Instituto Federal Goiano - Campus Ceres
Bacharelado em Sistemas de Informação
Prof. Me. Ronneesley Moura Teles

Adallberto Lucena Moura
Andrey Silva Ribeiro
Anny Karoliny Moraes Ribeiro
Brener Gomes de Jesus
Davi Ildeu de Faria
Eduardo de Oliveira Silva
Gleyson Israel Alves
Gusttavo Nunes Gomes
Ianka Talita Bastos de Assis
Ígor Justino Rodrigues

Algoritmo Viola-Jones

Outubro
2017

Sumário

Algoritmo Viola-Jones

1 O artigo

Em 2001, *Paul Viola* e *Michael Jones*, dois pesquisadores de Cambridge publicaram uma artigo entitulado: *"Rapid Object Detection using a Boosted Cascade of Simple Features"* que demonstrava um novo método de detecção de faces. O artigo se diferencia e se paltava em 3 pontos importantes.

A primeira foi uma nova maneira de se representar uma imagem, a "imagem integral" (*Integral Image*, em inglês), que permitiu os detectores usados por eles, computarem a imagem de maneira mais rápida.

A segunda foi o algoritmo de aprendizado baseado no *AdaBoost*, que selecionava um número pequeno de características visuais críticas de um conjunto maior e com seus classificadores, extremamente eficientes.

O terceiro aspecto importante, foi o método de combinar e incrementar classificadores em "cascata", o que permitia regiões do fundo da foto de serem rapidamente descatadas, disponibilizando maior processamento computacional em posições com maior possíbilidade de ser o objeto no qual se está procurando, como um rosto.

Os autores aprofundaram seus métodos de como construiram esse algoritmo, de como funcionavam as equações e apresentaram os resultados encontrados e compararam com algoritmos similares da época, e como os algoritmos *"Rowley-Baluja-Kanade"*, *"Schneiderman-Kanade"* e *"Roth-Yang-Ahuja"*.

Devido sua implementação seguir uma abordagem diferente para construção de um sistema de detecção de face, aproximadamente 15 vezes mais rápida que os métodos anteriores, o algoritmo *Viola-Jones* revolucionou esse campo da computação se tornando uma referência.

2 Funcionamento do algoritmo

3 Vantagem

- 15 vezes mais rápido que o algoritmo *"Rowley-Baluja-Kanade"* no processamento da imagem.

- 600 vezes se comparado ao *"Schneiderman-Kanade"*.

4 Desvantagem

- A detecção de faces, só é possível se o rosto estiver na posição frontal.

- A base de dados usada, precisa de faces em diferentes condições incluindo: iluminação, brilho, escala, pose e variações de câmera.

- Nível de detecção na literatura - 80% (FAUX,2012)

- É um algoritmo de detecção de face e não de reconhecimento facial.

5 Implementação Viola-Jones em Python

5.1 Porquê a implementação em Python?

Existem inúmeras implementações do algoritmo na internet, nas mais diversas linguagens, sendo mais comuns em *MatLab*, *Java* e *Python*.

A implementação em MatLab não era muito interessante devido a ferramenta em si, é uma boa ferramenta para desenvolver procedimentos tais como esse de detecção facial. entretanto é uma ferramenta paga,e em um projeto nessa escala, no qual muitas pessoas iriam interagir, não ter a licença de uso, dificultaria muito o processo, e por isso foi descartada.

Java era muito abundante no *GitHub*, entretanto a maioria estava incompleta, não possuindo os códigos para treinar a rede, o que é fundamental para o sucesso do algoritmo, ter uma base de dados ampla gerando uma maior eficiência na detecção de imagens. Os códigos que estavam completos, a documentação era quase inexistente, o que dificultaria a utilização, sendo talvezmais vantajoso desenvolver desde o início uma amplicação do que gastar tempo em código alheio.

Entretanto escolhemos a implementação de *Simon Hohberg* do algoritmo em *Python* disponibilizado no seu *GitHub* pessoal: `https://github.com/Simon-Hohberg/Viola-Jones`

Python é uma linguagem mais simples, mas nem por isso menos robusta, é amplamente usada em processos de *machine learning* e a implementação encontrada, está com todas as partes do algoritmo, desde a parte de detecção até a de treinamento, e a documentação do código está bem feita, o que possibilitaria o uso.

A seguir segue os principais códigos do algoritmo implementado em *Python* por *Simon Hohberg*: `https://github.com/Simon-Hohberg/Viola-Jones`

5.2 IntegralImage.py

```python
import numpy as np


"""
In an integral image each pixel is the sum of all pixels in the
    original image
that are 'left and above' the pixel.

Original     Integral
+---------   +------------
| 1 2 3 .    | 0  0  0  0 .
| 4 5 6 .    | 0  1  3  6 .
| . . . .    | 0  5 12 21 .
             | . . . . . .

"""


def to_integral_image(img_arr):
    """
    Calculates the integral image based on this instance's original
    image data.
    :param img_arr: Image source data
    :type img_arr: numpy.ndarray
    :return Integral image for given image
    :rtype: numpy.ndarray
```

```
25      """
26      # an index of −1 refers to the last row/column
27      # since row_sum is calculated starting from (0,0),
28      # rowSum(x, −1) == 0 holds for all x
29      row_sum = np.zeros(img_arr.shape)
30      # we need an additional column and row
31      integral_image_arr = np.zeros((img_arr.shape[0] + 1, img_arr.shape
        [1] + 1))
32      for x in range(img_arr.shape[1]):
33          for y in range(img_arr.shape[0]):
34              row_sum[y, x] = row_sum[y−1, x] + img_arr[y, x]
35              integral_image_arr[y+1, x+1] = integral_image_arr[y+1, x
        −1+1] + row_sum[y, x]
36      return integral_image_arr
37
38
39  def sum_region(integral_img_arr, top_left, bottom_right):
40      """
41      Calculates the sum in the rectangle specified by the given tuples.
42      :param integral_img_arr:
43      :type integral_img_arr: numpy.ndarray
44      :param top_left: (x, y) of the rectangle's top left corner
45      :type top_left: (int, int)
46      :param bottom_right: (x, y) of the rectangle's bottom right corner
47      :type bottom_right: (int, int)
48      :return The sum of all pixels in the given rectangle
49      :rtype int
50      """
51      # swap tuples
52      top_left = (top_left[1], top_left[0])
53      bottom_right = (bottom_right[1], bottom_right[0])
54      if top_left == bottom_right:
55          return integral_img_arr[top_left]
56      top_right = (bottom_right[0], top_left[1])
57      bottom_left = (top_left[0], bottom_right[1])
58      return integral_img_arr[bottom_right] − integral_img_arr[top_right]
        − integral_img_arr[bottom_left] + integral_img_arr[top_left]
```

recursos/codigo_python/Viola–Jones–master/violajones/IntegralImage.py

5.3 HaarLikeFeature.py

```
1  import violajones.IntegralImage as ii
2
3
4  def enum(**enums):
5      return type('Enum', (), enums)
6
7  FeatureType = enum(TWO_VERTICAL=(1, 2), TWO_HORIZONTAL=(2, 1),
       THREE_HORIZONTAL=(3, 1), THREE_VERTICAL=(1, 3), FOUR=(2, 2))
8  FeatureTypes = [FeatureType.TWO_VERTICAL, FeatureType.TWO_HORIZONTAL,
       FeatureType.THREE_VERTICAL, FeatureType.THREE_HORIZONTAL,
       FeatureType.FOUR]
9
10
11 class HaarLikeFeature(object):
12     """
13     Class representing a haar−like feature.
14     """
```

```python
    def __init__(self, feature_type, position, width, height, threshold
, polarity):
        """
        Creates a new haar-like feature.
        :param feature_type: Type of new feature, see FeatureType enum
        :type feature_type: violajonse.HaarLikeFeature.FeatureTypes
        :param position: Top left corner where the feature begins (x, y
)
        :type position: (int, int)
        :param width: Width of the feature
        :type width: int
        :param height: Height of the feature
        :type height: int
        :param threshold: Feature threshold
        :type threshold: float
        :param polarity: polarity of the feature -1 or 1
        :type polarity: int
        """
        self.type = feature_type
        self.top_left = position
        self.bottom_right = (position[0] + width, position[1] + height)
        self.width = width
        self.height = height
        self.threshold = threshold
        self.polarity = polarity
        self.weight = 1

    def get_score(self, int_img):
        """
        Get score for given integral image array.
        :param int_img: Integral image array
        :type int_img: numpy.ndarray
        :return: Score for given feature
        :rtype: float
        """
        score = 0
        if self.type == FeatureType.TWO_VERTICAL:
            first = ii.sum_region(int_img, self.top_left, (self.
top_left[0] + self.width, int(self.top_left[1] + self.height / 2)))
            second = ii.sum_region(int_img, (self.top_left[0], int(self
.top_left[1] + self.height / 2)), self.bottom_right)
            score = first - second
        elif self.type == FeatureType.TWO_HORIZONTAL:
            first = ii.sum_region(int_img, self.top_left, (int(self.
top_left[0] + self.width / 2), self.top_left[1] + self.height))
            second = ii.sum_region(int_img, (int(self.top_left[0] +
self.width / 2), self.top_left[1]), self.bottom_right)
            score = first - second
        elif self.type == FeatureType.THREE_HORIZONTAL:
            first = ii.sum_region(int_img, self.top_left, (int(self.
top_left[0] + self.width / 3), self.top_left[1] + self.height))
            second = ii.sum_region(int_img, (int(self.top_left[0] +
self.width / 3), self.top_left[1]), (int(self.top_left[0] + 2 *
self.width / 3), self.top_left[1] + self.height))
            third = ii.sum_region(int_img, (int(self.top_left[0] + 2 *
self.width / 3), self.top_left[1]), self.bottom_right)
            score = first - second + third
        elif self.type == FeatureType.THREE_VERTICAL:
```

```python
                first = ii.sum_region(int_img, self.top_left, (self.
    bottom_right[0], int(self.top_left[1] + self.height / 3)))
                second = ii.sum_region(int_img, (self.top_left[0], int(self
    .top_left[1] + self.height / 3)), (self.bottom_right[0], int(self.
    top_left[1] + 2 * self.height / 3)))
                third = ii.sum_region(int_img, (self.top_left[0], int(self.
    top_left[1] + 2 * self.height / 3)), self.bottom_right)
                score = first - second + third
            elif self.type == FeatureType.FOUR:
                # top left area
                first = ii.sum_region(int_img, self.top_left, (int(self.
    top_left[0] + self.width / 2), int(self.top_left[1] + self.height /
     2)))
                # top right area
                second = ii.sum_region(int_img, (int(self.top_left[0] +
    self.width / 2), self.top_left[1]), (self.bottom_right[0], int(self
    .top_left[1] + self.height / 2)))
                # bottom left area
                third = ii.sum_region(int_img, (self.top_left[0], int(self.
    top_left[1] + self.height / 2)), (int(self.top_left[0] + self.width
     / 2), self.bottom_right[1]))
                # bottom right area
                fourth = ii.sum_region(int_img, (int(self.top_left[0] +
    self.width / 2), int(self.top_left[1] + self.height / 2)), self.
    bottom_right)
                score = first - second - third + fourth
        return score

    def get_vote(self, int_img):
        """
        Get vote of this feature for given integral image.
        :param int_img: Integral image array
        :type int_img: numpy.ndarray
        :return: 1 iff this feature votes positively, otherwise -1
        :rtype: int
        """
        score = self.get_score(int_img)
        return self.weight * (1 if score < self.polarity * self.
    threshold else -1)
```

recursos/codigo_python/Viola–Jones–master/violajones/HaarLikeFeature.py

5.4 AdaBoost.py

```python
from functools import partial
import numpy as np
from violajones.HaarLikeFeature import HaarLikeFeature
from violajones.HaarLikeFeature import FeatureTypes
import progressbar
from multiprocessing import Pool


LOADING_BAR_LENGTH = 50



# TODO: select optimal threshold for each feature
# TODO: attentional cascading

def learn(positive_iis, negative_iis, num_classifiers=-1,
    min_feature_width=1, max_feature_width=-1, min_feature_height=1,
```

```python
        max_feature_height=-1):
        """
        Selects a set of classifiers. Iteratively takes the best
        classifiers based
        on a weighted error.
        :param positive_iis: List of positive integral image examples
        :type positive_iis: list[numpy.ndarray]
        :param negative_iis: List of negative integral image examples
        :type negative_iis: list[numpy.ndarray]
        :param num_classifiers: Number of classifiers to select, -1 will
        use all
        classifiers
        :type num_classifiers: int

        :return: List of selected features
        :rtype: list[violajones.HaarLikeFeature.HaarLikeFeature]
        """
        num_pos = len(positive_iis)
        num_neg = len(negative_iis)
        num_imgs = num_pos + num_neg
        img_height, img_width = positive_iis[0].shape

        # Maximum feature width and height default to image width and
        height
        max_feature_height = img_height if max_feature_height == -1 else
        max_feature_height
        max_feature_width = img_width if max_feature_width == -1 else
        max_feature_width

        # Create initial weights and labels
        pos_weights = np.ones(num_pos) * 1. / (2 * num_pos)
        neg_weights = np.ones(num_neg) * 1. / (2 * num_neg)
        weights = np.hstack((pos_weights, neg_weights))
        labels = np.hstack((np.ones(num_pos), np.ones(num_neg) * -1))

        images = positive_iis + negative_iis

        # Create features for all sizes and locations
        features = _create_features(img_height, img_width,
        min_feature_width, max_feature_width, min_feature_height,
        max_feature_height)
        num_features = len(features)
        feature_indexes = list(range(num_features))

        num_classifiers = num_features if num_classifiers == -1 else
        num_classifiers

        print('Calculating scores for images..')

        votes = np.zeros((num_imgs, num_features))
        bar = progressbar.ProgressBar()
        # Use as many workers as there are CPUs
        pool = Pool(processes=None)
        for i in bar(range(num_imgs)):
            votes[i, :] = np.array(list(pool.map(partial(_get_feature_vote,
        image=images[i]), features)))

        # select classifiers

        classifiers = []
```

```python
      print('Selecting classifiers..')
      bar = progressbar.ProgressBar()
      for _ in bar(range(num_classifiers)):

          classification_errors = np.zeros(len(feature_indexes))

          # normalize weights
          weights *= 1. / np.sum(weights)

          # select best classifier based on the weighted error
          for f in range(len(feature_indexes)):
              f_idx = feature_indexes[f]
              # classifier error is the sum of image weights where the classifier
              # is right
              error = sum(map(lambda img_idx: weights[img_idx] if labels[
    img_idx] != votes[img_idx, f_idx] else 0, range(num_imgs)))
              classification_errors[f] = error

          # get best feature, i.e. with smallest error
          min_error_idx = np.argmin(classification_errors)
          best_error = classification_errors[min_error_idx]
          best_feature_idx = feature_indexes[min_error_idx]

          # set feature weight
          best_feature = features[best_feature_idx]
          feature_weight = 0.5 * np.log((1 - best_error) / best_error)
          best_feature.weight = feature_weight

          classifiers.append(best_feature)

          # update image weights
          weights = np.array(list(map(lambda img_idx: weights[img_idx] *
    np.sqrt((1-best_error)/best_error) if labels[img_idx] != votes[
    img_idx, best_feature_idx] else weights[img_idx] * np.sqrt(
    best_error/(1-best_error)), range(num_imgs))))

          # remove feature (a feature can't be selected twice)
          feature_indexes.remove(best_feature_idx)

      return classifiers


def _get_feature_vote(feature, image):
      return feature.get_vote(image)


def _create_features(img_height, img_width, min_feature_width,
    max_feature_width, min_feature_height, max_feature_height):
      print('Creating haar-like features..')
      features = []
      for feature in FeatureTypes:
          # FeatureTypes are just tuples
          feature_start_width = max(min_feature_width, feature[0])
          for feature_width in range(feature_start_width,
    max_feature_width, feature[0]):
              feature_start_height = max(min_feature_height, feature[1])
              for feature_height in range(feature_start_height,
    max_feature_height, feature[1]):
```

8

```
117                for x in range(img_width - feature_width):
118                    for y in range(img_height - feature_height):
119                        features.append(HaarLikeFeature(feature, (x, y)
       , feature_width, feature_height, 0, 1))
120                        features.append(HaarLikeFeature(feature, (x, y)
       , feature_width, feature_height, 0, -1))
121        print('..done. ' + str(len(features)) + ' features created.\n')
122        return features
```

recursos/codigo_python/Viola–Jones–master/violajones/AdaBoost.py

5.5 Utils.py

```
1  import numpy as np
2  from PIL import Image
3  from violajones.HaarLikeFeature import FeatureType
4  from functools import partial
5  import os
6
7
8  def ensemble_vote(int_img, classifiers):
9      """
10      Classifies given integral image (numpy array) using given
       classifiers, i.e.
11      if the sum of all classifier votes is greater 0, image is
       classified
12      positively (1) else negatively (0). The threshold is 0, because
       votes can be
13      +1 or -1.
14      :param int_img: Integral image to be classified
15      :type int_img: numpy.ndarray
16      :param classifiers: List of classifiers
17      :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
18      :return: 1 iff sum of classifier votes is greater 0, else 0
19      :rtype: int
20      """
21      return 1 if sum([c.get_vote(int_img) for c in classifiers]) >= 0
       else 0
22
23
24  def ensemble_vote_all(int_imgs, classifiers):
25      """
26      Classifies given list of integral images (numpy arrays) using
       classifiers,
27      i.e. if the sum of all classifier votes is greater 0, an image is
       classified
28      positively (1) else negatively (0). The threshold is 0, because
       votes can be
29      +1 or -1.
30      :param int_imgs: List of integral images to be classified
31      :type int_imgs: list[numpy.ndarray]
32      :param classifiers: List of classifiers
33      :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
34      :return: List of assigned labels, 1 if image was classified
       positively, else
35      0
36      :rtype: list[int]
37      """
38      vote_partial = partial(ensemble_vote, classifiers=classifiers)
```

9

```python
39          return list(map(vote_partial, int_imgs))


42  def reconstruct(classifiers, img_size):
43          """
44          Creates an image by putting all given classifiers on top of each
        other
45          producing an archetype of the learned class of object.
46          :param classifiers: List of classifiers
47          :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
48          :param img_size: Tuple of width and height
49          :type img_size: (int, int)
50          :return: Reconstructed image
51          :rtype: PIL.Image
52          """
53          image = np.zeros(img_size)
54          for c in classifiers:
55              # map polarity: -1 -> 0, 1 -> 1
56              polarity = pow(1 + c.polarity, 2)/4
57              if c.type == FeatureType.TWO_VERTICAL:
58                  for x in range(c.width):
59                      sign = polarity
60                      for y in range(c.height):
61                          if y >= c.height/2:
62                              sign = (sign + 1) % 2
63                          image[c.top_left[1] + y, c.top_left[0] + x] += 1 *
        sign * c.weight
64              elif c.type == FeatureType.TWO_HORIZONTAL:
65                  sign = polarity
66                  for x in range(c.width):
67                      if x >= c.width/2:
68                          sign = (sign + 1) % 2
69                      for y in range(c.height):
70                          image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
        sign * c.weight
71              elif c.type == FeatureType.THREE_HORIZONTAL:
72                  sign = polarity
73                  for x in range(c.width):
74                      if x % c.width/3 == 0:
75                          sign = (sign + 1) % 2
76                      for y in range(c.height):
77                          image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
        sign * c.weight
78              elif c.type == FeatureType.THREE_VERTICAL:
79                  for x in range(c.width):
80                      sign = polarity
81                      for y in range(c.height):
82                          if x % c.height/3 == 0:
83                              sign = (sign + 1) % 2
84                          image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
        sign * c.weight
85              elif c.type == FeatureType.FOUR:
86                  sign = polarity
87                  for x in range(c.width):
88                      if x % c.width/2 == 0:
89                          sign = (sign + 1) % 2
90                      for y in range(c.height):
91                          if x % c.height/2 == 0:
92                              sign = (sign + 1) % 2
```

```
93                          image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
         sign * c.weight
94          image -= image.min()
95          image /= image.max()
96          image *= 255
97          result = Image.fromarray(image.astype(np.uint8))
98          return result
99

100
101  def load_images(path):
102      images = []
103      for _file in os.listdir(path):
104          if _file.endswith('.png'):
105              img_arr = np.array(Image.open((os.path.join(path, _file))),
         dtype=np.float64)
106              img_arr /= img_arr.max()
107              images.append(img_arr)
108      return images
```

recursos/codigo_python/Viola–Jones–master/violajones/Utils.py

5.6   example.py

```
1  import violajones.IntegralImage as ii
2  import violajones.AdaBoost as ab
3  import violajones.Utils as utils
4
5  if __name__ == "__main__":
6      pos_training_path = 'trainingdata/faces'
7      neg_training_path = 'trainingdata/nonfaces'
8      pos_testing_path = 'trainingdata/faces/test'
9      neg_testing_path = 'trainingdata/nonfaces/test'
10
11      num_classifiers = 2
12      # For performance reasons restricting feature size
13      min_feature_height = 8
14      max_feature_height = 10
15      min_feature_width = 8
16      max_feature_width = 10
17
18      print('Loading faces..')
19      faces_training = utils.load_images(pos_training_path)
20      faces_ii_training = list(map(ii.to_integral_image, faces_training))
21      print('..done. ' + str(len(faces_training)) + ' faces loaded.\n\
         nLoading non faces..')
22      non_faces_training = utils.load_images(neg_training_path)
23      non_faces_ii_training = list(map(ii.to_integral_image,
         non_faces_training))
24      print('..done. ' + str(len(non_faces_training)) + ' non faces
         loaded.\n')
25
26      # classifiers are haar like features
27      classifiers = ab.learn(faces_ii_training, non_faces_ii_training,
         num_classifiers, min_feature_height, max_feature_height,
         min_feature_width, max_feature_width)
28
29      print('Loading test faces..')
30      faces_testing = utils.load_images(pos_testing_path)
31      faces_ii_testing = list(map(ii.to_integral_image, faces_testing))
```

```python
32     print('..done. ' + str(len(faces_testing)) + ' faces loaded.\n\
    nLoading test non faces..')
33     non_faces_testing = utils.load_images(neg_testing_path)
34     non_faces_ii_testing = list(map(ii.to_integral_image,
    non_faces_testing))
35     print('..done. ' + str(len(non_faces_testing)) + ' non faces loaded
    .\n')
36
37     print('Testing selected classifiers..')
38     correct_faces = 0
39     correct_non_faces = 0
40     correct_faces = sum(utils.ensemble_vote_all(faces_ii_testing,
    classifiers))
41     correct_non_faces = len(non_faces_testing) - sum(utils.
    ensemble_vote_all(non_faces_ii_testing, classifiers))
42
43     print('..done.\n\nResult:\n      Faces: ' + str(correct_faces) + '/
    ' + str(len(faces_testing))
44          + ' (' + str((float(correct_faces) / len(faces_testing)) *
    100) + '%)\n  non-Faces: '
45          + str(correct_non_faces) + '/' + str(len(non_faces_testing))
    + ' ('
46          + str((float(correct_non_faces) / len(non_faces_testing)) *
    100) + '%)')
47
48     # Just for fun: putting all haar-like features over each other
    generates a face-like image
49     recon = utils.reconstruct(classifiers, faces_testing[0].shape)
50     recon.save('reconstruction.png')
```

recursos/codigo_python/Viola–Jones–master/example.py

6   Referências Bibliográfica

VIOLA, P. e JONES, M. **Rapid object detection using a boosted cascade of simple features.** Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, v. 1, p. I-511-I-518, 2001. Disponível em: <http://ieeexplore.ieee.org/document/990517/>.

VIOLA, P. e JONES, M. **Robust Real-Time Face Detection** International Journal of Computer Vision 57(2), 137–154, 2004.

IRGENS, Peter et al. **An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm.** HardwareX, v. 1, p. 68–75, 2017. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S2468067216300116>.

SANTOS, Ligneul. **Detecção de faces através do algoritmo de Viola-Jones**. Coppe/Ufrj, 2011.

FAUX, Francis e LUTHON, Franck. **Theory of evidence for face detection and tracking**. International Journal of Approximate Reasoning, v. 53, n. 5, p. 728–746, 2012. Disponível em: <http://dx.doi.org/10.1016/j.ijar.2012.02.002>.

BODHI, S. R. e NAVEEN, S. **Face detection, registration and feature localization experiments with RGB-D face database**. Procedia Computer Science, v. 46, n. Icict 2014, p. 1778–1785, 2015. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.02.132>.