Centro Estadual de Educação Tecnológica Paula Souza
Faculdade de Tecnologia de Carapicuíba
10 a 15 de julho de 2020
ISSN: 2526-6241

PROJETO FATEQUINO: Reconhecimento de Faces e Gestos

Equipe Visão

Bruna Weber da Nóbrega – Fatec Carapicuíba
Caio César Silva Paulino – Fatec Carapicuíba
Feliphe Lorrã Serodio Jesus – Fatec Carapicuíba
Francisco Roniele Melo de Castro – Fatec Carapicuíba
Gabriel Soler Belmonte – Fatec Carapicuíba
Guilherme Carvalho Caldeira – Fatec Carapicuíba
Lucas Paes de Oliveira – Fatec Carapicuíba

Prof.(a) Mario Marques – Fatec Carapicuíba

CARAPICUIBA/SP

2020

RESUMO

Este projeto traz informações e soluções para problemas em um software na área de inteligência artificial do grupo de visão. Por ser um tema amplo para exploração e desenvolvimento, realizou-se pesquisas referente a captura de imagens por intermédio de uma câmera com o objetivo de reconhecimento de faces e movimentos. A partir daí, foi modelado um software para atender essa necessidade. Para o aprimoramento utiliza-se as bibliotecas de reconhecimento facial. Por não haver projetos semelhantes na Fatec Carapicuíba, esse trabalho auxiliará outros desenvolvedores para mais implementações e aplicações. Constata-se que uma implantação desse software aprimorara significativamente a interação do Fatequino.

Palavras-chave: Interação. Software. Visão.

ABSTRACT

This project brings information and solutions to problems in software in the artificial intelligence area of the vision group. As it is a broad topic for exploration and development, research was carried out regarding the capture of images through a camera with the objective of recognizing faces and movements. From there, software was modeled to meet this need. A practical tool for this improvement is used in libraries for facial recognition. As there are no similar projects at Fatec Carapicuíba, this work will assist other developers for more implementations and applications. It appears that an implementation of this software had significantly improved the interaction of Fatequino.

Keywords: Interaction. Software. Vision.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente na Fatec Carapicuíba os alunos conseguem informações sobre aulas e professores somente pelo site disponibilizado (SIGA e site oficial da Fatec Carapicuíba), entretanto há alunos que possuem dificuldades para realizar consultas sobre horário de aulas e professores.

Pensando nisso foi iniciado um projeto para que este problema seja solucionado de forma rápida e eficiente. O projeto nomeado “Fatequino”, possui esse propósito de auxiliar tanto alunos que não possuem conhecimento da instituição ex: novatos, novos professores e visitantes. Auxiliando-os e facilitando em suas buscas através de uma informação simples e rápida.

O projeto está dividido em 6 grupos que realizarão o desenvolvimento de suas respectivas áreas: Controle, Interação, Mecânica 1 e 2, Visão e Web.

Neste artigo falaremos sobre a parte de visão, nosso objetivo principal é realizar reconhecimento facial de pessoas e objetos, a fim de que seja alcançado o resultado estipulado pelo Fatequino.

2 FUNDAMENTAÇÃO TEÓRICA

Utilizamos dos arquivos anteriores disponibilizados no Trello, Google Drive e GitHub para maior entendimento do que deveria ser aprimorado e para o desenvolvimento utilizamos a Internet como fonte de pesquisa.

2.1 O que é segmentação

Em visão computacional, segmentação se refere ao processo de dividir uma imagem digital em múltiplas regiões (conjunto de pixels) ou objetos, com o objetivo de simplificar e/ou mudar a representação de uma imagem para facilitar a sua análise (SALDANHA, 2009). Segmentação de imagens é tipicamente usada para localizar objetos e formas (linhas, curvas etc.) em imagens (NIELSEN, 2004).

Como resultado, é exibido um conjunto de contornos retirados da imagem após a segmentação, tendo cada pixel da região com menção a alguma característica ou propriedade computacional (ex. cor, textura, continuidade etc.). Áreas próximas devem ter diferenças expressivas com relação a mesma característica.

Segundo Sobral (2002, p.1)

A segmentação pode seguir duas estratégias genéricas:

Descontinuidade: A partição da imagem é efetuada com base nas alterações bruscas de intensidade (ex: detecção de contornos).
Similaridade: A partição é efetuada com base na similaridade entre pixels, seguindo um determinado critério (ex: binarização, crescimento de regiões, divisão e junção de regiões).

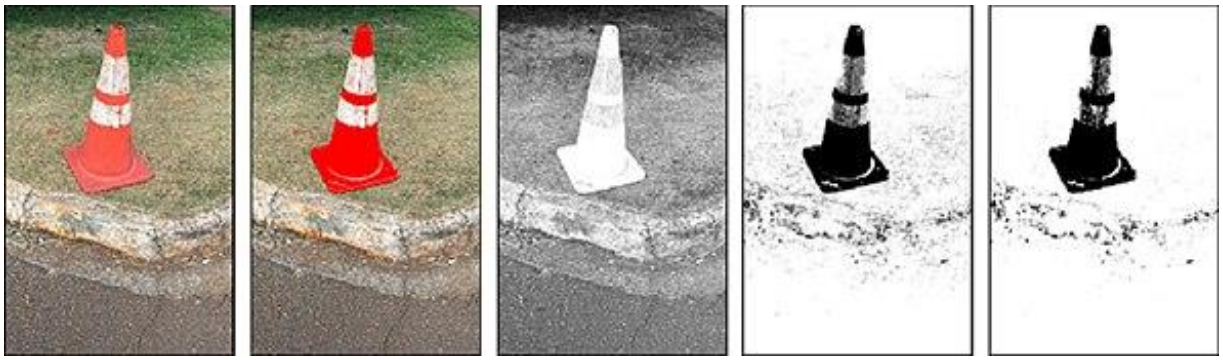
2.2 Segmentação de Movimento

O movimento é uma ação usada por seres humanos e animais, podendo ser utilizado para retirar um objeto de interesse de um fundo com detalhes desnecessários, sendo assim é muito útil na segmentação. O modelo básico é formado pela comparação pixel a pixel entre duas imagens e detectadas de um mesmo ambiente.

Com isso a imagem obtida será apenas a silhueta do objeto que se movimentou, pois o ambiente continuará o mesmo e será removida com a diferença.

A seguir é possível ver dois exemplos de imagens segmentadas, conforme Figura 01 e Figura 02.

Figura 01: Imagem Segmentada



Fonte: GABAN, Jean (2018, Wikipedia)

Figura 02: Imagem Segmentada Após Erosão

Fonte: SOUZA, Guilherme Schirmer (Visão Computacional)

Entretanto este método possui empecilhos: o local deve estar continuamente bem iluminado, o tempo entre as imagens deve ser pequeno para pegar somente uma silhueta do objeto e grande o bastante para ser possível a observação do movimento. Uma possibilidade de melhoria é a utilização de várias imagens, para as diferenças serem acumulativas, sendo mais nítida e com poucos ruídos da imagem.

3 PROCEDIMENTOS METODOLÓGICOS

3.1 Linguagem de Programação utilizada

Como nos grupos anteriores continuaremos a utilizar a linguagem Python, pois ela demonstra-se perfeita e compatível para continuarmos o projeto. Utilizaremos também o OpenCV que é uma biblioteca multiplataforma. Esta ferramenta é totalmente gratuita, voltada para estudos acadêmicos, têm a capacidade de processamento de imagens, ajudando assim, a distinguir os rostos das pessoas que serão cadastradas no Fatequino.

3.2 Análise do Código

O código disponibilizado pela equipe anterior traz informações de como o programa funciona, para melhor utilização é necessário um arquivo que recebe as imagens que são capturadas pela câmera e são tratadas (através do ENCODE_FACE.txt e o RECOGNIZE_FACES_VIDEO.txt).

ENCODE FACES: Nesse arquivo temos as bibliotecas utilizadas na aplicação, a busca de caminho das imagens, a interação com as mesmas e temos os encoding faciais salvos em um arquivo

RECOGNIZE FACES VIDEO: Já neste arquivo temos as importações de bibliotecas responsáveis pela inicialização da stream de vídeo, busca de encode facial e similares para comparação.

3.2.1 Atualização do Código

Devido a complicações com algumas bibliotecas no código utilizado pelo grupo anterior, foi recomendado e orientado pelo professor uma alteração dele. Então nos foi dado outras referências como GENT (2016) e VIANA (2018), assim implantamos outro método de reconhecimento de faces pela câmera, utilizando os arquivos landmarks.py e o arquivo.dat (que nomeamos como no projeto original). Durante o desenvolvimento usamos como base o site PyPI e o Anaconda, portanto todo o processo de instalação foi realizado dentro do ambiente virtual Anaconda.

landmarks.py: No arquivo encontram-se as bibliotecas, as coordenadas da detecção de 68 pontos de referência de faces pela câmera, que são checados com base no arquivo ".dat".

"arquivo".dat: Arquivo identificador de pontos de referência, que no método "shapepredictor" da biblioteca dlib (no arquivo landmarks.py), busca desse arquivo as coordenadas para a detecção dos pontos de referências.

4 DESENVOLVIMENTO

4.1 Grupo Anterior

Estava desenvolvendo o reconhecimento facial de pessoas para quando o robô entrasse em contato com a pessoa, o Fatequino pudesse reconhecê-la e auxiliá-la através de perguntas feitas pela pessoa.

4.2 Grupo Atual

Documentar os métodos de instalação no projeto, atualizar no repositório github os arquivos e se possível implementar o reconhecimento de gestos com o propósito de identificar faces e gestos. Caso alguém queira interagir com o Fatequino, o usuário deverá fazer o sinal de "V", levantando o dedo indicador e médio de forma direta para a câmera do robô a fim de

que o Fatequino tenha a ciência de sua solicitação para responder alguma dúvida, seja de alunos ou professores.

4.3 Implementação

Trabalharemos com a segmentação (processamento de imagem) para identificar formas nas imagens, com o objetivo de poder identificar possíveis rostos e gestos. Há vários tipos de segmentação, porém a que mais se adequa ao nosso projeto é a descontinuidade de movimento, pois como o Fatequino estará sempre em movimento na Fatec, os obstáculos serão móveis e não fixos (no caso de pessoas), para futuramente também poder utilizar o aceno (HandGesture.py), esse tipo de segmentação irá se adequar bem ao que foi sugerido para o projeto.

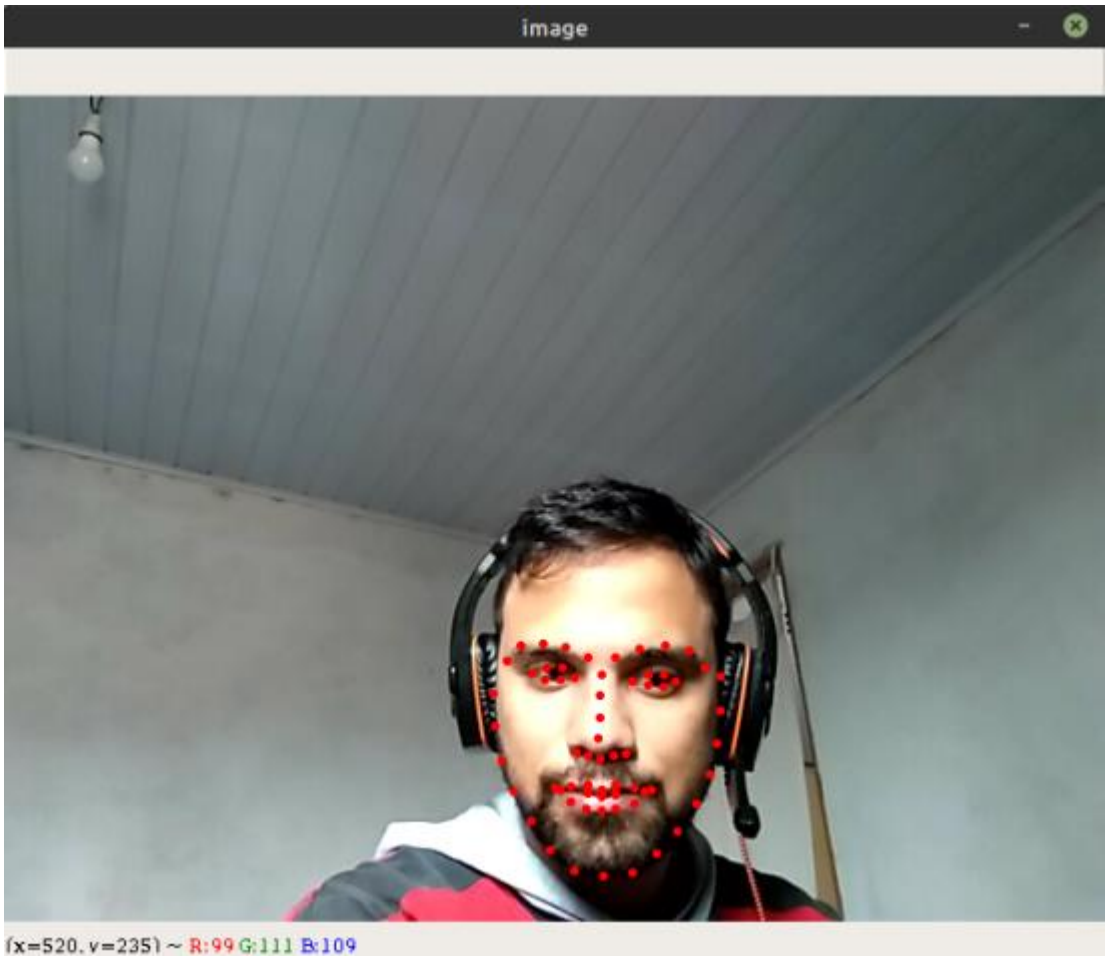
O programa Landmarks.py utiliza de um arquivo .dat (machine learning) como referência de análise dentro do algoritmo para identificar a face e assim diferenciar falsos positivos.

5 RESULTADOS E DISCUSSÃO

Durante o desenvolvimento do projeto encontramos algumas dificuldades: as bibliotecas de implementação da linguagem python estavam desatualizadas, impedindo o uso. Houve a tentativa de alterar o código pré-existente para que pudesse ser reutilizado, porém sem sucesso. A cada tentativa surgiam mais erros na programação (como: instalação de bibliotecas e execução) pois estavam dependendo de bibliotecas que não poderiam ser utilizadas. Para contornar esses problemas foram inseridos dois novos códigos (Landmarks.py e HandGesture.py) com o auxílio do professor e fontes já especificadas anteriormente, para atender as expectativas propostas ao grupo.

O código Landmarks.py é capaz de reconhecer faces, ao identificá-lo realiza a marcação com pontos em vermelho conforme Figura 03.

Figura 03: Exemplo Face (Alunos)

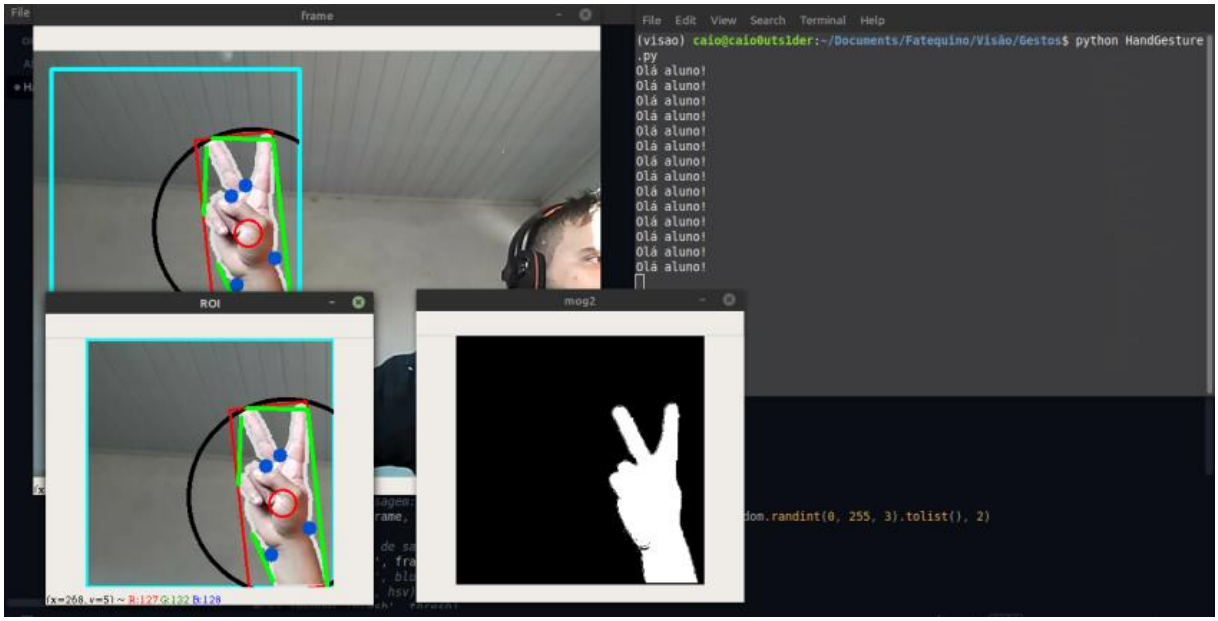


Fonte: Próprios autores

O código HandGesture é capaz de reconhecer gestos, ao localiza-lo o coloca em uma caixa na cor azul clara mostrando que está conseguindo identificar algo, em seguida faz uma marcação com contorno na cor verde com pontos na cor azul para especificar o gesto, por último realiza a segmentação de imagem deixando o gesto em branco com fundo preto. Caso seja retirada a mão, exibe uma mensagem de gesto não identificado (“No Hand Detected”).

Na Figura 04 é possível ver a interação do programa com o usuário imprimindo na tela a mensagem: “Olá Aluno!”.

Figura 04: Exemplo Gesto (Alunos)



Fonte: Próprios autores

6 CONSIDERAÇÕES FINAIS

Ao desenvolver o projeto pudemos notar que as implantações dos códigos Landmarks.py e HandGesture.py facilitarão e contribuirão para a conclusão do Fatequino.

Apesar dos empecilhos durante o desenvolvimento, podemos observar que os códigos farão o auxílio para futuros grupos pois eles são capazes de reconhecer gestos específicos e faces realizando a segmentação de imagem. Futuramente esses códigos farão parte da integração no Fatequino, sendo possível interagir com o aluno ou visitante.

Todos os códigos, leia-me, licenças, documentação, powerpoint e exemplos estão disponíveis com o professor responsável, no trello e github.

REFERÊNCIAS

SOBRAL, João Luís. Segmentação de Imagem. *In*: **Segmentação de Imagem.** [*S. l.*], 21 out. 2002. Disponível em: http://gec.di.uminho.pt/lesi/vpc0304/Aula07Segmenta%C3%A7%C3%A3o.pdf. Acesso em: 18 mar. 2020.

SALDANHA, Marcus S. F.; FREITAS, Drª Corina da Costa. Divisão de Processamento de Imagens. *In*: **Segmentação de Imagens Digitais: Uma Revisão**. [*S. l.*], 06 fev. 2009. Disponível em: http://mtc-m16c.sid.inpe.br/col/sid.inpe.br/mtc-m18@80/2010/06.22.18.13/doc/106003.pdf. Acesso em: 18 mar. 2020.

NIELSEN, Frank; NOCK, Richard. Transactions on Pattern Analysis and Machine Intelligence. *In*: **IEEE Transactions on Pattern Analysis and Machine Intelligence**. [*S. l.*], 20 set. 2004. Disponível em: https://ieeexplore.ieee.org/document/1335450. Acesso em: 18 mar. 2020.

GOOGLE Cloud Vision. *In*: **CONHEÇA A FANTÁSTICA GOOGLE CLOUD VISION API PARA IDENTIFICAÇÃO DE OBJETOS**. [*S. l.*], 06 jan. 2016.

Disponível em: https://mundoapi.com.br/materias/conheca-a-fantastica-google-cloud-vision-api-para-identificacao-de-objetos/. Acesso em: 18 mar. 2020.

JR, Valdir Stumm; DORNELES, Elias. ACESSANDO APIS REST COM PYTHON. *In*: **ACESSANDO APIS REST COM PYTHON**. [*S. l.*], 06 jan. 2016.

Disponível em: https://pythonhelp.wordpress.com/2014/07/25/acessando-apis-rest-com-python/. Acesso em: 20 maio 2020.

OPENCV. *In*: **OPENCV**. [*S. l.*], 10 fev. 2014.

Disponível em: https://pt.wikipedia.org/wiki/OpenCV. Acesso em: 20 maio 2020.

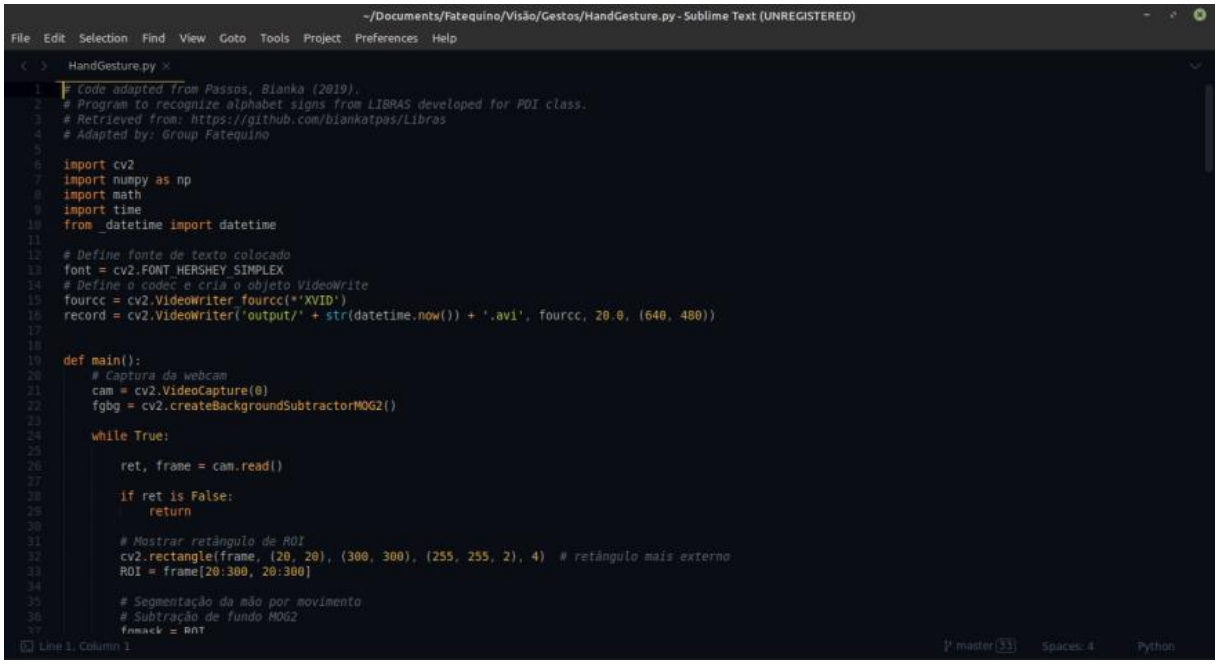
GENT, Paul Van. *In*: **EMOTION RECOGNITION USING FACIAL LANDMARKS, PYTHON, DLIB AND OPENCV.** [*S. l.*], 05 ago. 2016. Disponível em: http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/. Acesso em: 06 jun. 2020.

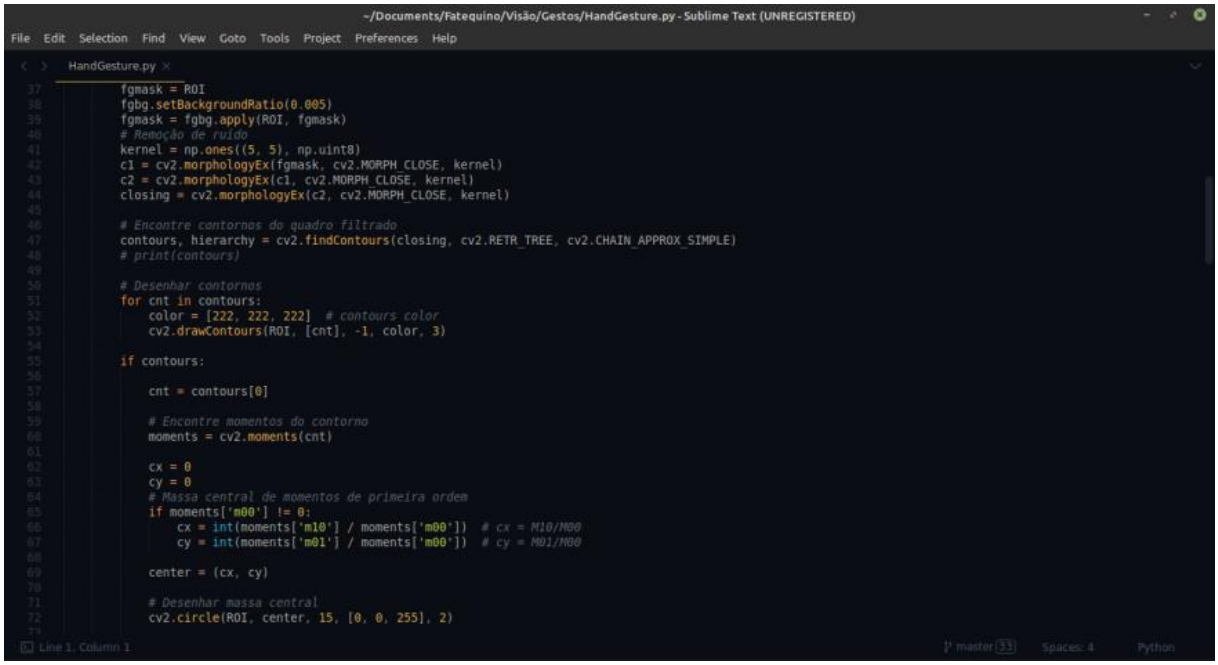
VIANA, Suzana. *In*: **CONFIGURANDO O AMBIENTE DLIB + PYTHON: GUIA PARA INICIANTES.** [*S. l.*], 13 jul. 2018. Disponível em: https://medium.com/@suzana.svm/configurando-o-ambiente-dlib-python-guia-para-iniciantes-81cdcffc937e**.** Acesso em: 06 de jun. 2020.

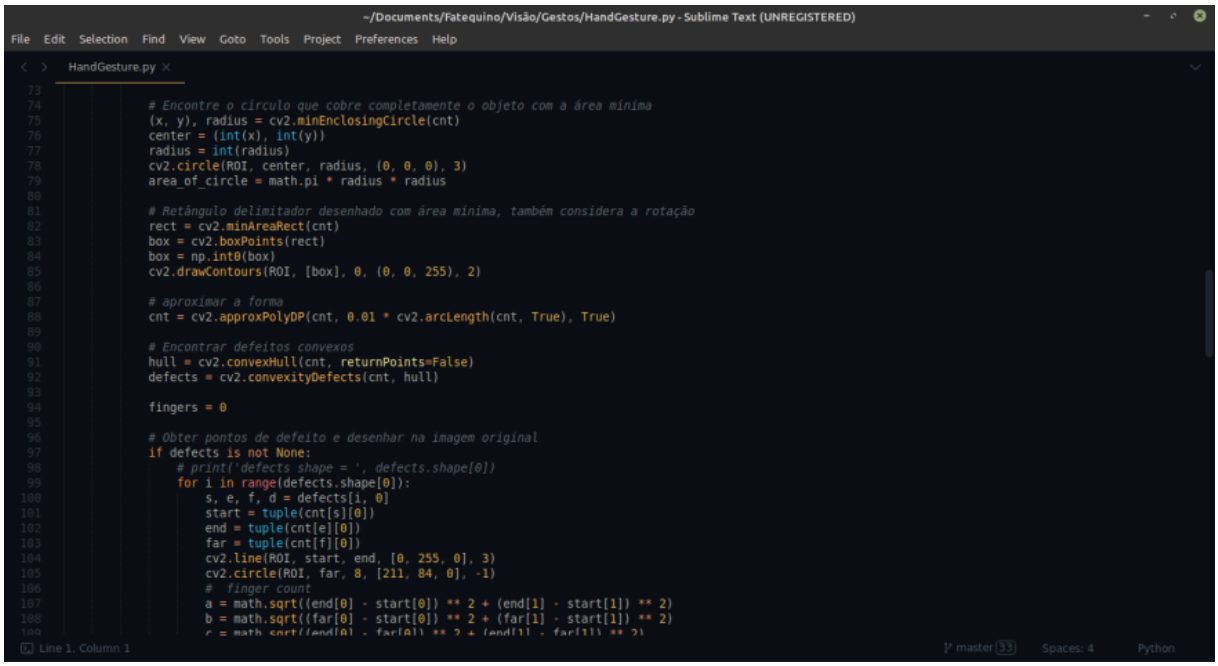
ANACONDA - Anaconda Inc. *In*: **Anaconda Individual Edition.** Disponível em: https://www.anaconda.com/products/individual. Acesso em: 20 de jun. 2020.

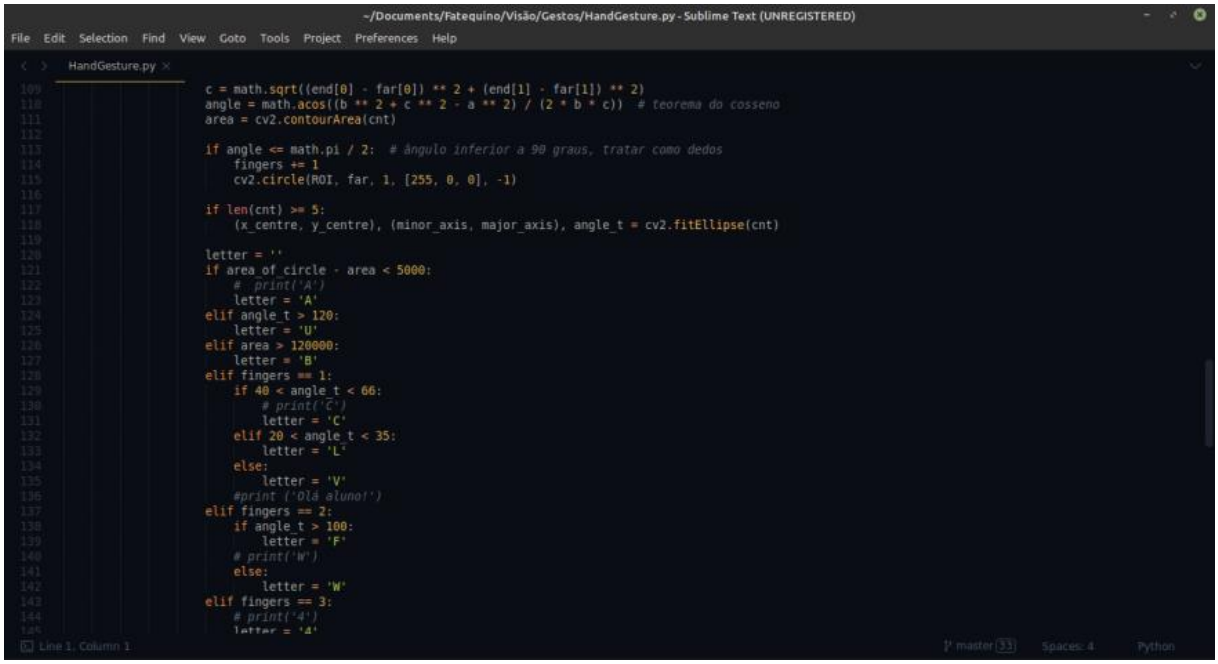
PYPI - Python Software Foundation. *In*: **The Python Package Index.** Disponível em: https://pypi.org/. Acesso em: 20 de jun. 2020.

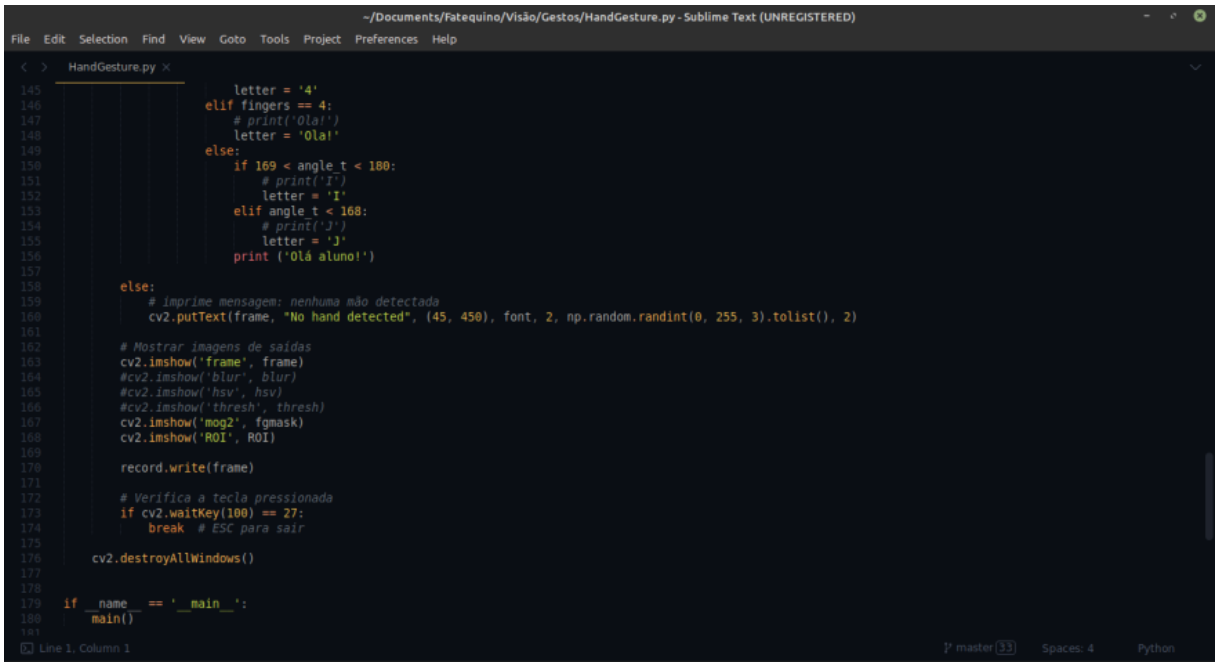
ANEXOS

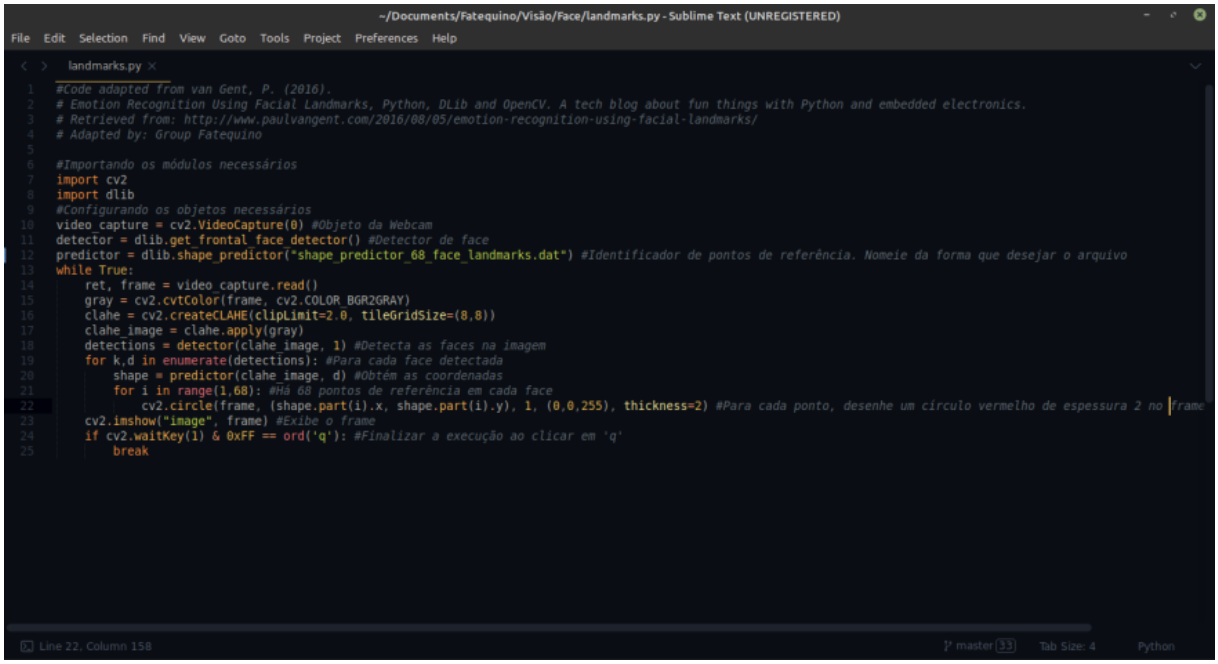

~/Documents/Fatequino/Visão/Gestos/HandGesture.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
HandGesture.py
# Code adapted from Passos, Bianka (2019).
# Program to recognize alphabet signs from LIBRAS developed for PDI class.
# Retrieved from: https://github.com/biankatpas/Libras
# Adapted by: Group Fatequino
import cv2
import numpy as np
import math
import time
from _datetime import datetime
# Define fonte de texto colocado
font = cv2.FONT_HERSHEY_SIMPLEX
# Define o codec e cria o objeto VideoWrite
fourcc = cv2.VideoWriter_fourcc(*'XVID')
record = cv2.VideoWriter('output/' + str(datetime.now()) + '.avi', fourcc, 20.0, (640, 480))
def main():
# Captura da webcam
cam = cv2.VideoCapture(0)
fgbg = cv2.createBackgroundSubtractorMOG2()
while True:
ret, frame = cam.read()
if ret is False:
return
Mostrar retângulo de ROI
cv2.rectangle(frame, (20, 20), (300, 300), (255, 255, 2), 4) # retângulo mais externo
ROI = frame[20:300, 20:300]
# Segmentação da mão por movimento
# Subtração de fundo MOG2
Line 1, Column 1
master (33)
Spaces: 4
Python


~/Documents/Fatequino/Visão/Gestos/HandGesture.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
HandGesture.py
fgmask = ROI
fgbg.setBackgroundRatio(0.005)
fgmask = fgbg.apply(ROI, fgmask)
# Remoção de ruído
kernel = np.ones((5, 5), np.uint8)
c1 = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
c2 = cv2.morphologyEx(c1, cv2.MORPH_CLOSE, kernel)
closing = cv2.morphologyEx(c2, cv2.MORPH_CLOSE, kernel)
# Encontre contornos do quadro filtrado
contours, hierarchy = cv2.findContours(closing, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# print(contours)
# Desenhar contornos
for cnt in contours:
color = [222, 222, 222] # contours color
cv2.drawContours(ROI, [cnt], -1, color, 3)
if contours:
cnt = contours[0]
# Encontre momentos do contorno
moments = cv2.moments(cnt)
cx = 0
cy = 0
# Massa central de momentos de primeira ordem
if moments['m00'] != 0:
cx = int(moments['m10'] / moments['m00']) # cx = M10/M00
cy = int(moments['m01'] / moments['m00']) # cy = M01/M00
center = (cx, cy)
# Desenhar massa central
cv2.circle(ROI, center, 15, [0, 0, 255], 2)
Line 1, Column 1
master (33)
Spaces: 4
Python


~/Documents/Fatequino/Visão/Gestos/HandGesture.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
HandGesture.py
Encontre o círculo que cobre completamente o objeto com a área mínima
(x, y), radius = cv2.minEnclosingCircle(cnt)
center = (int(x), int(y))
radius = int(radius)
cv2.circle(ROI, center, radius, (0, 0, 0), 3)
area_of_circle = math.pi * radius * radius
# Retângulo delimitador desenhado com área mínima, também considera a rotação
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
cv2.drawContours(ROI, [box], 0, (0, 0, 255), 2)
# aproximar a forma
cnt = cv2.approxPolyDP(cnt, 0.01 * cv2.arcLength(cnt, True), True)
# Encontrar defeitos convexos
hull = cv2.convexHull(cnt, returnPoints=False)
defects = cv2.convexityDefects(cnt, hull)
fingers = 0
# Obter pontos de defeito e desenhar na imagem original
if defects is not None:
# print('defects shape = ', defects.shape[0])
for i in range(defects.shape[0]):
s, e, f, d = defects[i, 0]
start = tuple(cnt[s][0])
end = tuple(cnt[e][0])
far = tuple(cnt[f][0])
cv2.line(ROI, start, end, [0, 255, 0], 3)
cv2.circle(ROI, far, 8, [211, 84, 0], -1)
# finger count
a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
Line 1, Column 1
master 33
Spaces: 4
Python


~/Documents/Fatequino/Visão/Gestos/HandGesture.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
HandGesture.py
c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
angle = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) # teorema do cosseno
area = cv2.contourArea(cnt)
if angle <= math.pi / 2: # ângulo inferior a 90 graus, tratar como dedos
fingers += 1
cv2.circle(ROI, far, 1, [255, 0, 0], -1)
if len(cnt) >= 5:
(x_centre, y_centre), (minor_axis, major_axis), angle_t = cv2.fitEllipse(cnt)
letter = ''
if area_of_circle - area < 5000:
# print('A')
letter = 'A'
elif angle_t > 120:
letter = 'U'
elif area > 120000:
letter = 'B'
elif fingers == 1:
if 40 < angle_t < 66:
# print('C')
letter = 'C'
elif 20 < angle_t < 35:
letter = 'L'
else:
letter = 'V'
#print ('Olá aluno!')
elif fingers == 2:
if angle_t > 100:
letter = 'F'
# print('W')
else:
letter = 'W'
elif fingers == 3:
# print('4')
Line 1, Column 1
master 33
Spaces: 4
Python


~/Documents/Fatequino/Visão/Gestos/HandGesture.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
HandGesture.py
letter = '4'
elif fingers == 4:
# print('Ola!')
letter = 'Ola!'
else:
if 169 < angle_t < 180:
# print('I')
letter = 'I'
elif angle_t < 168:
# print('J')
letter = 'J'
print ('Olá aluno!')
else:
# imprime mensagem: nenhuma mão detectada
cv2.putText(frame, "No hand detected", (45, 450), font, 2, np.random.randint(0, 255, 3).tolist(), 2)
# Mostrar imagens de saidas
cv2.imshow('frame', frame)
#cv2.imshow('blur', blur)
#cv2.imshow('hsv', hsv)
#cv2.imshow('thresh', thresh)
cv2.imshow('mog2', fgmask)
cv2.imshow('ROI', ROI)
record.write(frame)
# Verifica a tecla pressionada
if cv2.waitKey(100) == 27:
break # ESC para sair
cv2.destroyAllWindows()
if __name__ == '__main__':
main()
Line 1, Column 1
master 33
Spaces: 4
Python


~/Documents/Fatequino/Visão/Face/landmarks.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
landmarks.py
#Code adapted from van Gent, P. (2016).
Emotion Recognition Using Facial Landmarks, Python, DLib and OpenCV. A tech blog about fun things with Python and embedded electronics.
Retrieved from: http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/
Adapted by: Group Fatequino
#Importando os módulos necessários
import cv2
import dlib
#Configurando os objetos necessários
video_capture = cv2.VideoCapture(0) #Objeto da Webcam
detector = dlib.get_frontal_face_detector() #Detector de face
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat") #Identificador de pontos de referência. Nomeie da forma que desejar o arquivo
while True:
ret, frame = video_capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
clahe_image = clahe.apply(gray)
detections = detector(clahe_image, 1) #Detecta as faces na imagem
for k,d in enumerate(detections): #Para cada face detectada
shape = predictor(clahe_image, d) #Obtém as coordenadas
for i in range(1,68): #Há 68 pontos de referência em cada face
cv2.circle(frame, (shape.part(i).x, shape.part(i).y), 1, (0,0,255), thickness=2) #Para cada ponto, desenhe um circulo vermelho de espessura 2 no frame
cv2.imshow("image", frame) #Exibe o frame
if cv2.waitKey(1) & 0xFF == ord('q'): #Finalizar a execução ao clicar em 'q'
break
Line 22, Column 158
master 33
Tab Size: 4
Python