Alberto Fernández Villán. *Mastering OpenCV 4 with Python: a practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7*. Packt Publishing Ltd, 2019.

[2] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[3] Frank P Kuhl and Charles R Giardina. "Elliptic Fourier features of a closed contour". In: *Computer graphics and image processing* 18.3 (1982), pp. 236–258.

[4] Krishna Manaswi Digumarti et al. "Quantifying dynamic shapes in soft morphologies". In: *Soft Robotics* 6.6 (2019), pp. 733–744.

[5] Katsuma Inoue et al. "Skeletonizing the dynamics of soft continuum body from video". In: *Soft Robotics* 9.2 (2022), pp. 201–211.

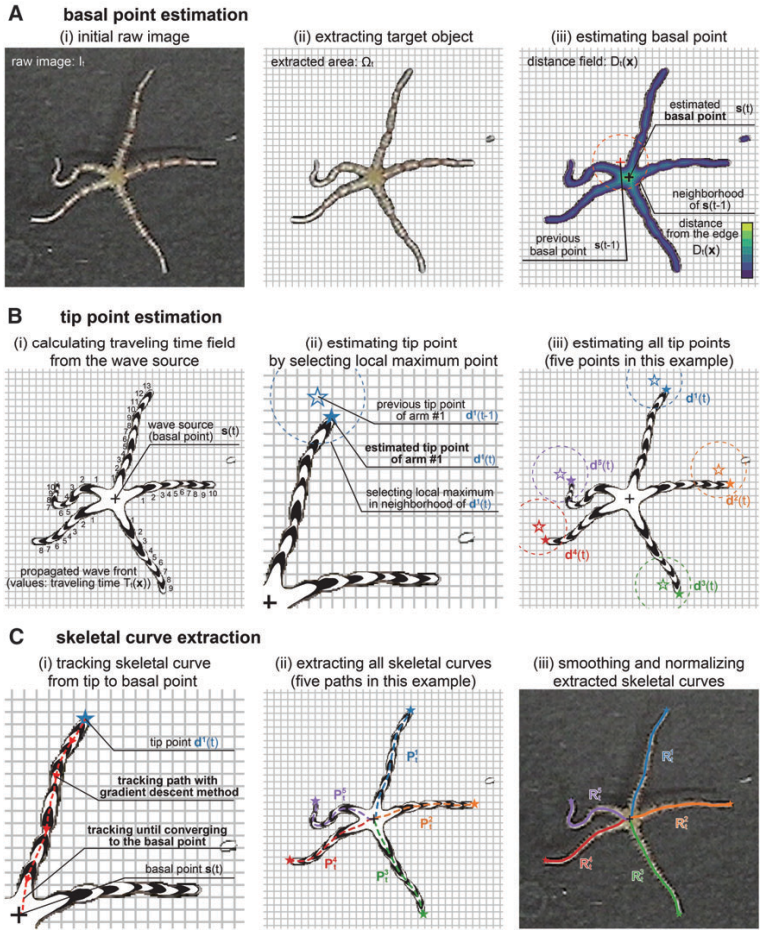

Figure 11: Centerline extraction scheme.