A Practical Digital Image Processing Course with `morph.py`

Francisco de Assis Zampirolli, João Marcelo Borovina Josko, Fernando Teubl, Celso Setsuo Kurashima
{fzampirolli,marcelo.josko,fernando.teubl,celso.kurashima}@ufabc.edu.br
Federal University of ABC (UFABC)
Av. dos Estados, 5001 – Santo André – 09210-580 – SP – Brazil

ABSTRACT

Teaching Digital Imaging Processing (DIP) is challenging, primarily because of its mathematical and algorithms complexities. Despite the recent growth in the field, comprehensive resources are lacking to support DIP education. To address this gap, this paper introduces a practical course utilizing a Python library named `morph.py` designed for beginners and accessible on Google Colab. This interactive course employs illustrative examples and hands-on exercises to facilitate the learning of fundamental DIP concepts and operators. It begins with basic concepts (e.g., image representation) and progresses to more advanced topics, including image transformations and feature extraction. We conducted an exploratory case study in one group ($N = 15$) and gathered their perception through a voluntary survey. Our quantitative analysis strongly supports our teaching method's effectiveness based on the `morph.py` library, which addresses the difficulties of teaching DIP to beginners.

KEYWORDS

Computing Education, Digital Image Processing, Colab.

1 INTRODUCTION

Digital Imaging Processing – DIP – is a computer science and engineering discipline dedicated to analyzing, manipulating, and interpreting digital images. This field has experienced substantial growth in recent years, driven by the expanding accessibility of digital images and the rising demand for image analysis across diverse domains (e.g., medical imaging, remote sensing, and autonomous vehicles) [4].

However, learning DIP can be challenging for beginners. It requires them to understand several theoretical concepts and develop practical skills. Hence, isolating lectures and reading materials may not be the most effective teaching and learning approach for subjects such as DIP. Interactive and practical methods can better engage students and support learning, according to studies by [17, 24].

Literature provides a few toolboxes to support DIP learning. Most toolboxes are based on a virtual notebook environment and provide a comprehensive set of functions for image analysis [17, 22]. However, some works are outdated, such as the `mmorph` toolbox that was used in Dougherty and Lotufo's book [3]. Additionally, none of them provide constructive feedback to students [24] or address morphological operators [17].

Furthermore, there are numerous studies aimed at teaching DIP using commercial libraries [24], characterized by complex syntax [23], unavailable environments [7], or outdated languages with limited DIP capabilities, such as Java [19, 20]. Additionally, no literature was found addressing the automated assessment of programming exercises in the context of DIP.

This paper presents an interactive DIP course utilizing a toolbox named `morph.py` developed especially in response to the identified gap. The course employs a hands-on approach, teaching theoretical concepts through practical examples and exercises, fostering the development of practical skills and problem-solving abilities in students. This course also counts on an automatic correction environment that provides feedback for each student's submitted DIP exercises. The introductory lessons of this course focus on fundamental concepts (e.g., image representation and sampling). In contrast, advanced ones include computer vision applications such as feature extraction and object detection.

We conducted a voluntary survey to gather students' ($N = 15$) perceptions of our DIP course approach. Our quantitative analysis revealed that the interactive format of the course contributed to students' learning of DIP concepts. Results indicate that this practical teaching approach is beneficial in addressing the challenges of teaching complex technical topics to beginners.

This work is organized as follows: In Section 2, we provide background information on DIP and review relevant previous studies. Our pedagogical approach is described in Section 3. The implementation of `morph.py` is presented in Section 4. In Section 5, we detail the usage of the `morph.py` library in exams. In Section 6, we present the results and engage in corresponding discussions. Lastly, we conclude this work in Section 7.

2 BACKGROUND

A digital image in two dimensions (2D) can be represented by a function $f(x, y)$ defined over a finite subset of the Cartesian plane (x, y), where the codomain is the interval $[0, k]$. When $k = 1$, f represents a binary image. For $k = 255$, it corresponds to a grayscale image. If f assumes three values within this interval, it can be defined as a color image in RGB (Red, Green, and Blue) format [4].

DIP is a field of computer science and engineering that deals with the analysis, manipulation, and interpretation of digital images, as seen in the preview. Python has gained popularity as a language for DIP due to its user-friendly nature, expansive community, and access to powerful libraries like OpenCV (opencv.org), Pillow (pillow.readthedocs.io), Skimage (scikit-image.org), and TensorFlow (tensorflow.org). These libraries offer a wide variety of methods

EduComp'24, Abril 22-27, 2024, São Paulo, Brasil (On-line)

for DIP applications, including those based on Mathematical Morphology (MM). Operators like dilation, erosion, opening, closing, and others are handy for image enhancement, segmentation, and feature extraction tasks.

`MMach` is a DIP library designed for MM and was developed by Prof. Dr. Junior Barrera [1]. `MMach` had contributions from several researchers, including students at different academic levels, and different versions were generated as Khoros versions changed [7]. In the mid-1990s, Prof. Dr. Roberto de Alencar Lotufo joined the team to make `MMach` platform-independent [11]. The most recent version of `MMach`, known as `mmorph`, was developed using AdessoWiki [13]. It is structured in XML and includes automatic code generation for C, Matlab, and Python languages and automatic documentation generation in TXT, HTML, and LaTeX formats. Although `mmorph` has been used in several publications, the latest version is no longer available online. One excellent example of its usage can be found in Dougherty and Lotufo's book [3].

This paper proposes using `morph.py`, an open-source Python library that provides methods for DIP. The library is available on github.com/fzampirolli/morph, and allows for community contributions. `morph.py` has an easy-to-use interface and offers a wide range of morphological operators on images, successfully utilized in various applications in DIP courses, including image segmentation, edge detection, and object recognition. The library also offers different implementations of the same operator, allowing students to compare them with those available in libraries like OpenCV and Skimage. The goal of `morph.py` is not only to encapsulate functions from other libraries (such as OpenCV), but also to provide a more simplified interface, allowing for a better focus on the addressed content. Additionally, it aims to standardize the behavior of operations, enabling the development of questions with predictable answers, which is crucial in activities involving automated evaluations.

Related Works

This section provides an overview of scientific papers and currently available toolboxes for teaching DIP topics. Silva et al. [22] developed a toolbox for teaching DIP courses using Python and Adesso [12]. This toolbox was used in one of the earliest Python-based DIP courses (dca.fee.unicamp.br/ia636) at UNICAMP, consisting of 83 DIP methods with code and documentation. However, the toolbox has not been updated. A newer version of the ia636 course is now available at github.com/MICLab-Unicamp/ia636, but it only includes 78 Python-based DIP methods without documentation. Prof. Lotufo's github.com/robertoalotufo/ia898 toolbox is a more comprehensive version of the ia636 course, including five lessons and dozens of documented methods with Jupyter Notebooks. However, this toolbox provides no morphological operators nor support for addition of new operators. Prof. Lotufo has developed another toolbox for DIP courses at github.com/robertoalotufo/ia870p3. This toolbox includes a comprehensive documentation of the `mmorph` toolbox, with methods now implemented in Python and serves as supplementary material to the book by Dougherty and Lotufo [3]. These toolboxes inspired the development of `morph.py`, a more recent toolbox that includes morphological operators and allows contributions from the community.

Rowe et al. [17] present a DIP teaching method using Jupyter Notebook, showing increased student comfort with Python and exposure to polar data. Yaniv et al. [25] introduce the SimpleITK toolkit for reproducible research and image analysis workflows through Jupyter Notebook. Although their work focuses on providing a toolkit for image analysis, our approach is different as we introduce a library specifically designed to support the teaching of DIP, including the construction of new operators. Yahya et al. [24] propose using Matlab to teach DIP, utilizing visual, interactive, and experimental methods.

The Matlab Image Processing Toolbox is a popular tool used in scientific and industrial applications for image analysis. It offers a wide range of functions and algorithms for image-processing tasks [14]. However, its proprietary nature can be a disadvantage, as it may limit access for users without licenses and pose financial challenges for those with limited budgets. In contrast, the Python environment, with its open community and collaborative nature, allows for constant improvements and innovations.

Among our review of related literature, only [8, 17] and our study included student feedback on teaching DIP. Additionally, we replicated some of the demonstrations from the ia870p3 course using `morph.py` in Colab and presented different implementations of the same operator. Notably, our pedagogical approach only requires the `morph.py` file, which can be easily inserted into a VPL (Virtual Programming lab for Moodle) activity for automatic code correction [16].

It is also possible to teach DIP using problem-solving approaches. In [8], an experiment was described in a class of 37 students to solve eleven DIP problems using group work in the Python language. This group work approach is questionable because the paper did not detail whether all students were able to absorb the DIP skills and competencies effectively.

Table 1 summarizes the similarities and differences between the toolboxes discussed in this section.

3 METHOD

This section presents the context of the DIP course (Section 3.1) and our pedagogical intervention (Section 3.2), focusing in how we used the `morph.py` library as a motivational factor. Finally, we discuss the development environments (Section 3.3).

3.1 Our Context

DIP is an elective course available to students in multiple programs at our university, such as the Bachelor of Science in Computer Science and various Engineering. The course lasts for 12 weeks, with four hours of class per week. The professor delivers synchronous laboratory classes to students through the projector screen, presenting Colabs containing concepts, examples, and exercises within the Moodle Learning Management System. In 2023, we offered the course to 25 students from February to May. We also provide recorded classes created during the Covid-19 pandemic.

3.2 Pedagogical Intervention

The course covers a comprehensive range of topics in DIP, with the first six weeks focused on equipping students with the necessary skills to develop DIP operators such as thresholds, histograms, convolution, erosion, dilation, filters, watershed, labeled, and distance

Table 1: Comparing toolboxes/papers for teaching DIP

Toolbox/ Paper	Teaching Tool	Focus	Morphological Operators	Student Feedback
Silva, 2003 [22]	Python	Toolbox	No	No
ia898 by Lotufo, 2017 [9]	Jupyter	Toolbox	No	No
ia870p3 by Lotufo, 2019 [10]	Jupyter	Toolbox	Yes	No
Yaniv, 2018 [25]	Jupyter	Toolbox	No	No
Rowe, 2018 [17]	Jupyter	Polar image	No	Yes
Yahya, 2019 [24]	Matlab	Interactive visual	No	No
López, 2016 [8]	Python	Problem-solving	No	Yes
Our Proposed Toolbox	Jupyter/Colab† /PC	Compare operators	Yes	Yes

† Use of special conditions for execution in the Colab environment.

transforms, following the textbook by Gonzalez and Woods [4]. In week seven, students will take Exam 1, and the remaining weeks (eight to ten) will focus on applying these DIP operators to solve real-world computer vision problems. To summarize, the first part of the course aims to provide students with the skills needed to develop DIP operators. In contrast, the second part builds upon this foundation by applying these operators to solve various computer vision problems. Week eleven is reserved for review, and the course concludes with a final exam in week twelve.

The evaluation strategy for this course consists of four individualized and parameterized assignments (15% of the final grade), which focus on the last topic covered in class. These assignments are provided to students before each class and are automatically evaluated using the integration of MCTest, Moodle, and VPL [26, 27]. In addition to these assignments, Exam 1 (30% of the final grade) consists of similar exercises. For the final individual project (15% of the final grade), students must develop a Colab to solve a DIP problem. The final exam (40% of the final grade) is similar to the project. Both the first and final exams must be completed within the two-hour class period.

3.3 Development Environments

Figure 1 provides an overview of the pedagogical intervention employed in the DIP course, utilizing the `morph.py` library, which encompasses 71 methods, including operations (such as minimum, maximum, and negation) and operators (such as erosions and dilations). While the ia870p3 course (refer to Section 2) covered some algorithms from the `mmorph` toolbox, neither that course nor our course covered the entire range of functionalities the toolbox offers. The course material consisted of six conceptual notebooks covered during the initial six weeks. Additionally, there were eight documents focusing on morphological operators. These resources were utilized until Exam 1, marking a significant milestone in the course progression. After Exam 1, the course incorporated a set of 27 notebook demonstrations that showcased the practical application of computer vision in real-world scenarios, adapted from `mmorph`. These notebook documentations are available in `ipynb` format and can be accessed using literate programming tools such as Colab or Jupyter [6]. While the `morph.py` library can also be utilized in a computer console, its primary application is in Programming Exercises (PE). These exercises include automatic correction and are submitted by students in VPL activities on Moodle [26, 27].

The automatic assessment process for this course comprises four customized tasks that assess the most recent topic discussed in class during the first four weeks. These tasks are distributed to students before each class and are automatically evaluated using a combination of MCTest, Moodle, and VPL as reported by Zampirolli et al. [26, 27]. Exam 1 also follows the same process. All methods implemented in `morph.py` are available in the VPL activity runtime files. The student must correctly call these methods or make adaptations to ensure the test case is executed correctly. For example, we can create a new morphological erosion question where the origin of a structuring function b must have one more argument of the method. Note that this simple change has no solution in traditional libraries nor in `morph.py`. So, the students must develop their solution. For each student, it is also possible to draw a different origin for the structuring function b, thus making plagiarism more difficult. During Exam 1, the SEB (Safe Exam Browser – safeexambrowser.org) feature is used, where the student cannot access the internet to try to find ready-made solutions. In the initial lists, even the student who consulted chatbots at the time did not return a correct solution to this question.

4 LEARNING AND DEVELOPING DIP THROUGH `morph.py`

We utilize `morph.py` to gradually introduce the fundamental concepts of DIP programming one by one, allowing students to understand how these concepts combine to create a complete DIP application for ten weeks without feeling overwhelmed. The specific details of this process will be explained in this section, emphasizing the importance of including codes for various methods to improve the reproducibility of this pedagogical approach.

4.1 Initial Part

In week 1, we introduce the initial structure of the `morph.py` library, as illustrated in Figure 2. This command will install the latest version of the Matplotlib (matplotlib.org), OpenCV (opencv.org), and Skimage (scikit-image.org) libraries if they are not already installed.

To use this method in Colab, simply download the `morph.py`, run the cell with the code `from morph import *` and `mm.install()`, as illustrated in Figure 3.

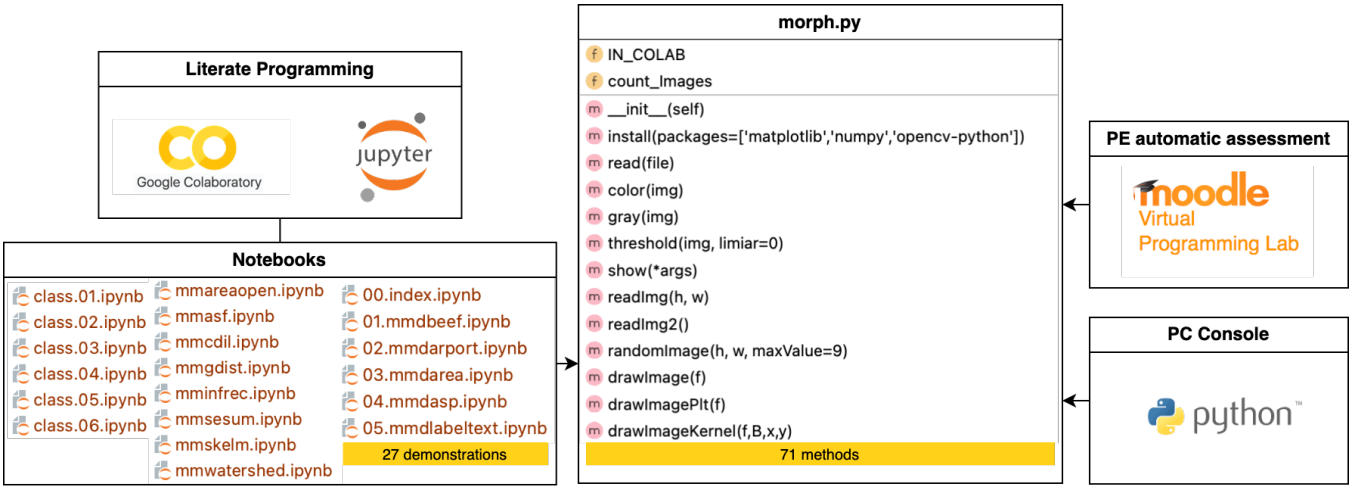

Figure 1: Overview of the pedagogical intervention used.

```python
import matplotlib.pyplot as plt, numpy as np, cv2, requests, sys, subprocess
from PIL import Image; from skimage import io
class mm(object): """ A helper class for image processing tasks. """
  count_Images, IN_COLAB = 0, 'google.colab' in sys.modules # INICIALIZATION
  def install(packages=['matplotlib','scikit-image','opencv-python']):
    """ This function will install the packages
    Input: <packages> list of packages.
    Example:
     mm.install(['matplotlib', 'scikit-image'])
    """
    for p in packages:
        subprocess.check_call([sys.executable, "-m", "pip", "install", p])
```

Figure 2: Initial part of the `morph.py` containing the method for package installation.

```python
# download morph.py from GitHub
# !wget https://raw.githubusercontent.com/fzampirolli/morph/main/morph.py

from morph import *
mm.install()
```

Figure 3: Code for downloading `morph.py` from GitHub.

4.2 Image Reading

Figure 4 shows a method for reading images in different ways. If a Google Drive ID is provided, starting with `'id='` (line 15), or a URL starting with `'http'` (line 17), the image will be read from there. Otherwise, the image will be read from the folder where the code cell is executed (line 20). To use this method, e.g., `img = mm.read('image.png')`.

4.3 Image Format Conversions

After reading an image, it is often necessary to convert it to a desired format. While many image formats are available, we recommend using the RGB format for color images and converting them to this format using the `mm.color()` method in `morph.py` (Figure 5). Additionally, we suggest using images with values between 0 and 255 of type 'uint8' whenever possible. To convert an image to grayscale, use the `mm.gray()` method (Figure 6).

Finally, to convert a grayscale image to binary, please refer to Figure 7 for an example. When the second parameter `threshold>0`

```python
def read(file):
  """ Reads an image from a local file path or URL.
  Input: <str> File path or URL (full or 'id=keyGoogleDrive').
  Output: the read image.
  Example:
   img_local  = mm.read('image.png')
   img_url    = mm.read('https://example.com/image.jpg')
   img_gdrive = mm.read('id=keyGoogleDrive')
  """
  if file.startswith(('http', 'id=')):
      url, pre = '', 'https://drive.google.com/file/d/'
      if pre in file:
          url = 'https://drive.google.com/uc?export=view&id='
          url += file[len(pre):].split('/')[0]
      elif file.startswith('id='):
          url = 'https://drive.google.com/uc?export=view&id=' + file[3:]
      else:
          url = file
      return io.imread(url)
  else:
      return cv2.imread(file)
```

Figure 4: Method for reading an image from a local file path or URL.

```python
def color(img):
  """ Converts an image to RGB color space.
  Input: <numpy.ndarray> Image in BGR, grayscale, or RGBA format.
  Output: RGB image in <numpy.ndarray> format.
  Example:
   img     = mm.read('image.png')
   img_rgb = mm.color(img)
  """
  if len(img.shape) == 2:
    return cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
  elif len(img.shape) == 3 and img.shape[2] == 4:
    return cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)
  elif len(img.shape) == 3 and img.shape[2] == 3:
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  else:
    raise ValueError("Unsupported image format.")
```

Figure 5: Method to convert an image to the RGB color space.

exists in this method, all pixels greater than `threshold` will receive the value 255. Furthermore, if this argument does not exist, as in the comment on line 7, the Otsu method will perform the thresholding [15].

```python
def gray(img):
  """ Converts a color image to grayscale.
  Input: <numpy.ndarray> Input color image.
  Output: grayscale image.
  Example:
   img = mm.read('image.png')
   img_gray = mm.gray(img)
  """
  if len(img.shape) == 3 and img.shape[2] == 4:
    return cv2.cvtColor(img, cv2.COLOR_BGRA2GRAY)
  else:
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

Figure 6: Method to convert an image to the grayscale space.

```python
def threshold(img, threshold=0):
  """ Thresholds an input image by a threshold value or using Otsu's method.
  Input: <numpy.ndarray> Input image to be thresholded.
  Output: <numpy.ndarray> Thresholded image.
  Example:
   img = mm.read('image.png')
   th = mm.threshold(img)
  """
  if threshold == 0:
      value,th=cv2.threshold(img,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
  else:
      value,th=cv2.threshold(img,threshold,255,cv2.THRESH_BINARY)
  return th
```

Figure 7: Method for converting a grayscale image to binary.

Please take note that, for image conversion using this method, it is significantly easier for students to remember the command:

```
th = mm.threshold(img)
```

compared to the command mentioned in line 10 of Figure 7:

```
value,th=cv2.threshold(img,0,255,cv2.THRESH_BINARY
            + cv2.THRESH_OTSU)
```

Furthermore, it's worth highlighting that this method can be further enhanced to accommodate the conversion of various image types and can be integrated with alternative libraries, in addition to OpenCV. This allows for the encapsulation of implementation details, eliminating the need for students to memorize them in order to effectively utilize the method.

4.4 Displaying Multiple Images

Figure 8, adapted from the `mmorph` toolbox, illustrates an important method for displaying multiple images. To use this method, call `mm.show(imgRGB, img1, img2)`, for example. The first image must be in RGB format, while the rest should be binary. The first image is the base, and the other images are overlaid with colors (see lines 11 and 12). Suppose the notebook is not being executed in Colab. In that case, if the student uses Jupyter Notebook or runs the code on a local computer's console, the code after line 18 will save the image to the local computer's local disk.

4.5 Morphological Erosion

In `morph.py`, we conventionally represent morphological operators as follows. First, if we disregard the weights of neighboring pixels in the structuring element `B` of a morphological operator, such as

```python
def show(*args):
  """ This method will draw images f
  Input: <*args> set of images f_i, where i>0 is binary image
  Output: image drawing
  Example:
   f1, f2 = np.zeros((100, 100,3)),  np.zeros((100, 100))
   f2[50:60, 50:60] = 1
   mm.show(f1, f2)
  """
  f = args[0].copy()
  colors = [[255,0,0], [0,255,0], [0,0,255], [255,0,255], [0,255,255],
   [255,255,0], [255,50,50], [50,255,50]] # red, green, blue, cyan, ...
  for i in range(1,len(args)):
      if i >= len(colors):
          break
      f[args[i] > 0] = colors[i-1]
  _ = plt.imshow(f, "gray")
  if not mm.IN_COLAB:
      plt.savefig('fig_' + str(mm.count_Images).zfill(4) + '.png')
      mm.count_Images += 1
```

Figure 8: Method for drawing multiple images, where the first image is RGB.

in erosion [1], we write `mm.ero0(img)`. Refer to Figure 9 for an example. Morphological erosion calculates, for each pixel in the output image, the minimum value among the corresponding pixels in the input image, considering the neighborhood defined by `B` with some origin (usually the center of `B`).

```python
def ero0(f, B=np.ones((3, 3), dtype='uint8')):
  """ Creates an erosion of the input image f by the structuring element B.
  Input: f (ndarray): The input image; B (ndarray, optional): The str. elem.
  Output: The result of the erosion.
  Example:
   mm.ero0(f, B=np.ones((5, 5), dtype='uint8'))
  """
  H, W,  Bh, Bw, g = f.shape, B.shape, f.copy()
  for y in range(H): # Loops over each pixel in the input image.
    for x in range(W):
      for by in range(Bh): # Loops over each neighbor of the current pixel.
        for bx in range(Bw):
          neig_y, neig_x = int(y + by - Bh/2 + 0.5), int(x + bx - Bw/2 + 0.5)
          # Check if the neighbor is within the bounds of image and B.
          if B[by, bx] and 0 <= neig_y < H and 0 <= neig_x < W:
            # Update current pixel with the minimum value of its neighbors.
            if g[y, x] > f[neig_y, neig_x]:
              g[y, x] = f[neig_y, neig_x]
  return g
```

Figure 9: Morphological erosion without weights on neighboring pixels.

Another option to consider is the incorporation of weights on neighboring pixels, achieved through a structuring function `b`. Refer to Figure 10 for an illustration. This modification can be observed in lines 15, 17, and 18 of Figures 9 and 10. In the latter case, we have the difference `- b[by, bx]`. It is crucial to draw the students' attention to this point, as this difference may be less than zero in some instances, which is often undesirable. In such cases, additional post-processing may be necessary to clamp it to zero.

Comparing with the OpenCV erosion usage, shown in Figure 11, it is important to note that these three implementations of erosion do not return exactly the same results, and the student needs to understand the differences between them. Another possibility for a more efficient implementation is to increase the image dimensions and not consider the border pixels in the neighborhood calculations.

```python
def ero1(f, b=np.ones((3, 3), dtype='uint8')):
 """ Creates an erosion of the input image f by the structuring function b.
 Input: f (ndarray): The input image; b (ndarray, optional): The str. func.
 Output: The result of the erosion.
 Example:
  mm.ero1(f, b=np.ones((5, 5), dtype='uint8'))
 """
 H, W,  bh, bw, g = f.shape, b.shape, f.copy()
 for y in range(H): # Loops over each pixel in the input image.
   for x in range(W):
     for by in range(bh): # Loops over each neighbor of the current pixel.
       for bx in range(bw):
         neig_y, neig_x = int(y + by - bh/2 + 0.5), int(x + bx - bw/2 + 0.5)
         # Check if the neighbor is within the bounds of the image.
         if 0 <= neig_y < H and 0 <= neig_x < W: # HERE IS DIFFERENT
           # Update the current pixel with the minimum value of b.
           if g[y, x] > f[neig_y, neig_x] - b[by, bx]: # HERE IS DIFFERENT
             g[y, x] = f[neig_y, neig_x] - b[by, bx]   # HERE IS DIFFERENT
 return g
```

Figure 10: Morphological erosion with weights on neighboring pixels.

Additionally, it is possible to create a highly efficient implementations in ANSI C and then create a Python method that simply calls this implementation, such as the `mmorph` toolbox, generated by AdessoWiki [13]. Furthermore, erosion can also be implemented on a GPU for even better performance [29].

```python
def ero(f, b=np.ones((3, 3), dtype='uint8')):
 """ Creates an erosion of the input image f by the structuring function b.
 Input: f (ndarray): The input image; b (ndarray, optional): The str. func.
 Output: The result of the erosion.
 Example:
  mm.ero(f, b=np.ones((5, 5), dtype='uint8'))
 """
 return cv2.erode(f, b)
```

Figure 11: Morphological erosion of OpenCV.

4.6 A Simple Example of Using the Library `morph.py`

The code to display the image (b) of Figure 13 is shown in Figure 12. The command on line 9, `np.ones((7,7),dtype='uint8')`, defines the structure element and can be replaced with the `mm.sebox(2)` method, which is also implemented in the `morph.py` library. If the code cell shown in Figure 3 has already been executed previously in the Jupyter Notebook or Colab, lines 1 to 5 in Figure 12 do not need to be executed anymore.

```python
# download morph.py from GitHub
# !wget https://raw.githubusercontent.com/fzampirolli/morph/main/morph.py

from morph import *
mm.install()
img = mm.read('https://www.dropbox.com/s/ekjbzp14jt90bfq/00004.jpg?dl=1')
img = img[25:120,30:285] # cropping the image - Figure 13-(a)
th = mm.threshold(mm.gray(img),30)
mm.show(img, th-mm.ero(th, np.ones((7,7), dtype='uint8'))) # Figure 13-(b)
```

Figure 12: Code for downloading `morph.py` from a GitHub key, reading an image from a Dropbox URL, and displaying the image.

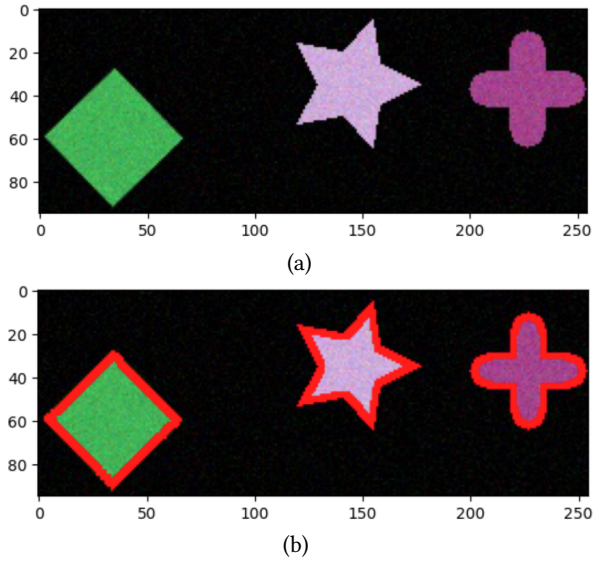

Figure 13: Image (a) shows the cropped portion of the image read in lines 6 and 7 of Figure 12. Image (b) provides an example of using `mm.threshold()` and `mm.show()` in lines 8 and 9 of Figure 12.

5 USING `morph.py` FOR DIP IN EXAMS

In addition to the pedagogical intervention discussed in Section 3.2 and development environments in Section 3.3, we will explain in detail in this section how the `morph.py` library was employed in the two exams conducted in the DIP course.

5.1 Exam 1

Exam 1 was designed parametrically, with the random selection of 3 questions for each student, as detailed in this section.

5.1.1 Question 1. In [28], the process of creating parametric questions was presented, offering a detailed example of a question designed for a programming logic course (CS1) focusing on matrices. This question was adapted for DIP, as illustrated in Figure 14.

The blue-colored text has been added to emphasize the parametric part, which varies for each generated exam and is removed in the PDF version that will be printed and delivered individually to the student. The student is required to solve this question in a Moodle VPL activity. This question can be easily adapted for DIP and was used in Exam 1, where the student should use erosion or dilation operators with a 3×3 neighborhood. For this specific case of **North lesser**, the student should use the structuring element `B = np.array([[1,1,1],[0,1,0],[0,0,0]])` along with `ero0(f,B)` (adapting code shown in Figure 9 and without considering the border elements). The input matrix/image, Figure 14-(left), is provided as input data in the test cases, and (right) represents the output.

To read this input image, there is also specific code in the `morph.py` library, `readImg2()`, which can be used when the dimensions of the image are not known, as shown in Figure 15. The `readImg(width, height)` method is also available when the image dimensions are known. This code reads input lines containing

1. An entry $a_0 = (i, j)$ of a matrix is called **North lesser** if its value is lesser than the neighbouring ones positioned at a_8, a_1, a_2, as indicated in the table. See in figure an example matrix/image f with dimensions 9×17 and its corresponding **North lesser**, where "-" means "-1", without considering the border elements.

a_8 = Northwest	a_1 = North	a_2 = Northeast
a_7 = West	$a_0 = (i, j)$	a_3 = East
a_6 = Southwest	a_5 = South	a_4 = Southeast

It is possible to solve this problem using the dilation or erosion operators defined below:

$$dil(f, B)(x) = (f \oplus B)(x) = \delta_B(f)(x) = \max\{f(y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

$$ero(f, B)(x) = (f \ominus B)(x) = \varepsilon_B(f)(x) = \min\{f(y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

$\forall x \in \mathbb{E}$, f $\in K^{\mathbb{E}}$ or $f \in [0, k]^{\mathbb{E}}$, k is a positive integer representing the maximum number of grayscale levels in the set $[0, k]$ for the digital image f defined over the domain $\mathbb{E}$. Also, consider $\mathbb{B} \subset \mathbb{Z} \times \mathbb{Z}$ as the structuring element (neighborhood/kernel), with its origin at its center.

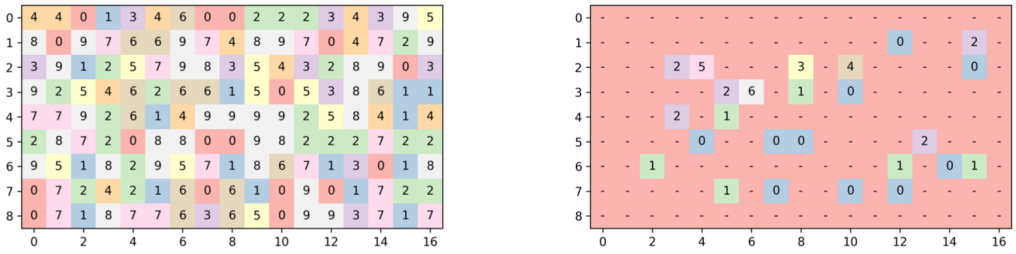

Figure 14: Question 1 on Exam 1 involves using adapted erosion or dilation.

pixel values, splits each line into individual pixel values, converts them to integers, and stores them in a list. It continues this process until there is no more input (the last line is empty). Finally, it converts the list into a NumPy array with an unsigned 8-bit integer data type and returns the resulting array.

```
def readImg2():
  """ This function reads an image of varying size from standard input.
  Example:
    f = mm.readImg2()
    255   0  255
    128  64  192
    0   192  128

  """
  f = []
  read_row = input()
  while read_row: # Read each line of the input until there is no more input
    # Split the line into individual pixel values and convert them to int.
    row = [int(i) for i in read_row.split() if i]
    f.append(row) # Add the row to the list of rows.
    read_row = input()
  return np.array(f).astype('uint8')
```

Figure 15: Read a variable size image.

Even if this question is presented as an activity to be performed with internet research, the student will not find a ready-made solution. For example, when asking about generative models like ChatGPT (chat.openai.com). Additionally, this question has 16 variations, including variations in the values and dimensions of the input image. There are eight variations for the cardinal directions, with two variations each for greater or lesser. Creating even more significant variations is possible using a neighborhood of 5×5 or 7×7.

5.1.2 Question 2. The second question of Exam 1 instructed the student to read an input image `f` and apply a filter from the OpenCV library. Such a parametric question was created for each of the filters:

(1) `cv2.medianBlur(f, bw)`,
(2) `cv2.filter2D(f, -1, bw)`, and
(3) `cv2.GaussianBlur(f, (bw, bh), 0)`.

Each student received a random question. For example, for the filter `cv2.medianBlur(f, bw)`, we randomly assigned each student a neighborhood `bw` of size 3, 5, or 7. We generated a random input image, as Figure 16 illustrates.

5.1.3 Question 3. The third question on Exam 1 required the student to calculate the distance transform (DT – shortest distance to a pixel with a value of zero) using successive erosions, as illustrated in Figure 17. Each student is assigned a different image f and neighborhood b, which can vary significantly. Furthermore, multiple test cases are generated for each student, with the randomly assigned neighborhood b determining the type of distance used. For the 2D case, various distance types are available, including Euclidean, City-Block, and Chessboard. [2]. In some variations of this question, students may also be asked to print the number of erosions performed until convergence to the DT.

These three questions presented in Exam 1 assess students' abilities and competencies in creating operators for DIP. Although students have access to the `morph.py` library during the assessment, they will need to make adaptations to ensure their solutions pass various test cases generated automatically for the Moodle VPL activity, following the method defined in [27].

1. The convolution of an image f with a kernel b ($bw \times bh$) is defined as follows:

$$(f * b)[x, y] = \sum_{m=0}^{bw-1} \sum_{n=0}^{bh-1} f[x + m - bw//2, y + n - bh//2]b[m, n]$$

Calculate the convolution using `cv2.medianBlur(f, 5)` (b has dimensions 5×5), as shown in the example below:

Note: The 2D dimensions of f change in each test case. The left side shows the input image, and the right side shows the output image.

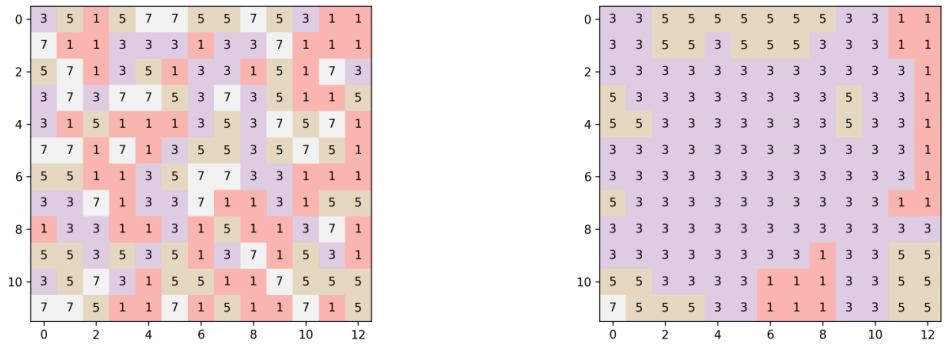

Figure 16: Question 2 on Exam 1 involves using the `cv2.medianBlur()` filter.

1. Implement the **1D Distance Transform (DT) using successive erosions**, considering erosion defined as:

$$ero(f, b)(x) = (f \ominus b)(x) = \varepsilon_b(f)(x) = \min\{f(y) - b(x - y) : y \in \mathbb{B}_x \cap \mathbb{E}\}$$

where $x \in \mathbb{E} \subset \mathbb{Z}$, $f \in K^{\mathbb{E}}$ or $f \in [0, k]^{\mathbb{E}}$, k is a positive integer representing the grayscale levels of the digital image with domain $\mathbb{E}$. Also, consider $b \in \mathbb{Z}^{\mathbb{B}}$ and $\mathbb{B} \subset \mathbb{Z}$, as the structuring function (neighborhood/kernel), with its center being the origin (in this example, the image f has 51 pixels). Implement the following DT algorithm, as shown in the example below:

$DT : K^{\mathbb{E}} \times \mathbb{Z}^{B} \to K^{\mathbb{E}}$
$\quad DT(f, b) = g$

$g = f;$
$i = 0;$
$\forall g \neq \varepsilon_b(g)$
$\quad g = \varepsilon_b(g);$
$\quad i++;$

Example inputs (b and f):

```
-4 -3 -2 0 -2 -3 -4
9 9 0 0 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 9 9 0 9 9 9
```

Example of the corresponding output ($DT(f, b)$):

```
3 2 0 0 2 3 4 6 7 8 7 6 4 3 2 0 2 3 4 6 7 8 7 6 4 3 2 0 2 3 4 6 7 8 8 7 6 4 3 2 0 2 3 4 4 3 2 0 2 3 4
```

Figure 17: Question 3 on Exam 1 for 1D Distance Transform (DT) using successive erosions.

5.2 Exam 2

Following Exam 1, students are expected to gain proficiency in utilizing the methods offered by the `morph.py` library, and other DIP libraries. Consequently, in the second part of the course features several practical examples of problem-solving using DIP, building upon concepts introduced in previous sections and evaluated in the final exam.

In the final exam, students were asked to detect nine classes, encompassing 27 randomly generated geometric objects based on

their color, size, and rotation. Figure 13-(a) shows an image displaying a selection of three objects. The URL below was presented in a student's question prompt:

https://www.dropbox.com/s/ekjbzp14jt90bfq/00004.jpg

These were the nine classes of random objects:

```
objects=['Triangle', 'Square', 'Pentagon', 'Hexagon',
  'Heptagon', 'Circle', 'Ellipse', 'Star', 'Cross']
```

In addition to this image, a TXT file was provided containing each object's class and bounding box on separate lines. Here's an example:

https://www.dropbox.com/s/s5z2muo0bx4calx/00004.txt

The following is a TXT file snippet showing classes 0, 1, and 2, which correspond to Triangle, Square, and Pentagon, respectively. In the first column is the class, and in the subsequent columns are the bounding box values represented as a percentage of width (columns 2 and 4) and height (columns 3 and 5):

```
0 0.2895 0.2007 0.3750 0.2862
0 0.0938 0.6678 0.1595 0.7336
0 0.4934 0.3076 0.5954 0.4095
1 0.0543 0.0872 0.1595 0.1924
1 0.3421 0.7911 0.4079 0.8569
1 0.2664 0.4424 0.3322 0.5082
2 0.4211 0.6661 0.5263 0.7714
2 0.5757 0.4688 0.6678 0.5609
2 0.0938 0.8355 0.2122 0.9539
```

Each student received a distinct image and a corresponding TXT file and was tasked with assessing the accuracy of their detection method. The question was set up with automatic corrections in the VPL activity, including various test cases, which means several pairs of images and corresponding TXT files containing bounding boxes. During the classes, it was mentioned that the datasets used to train convolutional neural networks often consist of pairs of images and accompanying TXT files, as seen in popular frameworks like YOLO [5].

6 RESULTS AND DISCUSSION

We conducted one voluntary survey to gather data from students in our DIP course. The survey was administered after the final exam and included 5 Likert-based questions to gather feedback on the course content and strategies.

For the students who completed the course, the survey received 15 responses ($\approx$ 60%) for the Likert questions (Table 2). We analyzed these responses and found that questions $Q2$ and $Q3$ resemble a normal distribution. This was determined by considering the skewness or kurtosis $z-values$ falling between -2.575 and 2.575 for 99% confidence level (CL) [18]. Therefore, we used a parametric method ($t-test$) for these two questions and an equivalent non-parametric method ($Wilcoxon\ Signed-Rank$) for the remaining. Additionally, we applied Cronbach's Alpha to assess the internal consistency of the Likert data, and the obtained value was .90. Finally, we tested the hypothesis below.

Table 2: Post-class Likert questions

Id	Question
1	Does the `morph.py` library contributed to your DIP's learning process by allowing you to focus on the problem to be solved?
2	Does the `morph.py` library motivated you to learn DIP?
3	Have you used `morph.py` through all course tasks?
4	Do you suggest more implementations into `morph.py`?
5	Do you think that `morph.py` was easier to use and learn than openCV?

Table 3: $Wilcoxon\ Signed-Rank$ analysis for questions $Q1$, $Q4$ and $Q5$

Q	T	$Median$	$z-score$	Effect Size(ES)	Power(CL=99%)
$Q1$	120	5	$< .01$	.93	.82
$Q4$	96	5	$< .01$	.74	.61
$Q5$	118	5	$< .01$	.87	.76

Table 4: $t-test$ analysis for all questions for questions $Q2$ and $Q3$

Q	t	df	$p-value$	Std. Deviation
	Mean	Median	Effect Size (ES)	Power(CL=99%)
$Q2$	7.67	14	$< .01$	.81
	4.56	5	1.92	.99
$Q3$	10.48	14	$< .01$	.61
	4.62	5	2.62	1

H_0 : the `morph.py` library had a neutral effect on students' learning, $mdn \leq 3$.

H_1 : the `morph.py` library positively affected students' learning, $mdn > 3$.

Table 3 presents the results of assessing students' perception of our library's value in their DIP learning process. The findings demonstrate significant differences between all the questions and the null hypothesis (H_0). For example, the students' positive perception of our library ($Q1$) indicates that the alternative hypothesis (H_1) holds statistical significance based on the one-group $Wilcoxon\ Signed-Rank$ test, $N = 15$, T=120, $p < 0.01$, ES=large (.82). This result suggests that our library facilitated students to explore the different pre-implemented algorithms without implementing them from scratch. Consequently, they could focus more on understanding the underlying concepts and techniques behind the algorithms, thereby reducing the initial learning curve. This finding aligns with previous studies [21, 25].

The results of a one-group t-test, measuring students' utilization of our library throughout the course, are presented in Table 4. Notably, the students' motivation scores ($Q2$) indicate that the alternative hypothesis (H_1) holds statistical significance, with a median score of 5, $t(14) = 7.67$, $p < 0.01$, ES=large (.99). This result aligns with the findings about $Q1$ and $Q3$, indicating the availability of

high-level functions provided by `morph.py` significantly simplified the implementation of complex image processing operations. Consequently, students could concentrate more on the practical aspects of image processing, as was also suggested by [17, 21].

Finally, out of the 25 students who completed the course, only 4 did not achieve approval, resulting in an 84% approval rate.

Threats to validity

Our study encounters challenges to its validity. Given that DIP is an elective course in our university primarily chosen by students nearing graduation, the results we have presented remain inconclusive, prompting the need for further experimentation. Some students enroll in multiple courses, often leaning toward those requiring less effort, which contributes to a lower course completion rate.

Moreover, conducting experiments involving both test and control groups necessitates approval from the ethics committee. Additionally, it requires the establishment of multiple classes, all taught by the same professor, with consistent learning conditions, scheduling within the same academic term, and a commitment to ensuring comparable levels of dedication. This undertaking poses logistical challenges at our university, where typically only one class is offered per year (since the founding of our university in 2006, the DIP has been offered in only ten classes, mainly after 2011).

Therefore, the detailed exposition of our teaching-learning-evaluation method in this article paves the way for future research to explore new avenues for experimentation.

7 CONCLUSIONS AND FUTURE WORKS

This paper presents an interactive course instructing novice learners in Digital Image Processing (DIP) using Python's `morph.py` library. The course is delivered through Google Colab and VPL activities on Moodle, offering practical examples and exercises to reinforce fundamental concepts and operators. The course encompasses essential topics, including image representation, sampling, quantization, and perception, and advanced subjects like transformations, enhancements, restoration, segmentation, feature extraction, object detection, and machine learning for computer vision.

The course has effectively addressed the challenges associated with instructing Digital Image Processing (DIP) beginners. Students have consistently appreciated the course's practical and interactive nature. By leveraging the `morph.py` library and engaging in hands-on exercises, students have successfully bridged the gap between theoretical knowledge and practical application, resulting in a deeper and more comprehensive understanding of DIP principles.

In the future, the course has the potential for expansion into more advanced topics in DIP, providing students with a broader skill set by adding more libraries and frameworks. To further enrich the learning experience, incorporating real-life case studies and projects can enhance students' problem-solving abilities and improve comprehension of practical DIP applications.

DATA AVAILABILITY STATEMENT

The data supporting the findings of this study are available at the following URL: github.com/fzampirolli/morph, and are open-source under the MIT License.

REFERENCES

[1] Gerald Jean Francis Banon and Junior Barrera. 1994. *Bases da Morfologia Matemática para a análise de imagens binárias.* UFPE-DI.

[2] Olivier Cuisenaire. 1999. *Distance transformations: fast algorithms and applications to medical image processing.* Technical Report.

[3] Edward R. Dougherty and Roberto de Alencar Lotufo. 2003. *Hands-on morphological image processing.* Vol. 59. SPIE press.

[4] Rafael C Gonzalez and Richard C Woods. 2009. *Processamento digital de imagens.* Pearson Educación.

[5] Muhammad Hussain. 2023. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines* 11, 7, 677.

[6] Donald Ervin Knuth. 1984. Literate programming. *Comput. J.* 27, 2, 97–111.

[7] Konstantinos Konstantinides and John R Rasure. 1994. The Khoros software development environment for image and signal processing. *IEEE Transactions on Image Processing* 3, 3, 243–252.

[8] Andrés Fernando Jiménez López, Marla Carolina Prieto Pelayo, and Ángela Ramírez Forero. 2016. Teaching image processing in engineering using python. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* 11, 3, 129–136.

[9] Roberto de Alencar Lotufo. 2017. Set of functions used in the course IA898 - Digital Image Processing. https://github.com/robertoalotufo/ia898. GitHub repository.

[10] Roberto de Alencar Lotufo. 2019. Set of functions used in the course IA870 - Mathematical Morphology. https://github.com/robertoalotufo/ia870p3. GitHub repository.

[11] Roberto de Alencar Lotufo, Francisco de Assis Zampirolli, Roberto Hirata Junior, and Junior Barrera. 1997. MMachLib functions and MMach operators. In *Brazilian Workshop'97 on Mathematical Morphology.*

[12] Rubens C Machado, Roberto de Alencar Lotufo, Alexandre G Silva, and André Vital Saúde. 2003. ADESSO. Scientific Software Development Environment. *Journal of Computer Science and Technology* 3, 01, 1–6.

[13] Rubens C. Machado, Leticia Rittner, and Roberto de Alencar Lotufo. 2011. Adessowiki-Collaborative platform for writing executable papers. *Procedia Computer Science* 4, 759–767.

[14] MathWorks. 2024. *MathWorks Image Processing Toolbox.* https://www.mathworks.com/products/image.html Acessado em 8 de janeiro de 2024.

[15] Nobuyuki Otsu. 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics* 9, 1, 62–66.

[16] Juan Carlos Rodríguez del Pino, Enrique Rubio Royo, and Zenón José Hernández Figueroa. 2010. VPL: laboratorio virtual de programación para Moodle. In *Jornadas de Enseñanza Universitaria de la Informática.* 429–435.

[17] Penny M Rowe and et al. 2018. Teaching image processing in an upper level CS undergraduate class using compuational guided inquiry and polar data. *Journal of computing sciences in colleges* 34, 1.

[18] Thomas P Ryan. 2013. *Sample size determination and power.* John Wiley & Sons.

[19] Daniel Sage and Michael Unser. 2001. Easy Java programming for teaching image-processing. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, Vol. 3. IEEE, 298–301.

[20] Daniel Sage and Michael Unser. 2003. Teaching image-processing programming in Java. *IEEE Signal Processing Magazine* 20, 6, 43–52.

[21] Daniel Sage and Michael Unser. 2003. Teaching image-processing programming in Java. *IEEE Signal Processing Magazine* 20, 6, 43–52.

[22] A.G. Silva and et al. 2003. Toolbox of image processing using the Python language. In *Proc. Int. Conf. on Image Processing*, Vol. 3. III–1049.

[23] Gloria Bueno García Oscar Deniz Suarez. 2013. *Learning image processing with OpenCV.*

[24] Ali Abdullah Yahya. 2019. Teaching Digital Image Processing Topics via Matlab Techniques. *International Journal of Information and Education Technology* 9, 10, 729–734.

[25] Ziv Yaniv and et al. 2018. SimpleITK image-analysis notebooks: a collaborative environment for education and reproducible research. *Journal of digital imaging* 31, 3, 290–303.

[26] F.A. Zampirolli, P.H. Pisani, J.M. Josko, G. Kobayashi, F. Fraga, D. Goya, and H.R. Savegnago. 2020. Parameterized and automated assessment on an introductory programming course. In *Anais do XXXI SBIE.* SBC, 1573–1582.

[27] F.A. Zampirolli, Cristiane M. Sato, Heitor Rodrigues Savegnago, Valério Ramos Batista, and Guiou Kobayashi. 2021. Automated assessment of parametric programming in a large-scale course. In *2021 XVI Latin American Conference on Learning Technologies (LACLO).* 357–363.

[28] F.A. Zampirolli, F. Teubl, and V.R. Batista. 2019. Online Generator and Corrector of Parametric Questions in Hard Copy Useful for the Elaboration of Thousands of Individualized Exams.. In *CSEDU.* 352–359.

[29] F. A. Zampirolli and Leonardo Filipe. 2017. A fast CUDA-based implementation for the Euclidean distance transform. In *2017 International Conference on High Performance Computing & Simulation (HPCS).* IEEE, 815–818.