### Contents

1	Basic Test Results	2
2	m ex3/README.md	4
3	ex3/answer q1.txt	5
4	ex3/answer q2.txt	6
5	ex3/answer q3.txt	7
6	ex3/sol3.py	8
7	ex3/externals/final dolphin.jpg	15
8	ex3/externals/final lafa.jpg	16
9	ex3/externals/final mask dolphin.jpg	18
10	ex3/externals/final mask refa.png	19
11	ex3/externals/final refa.jpg	21
<b>12</b>	ex3/externals/final water.jpg	23

#### 1 Basic Test Results

```
ex3/
1
    ex3/README.md
   ex3/answer_q1.txt
    ex3/answer_q2.txt
4
    ex3/answer_q3.txt
    ex3/externals/
    ex3/externals/final_dolphin.jpg
    ex3/externals/final_lafa.jpg
    ex3/externals/final_mask_dolphin.jpg
    ex3/externals/final_mask_refa.png
    ex3/externals/final_refa.jpg
11
    ex3/externals/final_water.jpg
12
    ex3/sol3.py
    Ex3 Presubmission Script
14
    _____
15
16
17
18
        Disclaimer
19
        The purpose of this script is to make sure that your code is compliant
20
21
        with the exercise API and some of the requirements
        The script does not test the quality of your results.
22
23
        Don't assume that passing this script will guarantee that you will get
24
        a high grade in the exercise
25
26
    === Check Submission ===
27
    README file:
28
29
    # ex3-yishai.hazi
30
31
    sol3.py
    externals:
    final_dolphin.jpg
33
34
    final_mask_dolphin.jpg
   final_water.jpg
35
   final_refa.jpg
36
37
    final_mask_refa.png
   final_lafa.jpg
38
39
    answer_q1.txt
40
    answer_q2.txt
    answer d3.txt
41
42
    README.md
43
44
45
    === Answers to questions ===
46
47
    By multiplying each level with a different value - we can control of "how much" we take from the frequencies.
    For example, if we multiply the higher levels of the pyramids with small value, it means less high frequencies
49
50
    in the final image.
51
    Answer to Q2:
52
53
    It looks like that if the filter size is bigger, it causes the blended image to be blur, thats probably because
   at each reduction (in the pyramid) we blur a larger range of pixels.
54
55
   More levels that the pyramid has, results in a smoother and accurate image. The reason for that is that
57
   each level "merge" different set of frequencies, so if the pyramid has more levels, more frequencies are merged. (the higher
    === Load Student Library ===
```

```
60
     Loading...
 61
 62
     === Section 3.1 ===
 63
 64
     Trying to build Gaussian pyramid...
 65
         Passed!
 66
     Checking Gaussian pyramid type and structure...
 67
 68
         Passed!
     Trying to build Laplacian pyramid...
 69
 70
         Passed!
 71
     Checking Laplacian pyramid type and structure...
         Passed!
 72
 73
 74
     === Section 3.2 ===
 75
 76
     Trying to build Laplacian pyramid...
 77
     Trying to reconstruct image from pyramid... (we are not checking for quality!)
 78
         Passed!
 79
     Checking reconstructed image type and structure...
 80
 81
         Passed!
 82
     === Section 3.3 ===
 83
 84
     Trying to build Gaussian pyramid...
 85
         Passed!
 86
 87
     Trying to render pyramid to image...
         Passed!
 88
 89
     Checking structure of returned image...
 90
     Trying to display image... (if DISPLAY env var not set, assumes running w/o screen)
 91
 92
         Passed!
 93
     === Section 4 ===
 94
 95
     Trying to blend two images... (we are not checking the quality!)
 96
 97
         Passed!
     Checking size of blended image...
 98
         Passed!
99
     Tring to call blending_example1()...
100
         Passed!
101
     Checking types of returned results...
102
103
     Tring to call blending_example2()...
104
     /tmp/bodek.woFmwe/impr/ex3/yishai.hazi/gitsub/testdir/test:8: DeprecationWarning: `imread` is deprecated!
105
106
      `imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
     Use ``imageio.imread`` instead.
107
       im = imread(filename)
108
109
     Checking types of returned results...
110
111
         Passed!
112
113
     === Presubmission Completed Successfully ===
114
115
116
         Please go over the output and verify that there were no failures / warnings.
117
          Remember that this script tested only some basic technical aspects of your implementation.
118
119
          It is your responsibility to make sure your results are actually correct and not only
120
         technically valid.
```

### 2 ex3/README.md

- # ex3-yishai.hazi

- # ex3-yishai.hazi
  sol3.py
  externals:
  final\_dolphin.jpg
  final\_mask\_dolphin.jpg
  final\_water.jpg
  final\_refa.jpg
  final\_refa.png
  final\_lafa.jpg
  answer\_q1.txt
  answer\_q2.txt
  README.md

## 3 ex3/answer q1.txt

- 1 By multiplying each level with a different value we can control of "how much" we take from the frequencies.
- For example, if we multiply the higher levels of the pyramids with small value, it means less high frequencies in the final image.

## 4 ex3/answer q2.txt

- It looks like that if the filter size is bigger, it causes the blended image to be blur, thats probably because at each reduction (in the pyramid) we blur a larger range of pixels.

## 5 ex3/answer q3.txt

- More levels that the pyramid has, results in a smoother and accurate image. The reason for that is that each level "merge" different set of frequencies, so if the pyramid has more levels, more frequencies are merged. (the higher

#### 6 ex3/sol3.py

```
import os
    import numpy as np
    import skimage.color as color
    import imageio
    import scipy.ndimage.filters
    import matplotlib.pyplot as plt
9
    def normalize(image):
10
11
        A function that normalize the elements inside
        a numpy matrix, and change their type to float64.
12
        :param image: a numpy matrix
13
         :return: the matrix with all elements normalized
         to the range: [0, 1].
15
16
        image = image.astype('float64')
17
        return image / 255
18
19
20
    def read_image(filename, representation):
21
22
        A function that reads a file according
23
24
        to a given representation.
25
         :param filename: The file name.
26
         :param representation: 1 represents rgb,
27
        2 represents gray_scale.
        :return: An image according to the
         given\ representation.
29
30
31
        image = imageio.imread(filename)
        if representation == 2 or image.ndim != 3:
32
            if np.any(image > 1):
                return normalize(image)
34
35
         elif representation == 1:
            return color.rgb2gray(image)
36
37
38
    def normalize_gaussian_filter(filter_vec):
39
40
41
        A function that normalize the gaussian
        filter.
42
43
        :param filter_vec: The filter.
         :return: the normalized filter.
45
46
        return filter_vec / np.sum(filter_vec)
47
48
    def build_gaussian_filter(filter_size):
^{49}
50
         A function that build a gaussian filter
51
        according to a given size.
        :param filter_size: odd integer that
53
54
         represents the desired size.
        :return: The gaussian filter as
55
        a numpy array with shape: [1, filter_size].
56
57
         (without normalization)
58
        number_of = filter_size - 2
59
```

```
60
          if filter_size > 1:
              small_kernel = np.array([1, 1])
 61
 62
          else:
              small_kernel = np.array([1])
 63
          the_filter = small_kernel
 64
 65
          for i in range(number_of):
             the_filter = np.convolve(the_filter, small_kernel)
 66
          return np.reshape(the_filter, (1, filter_size))
 67
 68
 69
     def convolve(im, filter_vec):
 70
 71
          A function that convolve an image with
 72
 73
          a given filter.
 74
          :param im: The given image as a grayscale
          image with double values in [0, 1].
 75
 76
          :param filter_vec: The filter as a row vector.
          :return: The image after the deletion.
 77
 78
          im_after_rows_convolution = scipy.ndimage.filters.convolve(
 79
             im, filter_vec)
 80
 81
          return scipy.ndimage.filters.convolve(
              im_after_rows_convolution, np.transpose(filter_vec))
 82
 83
 84
 85
     def delete_elements(im):
 86
 87
          A function that delete all the elements
          in the odd indexes. (as well as all the
 88
 89
          odd rows)
 90
          :param im: The given image.
          :return: The image after the deletion.
 91
 92
          odd_indexes_of_rows = np.arange(1, im.shape[0] + 1, 2)
odd_indexes_of_cols = np.arange(1, im.shape[1] + 1, 2)
 93
 94
 95
          im_after_horizontal_clean = np.delete(im,
 96
                                                   odd_indexes_of_rows, axis=0)
 97
          im_after_vertical_clean = np.delete(im_after_horizontal_clean,
                                                 odd_indexes_of_cols,
 98
                                                 axis=1)
 99
100
          return im_after_vertical_clean
101
102
103
      def reduce(im, filter_vec):
104
          A function that creates a smaller image
105
106
          (smaller by factor of 2) of the given image.
          :param im: The given image.
107
108
          : param\ filter\_vec\colon\ \textit{The given filter}
109
          that blur the image.
          :return: The smaller image.
110
111
112
          im_after_convolution = convolve(im, filter_vec)
113
          return delete_elements(im_after_convolution)
114
115
     def build_gaussian_pyramid(im, max_levels, filter_size):
116
117
          A function that construct a Gaussian
118
119
          pyramid of a given image.
120
          :param im: A grayscale image with double
          values in [0, 1].
121
          :param max_levels: The maximal number of levels1
122
123
          in the resulting puramid.
          :param filter_size: The size of the Gaussian filter
124
          (an odd scalar that represents a squared filter) to be used
125
          in constructing the pyramid filter.
126
127
          : return: \ \textit{The resulting pyramid pyr as a standard python array},
```

```
128
          and filter\_vec - which is a row vector of shape (1, filter\_size).
129
130
         filter_vec = build_gaussian_filter(filter_size)
          filter_vec = normalize_gaussian_filter(filter_vec)
131
         pyr = [im]
132
133
          smallest_image = im
134
          max_levels -= 1
          while max_levels >= 1 and smallest_image.shape[0] > 16 \setminus
135
136
                  and smallest_image.shape[1] > 16:
              smallest_image = reduce(smallest_image, filter_vec)
137
              pyr.append(smallest_image)
138
139
              max_levels -= 1
          return pyr, filter_vec
140
141
142
     def zero_pad(im):
143
144
          A function that expand an image by
145
         adding zeros in the odd indexes (and odd
146
147
          rows)
          :param im: The given image as a grayscale
148
149
          image with double values in [0, 1].
          :return: The image after extension.
150
151
152
          odd_indexes_in_rows = np.arange(1, im.shape[0] + 1)
          odd_indexes_in_cols = np.arange(1, im.shape[1] + 1)
153
          im = np.insert(im, odd_indexes_in_rows, 0, axis=0)
154
155
          im = np.insert(im, odd_indexes_in_cols, 0, axis=1)
         return im
156
157
158
     def expand(im, filter_vec):
159
160
161
          A function that expand an image by
          adding zeros to it and then blur the
162
163
          result with the given filter.
164
          :param im: The given image as a grayscale
          image with double values in [0, 1].
165
          :param filter_vec:
166
          :return: The given filter we use for blur.
167
168
          (a row vector of shape (1, filter_size).)
169
170
         im = zero_pad(im)
171
          im_after_convolve = convolve(im, filter_vec)
          return im_after_convolve
172
173
174
     def build_laplacian_pyramid(im, max_levels, filter_size):
175
176
177
          A function that construct a Laplacian pyramid
          of a given image.
178
179
          :param im: A grayscale image with double
180
          values in [0, 1].
181
          : param\ max\_levels \colon\ The\ maximal\ number\ of\ levels 1
          in the resulting pyramid.
182
          :param filter_size: The size of the Gaussian filter
183
184
          (an odd scalar that represents a squared filter) to be used
185
          in constructing the pyramid filter.
          :return:The resulting pyramid pyr as a standard python array,
186
187
          and filter\_vec - which is row vector of shape (1, filter\_size).
188
189
          gaussian_pyr, filter_vec = build_gaussian_pyramid(im, max_levels,
190
                                                              filter size)
         pyr = []
191
          for i in range(len(gaussian_pyr) - 1):
192
193
              pyr.append(gaussian_pyr[i] - expand(gaussian_pyr[i + 1],
                                                    filter_vec * 2))
194
195
         pyr.append(gaussian_pyr[-1])
```

```
196
         return pyr, filter_vec
197
198
     def laplacian_to_image(lpyr, filter_vec, coeff):
199
200
201
          A function that construct an image from its
          Laplacian Pyramid.
202
          :param lpyr: The Laplacian pyramid
203
204
          : param\ filter\_vec\colon \textit{The filter that is generated}
          by the function: "build_laplacian_pyramid".
205
          :param coeff: A python list. Each level i of the
206
207
          laplacian pyramid is multiplied by its corresponding
208
          coefficient coeff[i].
209
          :return: The original image.
210
          for i in range(len(lpyr)):
211
212
             lpyr[i] = lpyr[i] * coeff[i]
          image = lpyr[-1]
213
          for i in range(len(lpyr) - 1, 0, -1):
214
215
              image = expand(image, filter_vec * 2) + lpyr[i - 1]
          return image
216
217
218
219
     def stretch_image(image):
220
221
          A function that stretch the values of
          the given image to the range: [0, 1].
222
223
          :param image: The given image.
224
          :param min_value: A given minimum.
225
          :param max_value: A given maximum.
226
          :return: The stretched image.
227
          return (image - np.min(image)) / (np.max(image) - np.min(image))
228
229
230
231
     def render_pyramid(pyr, levels):
232
233
          A function that creates a single black image
          in which the pyramid levels of the given pyramid
234
          pyr are stacked horizontally.
235
236
          :param pyr: The given pyramid (a Gaussian
237
          or Laplacian pyramid)
          :param levels: the number of levels of the pyramid.
238
239
          :return: A black image in which the pyramid levels of the
          given pyramid pyr are stacked horizontally.
240
241
242
          res = stretch_image(pyr[0])
         for i in range(1, min(levels, len(pyr))):
243
              black_image = [[0] * pyr[i].shape[1]] * pyr[i].shape[0] * (2 ** i - 1)
244
              stretched_image = stretch_image(pyr[i])
245
              smaller_image_with_black_part = np.vstack(
246
247
                  (stretched_image, black_image))
248
              res = np.hstack((res, smaller_image_with_black_part))
249
          return res
250
251
252
     def display_pyramid(pyr, levels):
253
          A function that use render_pyramid to
254
255
          internally render and then display
256
          the stacked pyramid image.
257
          :param pyr: A Gaussian or Laplacian pyramid.
258
          :param levels: The number of levels in pyr.
          :return: None.
259
260
          image = render_pyramid(pyr, levels)
261
          plt.imshow(image, cmap='gray')
262
263
         plt.show()
```

```
264
265
266
     def blend(pyr_im1, pyr_im2, pyr_mask):
267
          A function that blends 2 Laplasian pyramids
268
269
          according to a mask pyramid.
          :param pyr_im1: First Laplasian pyramid.
270
          :param pyr_im2: Second Laplasian pyramid.
271
272
          :param pyr_mask: Mask pyramid.
          :return: The blended pyramid.
273
274
275
          blended_pyr = []
276
          for i in range(len(pyr_mask)):
              blended_pyr.append(pyr_mask[i] * pyr_im1[i] +
277
278
                                  (1 - pyr_mask[i]) * pyr_im2[i])
          return blended_pyr
279
280
281
     def pyramid_blending(im1, im2, mask, max_levels, filter_size_im,
282
                            filter_size_mask):
283
284
          A function that do a pyramid blending as described in the lecture.
285
          :param im1: First image to be blended.
286
287
          :param im2: Second image to be blended.
288
          :param mask: A boolean mask containing True and False representing
289
          which parts of im1 and im2 should appear in the resulting im_blend.
          :param\ max\_levels\colon \textit{The max\_levels parameter when we generating the}
290
291
          Gaussian and Laplacian pyramids.
          :param filter_size_im: The size of the Gaussian filter
292
293
          (an odd scalar that represents a squared filter) which defining the
294
          filter used in the construction of the Laplacian pyramids of im1 and im2.
          param filter_size_mask: The size of the Gaussian filter:
295
296
          (an odd scalar that represents a squared filter) which defining the
297
          filter used in the construction of the Gaussian pyramid of mask.
          :return: The blended image as a valid gray scale image in the range [0, 1].
298
299
300
          pyr_im1, filter_vec = build_laplacian_pyramid(im1, max_levels,
301
302
                                                          filter_size_im)
          pyr_im2, filter_vec = build_laplacian_pyramid(im2, max_levels,
303
304
                                                          filter_size_im)
305
          pyr_mask, filter_vec_mask = build_gaussian_pyramid(mask.astype(np.float64),
306
                                                               max levels.
307
                                                               filter_size_mask)
308
          blended_pyr = blend(pyr_im1, pyr_im2, pyr_mask)
309
310
          im_blend = laplacian_to_image(blended_pyr, filter_vec,
                                         [1] * len(blended_pyr))
311
          return stretch_image(im_blend)
312
313
314
315
     def relpath(filename):
316
317
          A function that concatenate the
          current path of the image with
318
          the given path.
319
320
          :param filename: The given path.
321
          :return: The concatenate path.
322
323
          return os.path.join(os.path.dirname(__file__), filename)
324
325
     def reshape_image(image, up, down, left, right):
326
327
          A function that cuts parts of the images
328
329
          in the examples so they would look better.
          :param image: A given image.
330
331
          :param up: Number of rows to cut from the
```

```
332
          upper side.
333
          :param down: Number of rows to cut from the
334
          lower side.
          :param left: Number of rows to cut from the
335
          left side.
336
337
          :param right: Number of rows to cut from the
338
          right side.
          :return: The image after the cutting.
339
340
          image = image[up: image.shape[0] - down]
341
          indexes_in_rows_left = np.arange(0, left)
342
343
          image = np.delete(image, indexes_in_rows_left, axis=1)
344
          indexes_in_rows_right = np.arange(image.shape[1] - right - 1,
345
                                             image.shape[1])
346
          return np.delete(image, indexes_in_rows_right, axis=1)
347
348
349
     def show_images(im1, im2, mask, im_blend):
350
351
          A function that plot the images of the
          examples in one plot.
352
          :param im1: The first image.
353
          :param im2: The second image.
354
          :param mask: The mask.
355
356
          :param im_blend: The blended image.
357
          :return: None.
358
359
          fig = plt.figure()
360
361
          a1 = fig.add_subplot(221)
362
          a2 = fig.add_subplot(222)
          a3 = fig.add_subplot(223)
363
364
          a4 = fig.add_subplot(224)
365
          al imshow(im1)
366
367
          a2.imshow(im2)
          a3.imshow(mask, cmap='gray')
368
          a4.imshow(im_blend)
369
370
          a1.set_axis_off()
371
372
          a2.set_axis_off()
373
          a3.set_axis_off()
          a4.set_axis_off()
374
375
376
         plt.show()
377
378
     def blending_example(image1, image2, mask, example):
379
380
          A function that makes the blending for the examples.
381
          :param image1: The first image.
382
383
          :param image2: The second image.
384
          :param mask: The mask.
385
          :param example: 1 for blending_example1,
          2 for blending_example2.
386
          :return: im1, im2, mask, im_blend
387
388
389
          im1 = read_image(image1, 2)
          im2 = read_image(image2, 2)
390
391
          mask = read_image(mask, 1)
          im_blend = im1.copy()
392
393
394
          red_pixels = pyramid_blending(im1[:, :, 0], im2[:, :, 0],
                                         mask, 3, 5, 3)
395
          green_pixels = pyramid_blending(im1[:, :, 1], im2[:, :, 1],
396
397
                                           mask, 3, 5, 3)
          blue_pixels = pyramid_blending(im1[:, :, 2], im2[:, :, 2],
398
399
                                          mask, 3, 5, 3)
```

```
400
          im_blend[:, :, 0] = red_pixels
          im_blend[:, :, 1] = green_pixels
im_blend[:, :, 2] = blue_pixels
401
402
403
404
          if example == 2:
              im1 = reshape_image(im1, 75, 75, 200, 200)
405
              im2 = reshape_image(im2, 75, 75, 200, 200)
406
              mask = reshape_image(mask, 75, 75, 200, 200)
407
408
              im_blend = reshape_image(im_blend, 75, 75, 200, 200)
409
          else:
              im1 = reshape_image(im1, 100, 100, 100, 100)
410
411
              im2 = reshape_image(im2, 100, 100, 100, 100)
              mask = reshape_image(mask, 100, 100, 100, 100)
412
              im_blend = reshape_image(im_blend, 100, 100, 100, 100)
413
414
          show_images(im1, im2, mask, im_blend)
415
416
417
          return im1, im2, mask.astype(np.bool), im_blend
418
419
420
     def blending_example1():
421
          A function that perform pyramid blending
422
423
          on the first set of images.
424
          :return: The first image, the second image, the mask,
425
          the blended image.
426
427
          refa = relpath("externals//final_refa.jpg")
         lafa = relpath("externals//final_lafa.jpg")
428
429
         mask = relpath("externals//final_mask_refa.png")
430
          return blending_example(refa, lafa, mask, 1)
431
432
     def blending_example2():
433
434
435
          A function that perform pyramid blending
436
          on the second set of images.
          :return: The first image, the second image, the mask,
437
          the blended image.
438
439
          dolphin = relpath("externals//final_dolphin.jpg")
440
         water = relpath("externals//final_water.jpg")
441
         mask = relpath("externals//final_mask_dolphin.jpg")
442
443
         return blending_example(dolphin, water, mask, 2)
```

## 7 ex3/externals/final dolphin.jpg



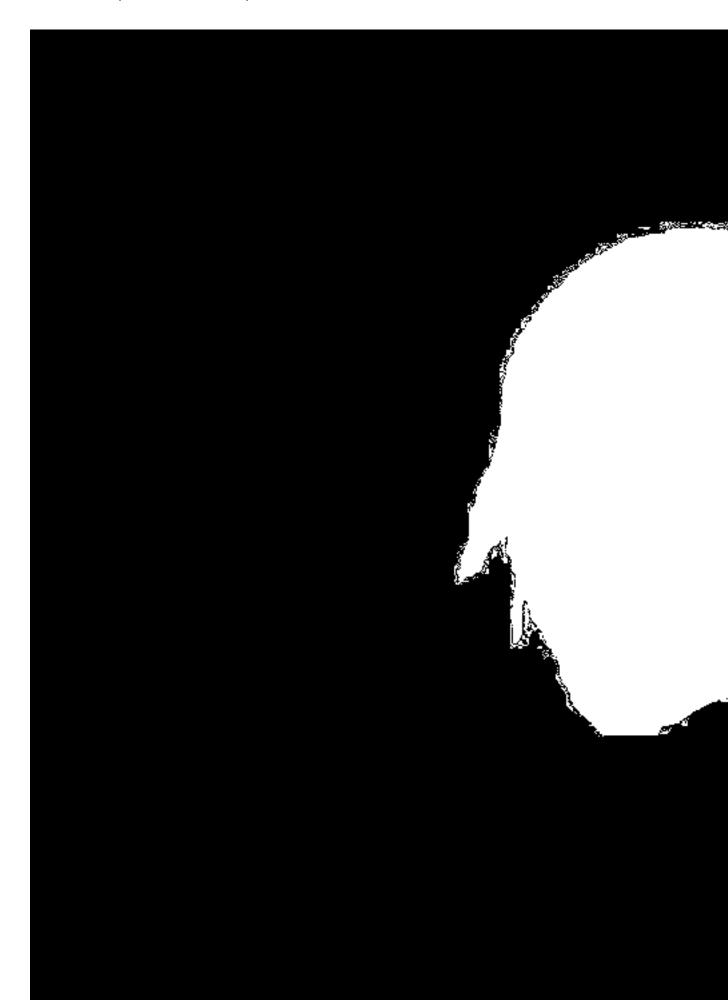
# 8 ex3/externals/final lafa.jpg



## 9 ex3/externals/final mask dolphin.jpg



10 ex3/externals/final mask refa.png



# 11 ex3/externals/final refa.jpg



## 12 ex3/externals/final water.jpg

