CORREÇÃO AUTOMÁTICA DE AVALIAÇÕES OBJETIVAS COM LEITURA DO CARTÃO-RESPOSTA POR VISÃO COMPUTACIONAL

Lucas Ceroni Menoncin[1], Ricardo Antonello[2]
[1]Instituto Federal Catarinense, Campus Luzerna / lucas.ceroni@hotmail.com
[2]Instituto Federal Catarinense, Campus Luzerna / ricardo.antonello@ifc.edu.br

***Resumo:** Visando facilitar e agilizar a correção de provas objetivas, apresentamos aqui um algoritmo escrito em Python utilizando a biblioteca OpenCV (Open Source Computer Vision) que faz a leitura da imagem e a identificação das respostas. A imagem a ser analisada é composta na parte superior por um QR-Code de identificação e um cartão-respostas abaixo contendo as respostas assinaladas pelo aluno. O sistema desenvolvido utiliza uma webcam comum com 1 megapixel de resolução e um computador padrão PC para processamento. O algoritmo recebe um streaming de vídeo capturado pela webcam e faz a análise frame a frame. Ao identificar um retângulo com 50 retângulos menores no frame (quantidade de alternativas), o algoritmo alinha a imagem e faz a identificação das respostas assinaladas no gabarito anexo. O código QR-Code é utilizado para identificar a avaliação e buscar o gabarito com as respostas corretas. O conjunto de respostas é gerado a partir da captura do cartão-resposta na imagem e é comparado com o gabarito para que seja calculada a nota final da avaliação. A precisão na avaliação é de 100% nos casos onde o cartão-resposta foi preenchido corretamente.*

***Palavras-Chave:** Visão computacional, Processamento de imagem, OpenCV.*

1. INTRODUÇÃO

O presente artigo apresenta os resultados de uma parcela do projeto chamado SmarTest: Sistema de Gerenciamento de Avaliações. O algoritmo apresentado tem como objetivo através da visão computacional reconhecer o cartão-resposta de uma avaliação objetiva e retirar as respostas assinaladas, para que assim possa ser comparado com um gabarito correto pré-determinado e seja realizada a correção automática.

O algoritmo em questão precisa de duas entradas, o gabarito correto da prova e uma entrada de vídeo. A saída então é um vetor numérico de 10 posições para cada uma das alternativas, e suas posições “0” a “4” para representar de “A” a “E”, e também a nota da prova.

Para o desenvolvimento do algoritmo foi utilizada a linguagem de programação Python, e bibliotecas como OpenCV - Open Source Computer Vision Library (OPENCV, 2017), Numpy - Numeric Python (NUMPY, 2017) e imutils (ROSEBROCK, 2017a), uma biblioteca auxiliar usada para manipular contornos da imagem.

2. RECURSOS UTILIZADOS

2.1. OpenCV

OpenCV - Open Source Computer Vision Library (OPENCV, 2017) é uma biblioteca de funções que trabalham com visão computacional. Foi construído para dispor uma infraestrutura

comum para aplicações de visão computacional, e acelerar o uso das percepções da máquina em produtor comerciais (OPENCV, 2017).

A biblioteca possui mais de 2.500 algoritmos otimizados, podendo ser usados para detectar e reconhecer faces, identificar objetos, classificar movimentos humanos em tempo real, seguir movimentos em câmeras, seguir objetos, dentre outras inúmeras utilidades, já alcançando assim a marca de 47 mil pessoas em sua comunidade e 14 milhões de downloads, e sendo utilizado desde companhia e grupos de pesquisa até órgãos governamentais (OPENCV, 2017).

Embora seja naturalmente escrito em C++ (C/C++, 2017), OpenCV (OPENCV, 2017) pode também ser utilizado nas linguagens C++ (C/C++, 2017), C (C/C++, 2017), Python (PYTHON, 2017), Java (JAVA, 2017), MATLAB (MATLAB, 2017) e suporta diversos sistemas operacionais.

2.2. Python

Criada por Guido van Rossum na década de 90, é uma linguagem de programação de altíssimo nível, que consegue combinar um notável poder de execução com uma sintaxe muito limpa, organizada e objetiva (PYTHON, 2017).

A linguagem possui uma imensa biblioteca, contendo classes, métodos e funções para a realização de todos os tipos de tarefa, suportando assim vários “níveis” de programação, desde programas simples e rápidos, até mesmo mais longos e complexos, podendo estar presente assim de forma confiável em grandes projetos (PYTHON, 2017).

Python além de tudo, é multiplataforma, então, os algoritmos escritos em uma plataforma podem ser tranquilamente compilados em outra, sem erros de compatibilidade (PYTHON, 2017).

2.3. Oracle VM VirtualBox

É um aplicativo que permite a execução de vários sistemas operacionais em um computador multiplataforma, podendo ser instalado em diversos sistemas operacionais (ORACLE, 2011).

Deste modo, podemos destacar algumas vantagens da utilização do Oracle Virtualbox:

• Criação de laboratórios para estudos sem danificar seu atual sistema operacional;

• Possui a funcionalidade de exportar/importar máquinas virtuais de outros fabricantes.

3. DESENVOLVIMENTO

Inicialmente tentamos buscar padrões nas imagens, porém, como estamos lidando com imagens vindas de câmeras diferentes temos muitas variáveis a considerar, como resolução, ângulo, luminosidade, distancia, dentre outros, o que torna muito difícil encontrar um ponto de referência em comum.

Baseando-se em um algoritmo desenvolvido por Adrian Rosebrock (2017b), utilizamos então os contornos para reconhecer tanto o retângulo do cartão respostas na imagem, quanto cada única alternativa.

Tabela 1 – Código fonte

Linha	Código
1	`from imutils.perspective import four_point_transform`
2	`from imutils import contours`
3	`import numpy as np`
4	`import imutils`
5	`import cv2`
6	`ct=0`
7	`cap = cv2.VideoCapture(0)`
8	`correct = 0`
9	`gb = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]`

Fonte: Os autores.

Linha 1-5 importamos as bibliotecas necessárias. Nas linhas 6 e 8 definimos variáveis importantes no futuro. Na linha 7 acionamos a entrada de vídeo e na linha 9 definimos as respostas corretas para o gabarito em que “0” corresponde a “A” sucessivamente até “4” corresponder a “E”.

Tabela 2 – Código fonte

Linha	Código
10	`while(1):`
11	` ct=0`
12	` ret, image = cap.read()`
13	` gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`
14	` blurred = cv2.GaussianBlur(gray, (3, 3), 0)`
15	` edged = cv2.Canny(blurred, 20, 150)`
16	` cv2.imshow("Camera", edged)`
17	` if cv2.waitKey(1) & 0xFF == ord('q'):`
18	`  break`
19	` cv2.moveWindow("Camera",0,0)`
20	
21	` cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,`
22	`  cv2.CHAIN_APPROX_SIMPLE)`
23	` cnts = cnts[0] if imutils.is_cv2() else cnts[1]`
24	` docCnt = None`
25	
26	` if len(cnts) > 0:`
27	`  cnts = sorted(cnts, key=cv2.contourArea, reverse=True)`
28	`  for c in cnts:`
29	`   peri = 0.02 * cv2.arcLength(c, True)`
30	`   approx = cv2.approxPolyDP(c, peri, True)`
31	`   if len(approx) == 4:`
32	`    ct = 1`
33	`    docCnt = approx`
34	`    break`
35	

Fonte: Os autores.

Abrimos aqui um laço de repetição que tem por objetivo procurar o retângulo do gabarito na imagem, ele só é quebrado no momento em que cumpre o seu papel ou encontra algo semelhante, mas isso não é problema, mais a frente existe outra função para ter certeza que o encontrado foi o cartão respostas.

Na linha 12 capturamos a imagem de vídeo, na linha 13 convertemos para preto e branco para que na linha 14 ela seja borrada. Trabalhando com visão computacional é muito melhor e mais seguro borrar as imagens. Na linha 15 pegamos os contornos da imagem e os mostramos na linha 16. Os comandos da linha 17 e 18 serve para deixar a imagem na tela, em tempo real, e 19 para mover a janela para um local determinado.

Na linha 21 agregamos os contornos a uma variável e começamos então a procurar nosso retângulo do cartão respostas. Passamos da linha 26 somente se a quantidade de contornos for maior que 0, caso contrário, voltamos ao início do laço de repetição na linha 10. O comando da linha 27 ordena os contornos por tamanho em ordem decrescente. Nas linhas 30 e 31 aproximamos os contornos para que na linha 32 se a aproximação tiver 4 pontos, assumimos que encontramos nosso cartão resposta ou algo semelhante a ele. Então acrescentamos 1 a variável “ct”, fornecemos o contorno encontrado a variável “docCnt” e quebramos o laço para continuarmos o código.

Tabela 3 – Código fonte

Linha	Código
36	`if ct==1:`
37	`paper = four_point_transform(image, docCnt.reshape(4, 2))`
38	`warped = four_point_transform(gray, docCnt.reshape(4, 2))`
39	`altura = paper.shape[0]//11`
40	`largura = paper.shape[0]//2.95`
41	`paper =paper[altura:paper.shape[0],largura:paper.shape[1]]`
42	
43	`thresh = cv2.threshold(warped, 0, 255,`
44	`cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]`
45	
46	`thresh = thresh[altura:thresh.shape[0],largura:thresh.shape[1]]`
47	
48	`if thresh.shape[0] > 0  and thresh.shape[1] > 0:`
49	`cnts = cv2.findContours(thresh.copy(),`
50	`cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)`
51	`cnts = cnts[0] if imutils.is_cv2() else cnts[1]`
52	`questionCnts = []`
53	
54	`for c in cnts:`
55	`tamanho = thresh.shape[1]/5`
56	`(x, y, w, h) = cv2.boundingRect(c)`
57	`ar = w / float(h)`
58	`approx = cv2.approxPolyDP(c, peri, True)`
59	`if (w<=tamanho and h < tamanho) and (ar>=1.6`

```
        and ar<=2.6) and (w>tamanho/10 and h>tamanho/10) :
60                                  questionCnts.append(c)
61                      print(len(questionCnts))
62                      if len(questionCnts) == 50:
63                            break
```

Fonte: Os autores.

Chegando aqui temos a imagem do possível cartão respostas, passando pelo *if* da linha 36, na linha 37 e 38 deixamos a imagem reta, e nas linhas 39, 40 e 41 recortamos a imagem para ter apenas os retângulos das alternativas, e não as letras e números, pois estes iriam atrapalhar muito no reconhecimento.

Na linha 43 transformamos os pixels mais escuros da imagem em preto e os mais claros em branco, e na linha 46 recortamos a imagem.

Para garantir que temos mesmo uma imagem passamos pela linha 48, e então retiramos os contornos da imagem preto e branco. E criamos a lista "questionCnts".

Entramos em outro laço de repetição (54), agora queremos analisar a quantidade de objetos que temos, que nesse caso são as alternativas do gabarito. Precisamos então filtrar os contornos obtidos para o caso de encontrar algo indesejado que atrapalhe na correção, para isso a linha 59. Essa linha analisa três coisas, o tamanho da divisão da largura pela altura do contorno, e os tamanhos mínimo e máximo que o objeto pode ter. Caso passe por todas essas condições, o contorno será incrementado na lista "questionCnts". Na linha 62, se o tamanho da lista for exatamente igual a 50 (número de alternativas "5" multiplicado ao número de questões "10"), então estamos lidando com o cartão resposta, pegamos a imagem da câmera para que a partir de agora possamos trabalhar em cima dela.

Tabela 4 – Código fonte

Linha	Código

```
64    cont = 0
65    x=0
66    y=0
67    res = []
68    bubbled = []
69    questao = []
70    for (q, i) in enumerate(np.arange(0, len(questionCnts), 5)):
71          cont=0
72          cnts = contours.sort_contours(questionCnts[i:i + 5])[0]
73          bubbled = []
74          for (j, c) in enumerate(cnts):
75                x = thresh.shape[0]
76                y = thresh.shape[1]
77                mask = np.zeros(thresh.shape, dtype="uint8")
78                cv2.drawContours(mask, [c], -1, 255, -1)
79
80                mask = cv2.bitwise_and(thresh, thresh, mask=mask)
81                total = cv2.countNonZero(mask)
```

```
82                  if total > x//20*y//10:
83                        bubbled.append(j)
84                        cont+=1
```

Fonte: Os autores.

Nas linhas 64 a 69 definimos algumas variáveis e listas importantes, entramos em um laço de repetição novamente. O objetivo aqui é reconhecer quais alternativas estão marcadas e inserir em uma lista chamada “res” através de “bubbled” no próximo passo. Cada vez que rodar o laço de repetição, ele irá analisar de 5 em 5 alternativas, começando pelo canto inferior esquerdo indo para a direita, e subindo, analisando cada alternativa de cada questão. Após o próximo loop da linha 74, existe uma máscara que revela apenas o retângulo da alternativa a ser analisada (78) e a aplicamos então em uma imagem binaria na linha 80, contamos então a quantia de pixels brancos (devido a binarização) na área do retângulo da alternativa.

Na linha 82 fazemos uma análise para confirmar se o que temos é realmente uma alternativa marcada, caso seja a lista “bubbled” irá receber um valor de 0 a 4, referente a letra assinalada no cartão resposta e “cont” irá receber 1 a cada vez que houver uma variável marcada, quando mudamos de linha, resetamos essa variável.

Tabela 5 – Código fonte

Linha	Código

```
85          if cont == 1:
86                res.append(bubbled[0])
87          else:
88                res.append(-1)
89          color = (0, 0, 255)
90          k = gb[q]
91          if cont == 1:
92                if k == bubbled[0]:
93                      color = (0, 255, 0)
94                      correct += 1
95
96          for s in range (cont):
97                cv2.drawContours(paper, [cnts[bubbled[s]]], -1, color, 3)
98    res2=[]
99    for i in range(len(res)):
100         res2.append(res[len(res)-i-1])
101   print("Gabarito:",gb)
102   print("Respostas:",res2)
103   print("Nota:",float(correct))
104   cv2.imshow("Cartao Resposta", paper)
105   cv2.waitKey(0)
106   cv2.imshow("real",image)
107   cv2.waitKey(0)
108   cap.release()
```

Fonte: Os autores.

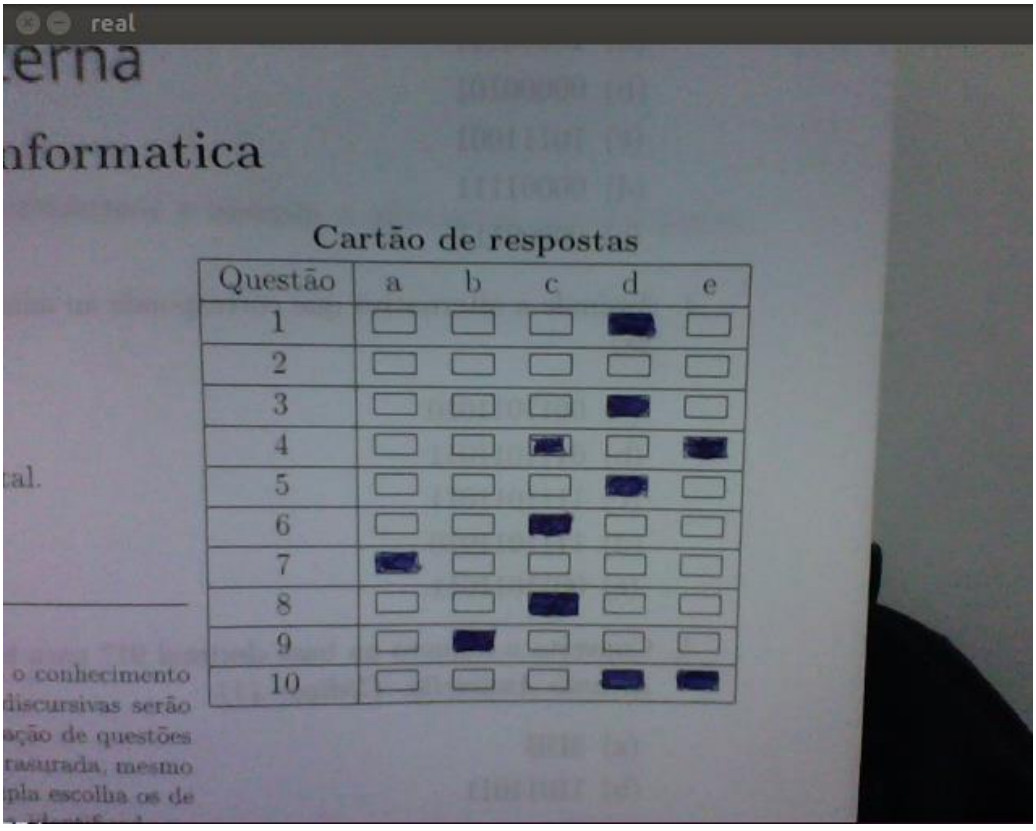
A variável “cont”, contava a quantidade de alternativas marcadas em uma única questão, caso seja apenas 1, mandamos para a lista “res” a resposta assinalada, caso seja 0, ou maior que 1, enviamos um “-1” anulando automaticamente a questão na prova. Logo a baixo começamos a fazer as comparações e dar cores aos contornos, verde aos corretos e vermelho aos incorretos e então nas linhas 96 e 97 desenhamos os contornos na imagem do cartão resposta.

As linhas 96, 97 e 98 servem para inverter a variável “res” na variável “res2”, pois quando analisamos a variável “res”, era de cima para baixo, e queremos o inverso disso. A partir da linha 101, sucessivamente mostramos na tela o gabarito correto, as respostas marcadas e a nota, e imprimimos na tela na forma de imagem o cartão respostas, e a imagem original com a qual trabalhamos.

4. RESULTADOS E DISCUSSÃO

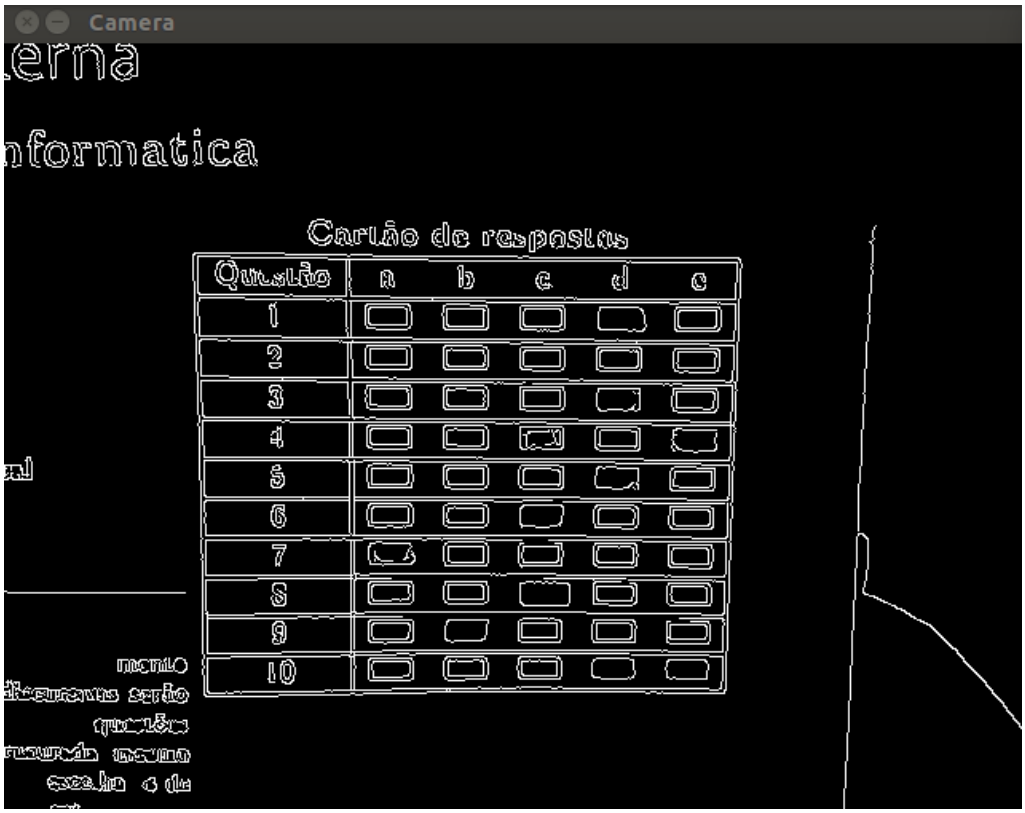
Para testar o algoritmo, inserimos o gabarito como se todas as respostas fossem a letra D, e então na entrada de vídeo, mostramos o gabarito da prova. No momento em que o programa detectou o gabarito, ele passa a trabalhar com as seguintes imagens:

Imagem 1 – Imagem real capturada pela câmera.



Fonte: Os autores.

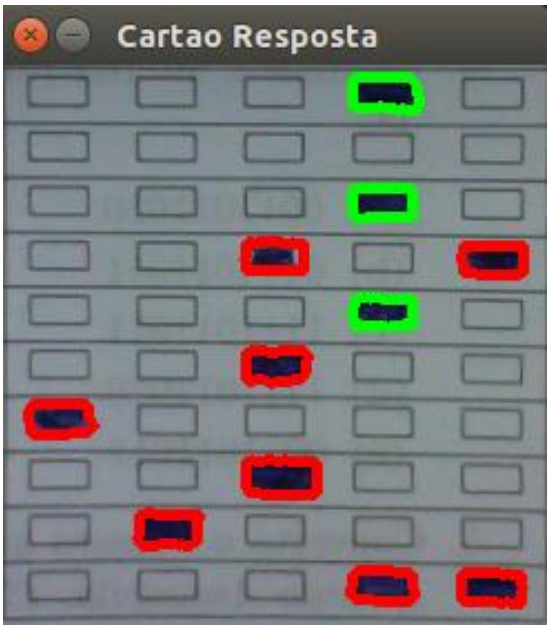
Imagem 2 – Imagem com um filtro de contornos.



Fonte: Os autores.

O algoritmo então recorta apenas o gabarito com todas as 50 alternativas e compara com as respostas corretas:

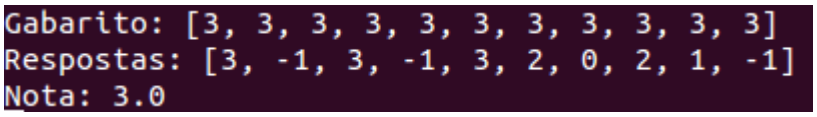
Imagem 3 – Gabarito recortado e respostas já marcadas.



Fonte: Os autores.

E a saída no terminal:

Imagem 4 – Terminal.



Fonte: Os autores.

Note que, para alternativas é melhor trabalhar com números em nosso caso, em que "0" corresponde a "A", e sucessivamente até "4" ser equivalente a "E". Temos então o gabarito sendo todas as corretas como "D". Já nas respostas notamos além das alternativas de "0" a "4", o número "-1", sendo ele o indicador de erro, para os casos de nenhuma, ou mais de uma alternativa serem marcadas.

5. CONSIDERAÇÕES FINAIS

Trabalhando com manipulação e reconhecimento de imagens é extremamente complexo desenvolver um algoritmo totalmente confiável e livre de erros. Porém, conseguimos um acerto de 100% no caso de imagens capturadas adequadamente, com boa iluminação e foco. Já para imagens parcialmente desfocadas e capturadas em condições adversas de luz os erros aumentam. O próximo passo é tornar o algoritmo o mais próximo possível de ser 100% confiável mesmo em baixas condições de luminosidade, para que possa ser utilizado de forma rápida e pratica por qualquer um.

AGRADECIMENTOS

Agradecemos ao IFC – Instituto Federal Catarinense pelo apoio por meio do edital 79/2016 para realização desse projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

C/C++. Disponível em: <http://cppinstitute.org/about>. Acesso em 11 de jul. 2017.

JAVA. **Obtenha Informações sobre a Tecnologia Java.** Disponível em: < https://www.java.com/pt_BR/about/>. Acesso em 11 de jul. 2017.

MATLAB. **The Language of Technical Computing.** Disponível em: < https://au.mathworks.com/products/matlab.html>. Acesso em 11 de jul. 2017.

NUMPY. **NumPy.** Disponível em: < http://www.numpy.org/>. Acesso em 27 de jul. 2017

OPENCV. **About OpenCV**. Disponível em: <http://opencv.org/about.html >. Acesso em: 11 de jul. 2017.

PYTHON. **Python is a programming language that lets you work quickly**

and integrate systems more effectively. Disponível em: <https://www.python.org/>. Acesso em: 11 de jul. 2017.

ROSEBROCK, Adrian. **I just open sourced my personal imutils package: A series of OpenCV convenience functions.** Disponível em: <http://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/>. Acesso em 11 de jul. 2017a.

ROSEBROCK, Adrian. **Bubble sheet multiple choice scanner and test grader using OMR, Python and OpenCV.** Disponível em: <http://www.pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/>. Acesso em 11 de jul. 2017b.

VIRTUALBOX. **Oracle VM VirtualBox.** Disponível em: <https://www.virtualbox.org/>. Acesso em: 11 de jul. 2017.