Programa de preparação de informação de aumento e aumento de imagens

Mário Esteves (up201607940@fe.up.pt), Ricardo Magalhães (up2015@fe.up.pt)

Faculdade de Engenharia da Universidade do Porto

1 Introdução

No âmbito da unidade curricular de Realidade Virtual e Aumentada foi-nos proposto o desenvolvimento de duas aplicações para a preparação de imagens com informação aumentada e para o aumento de imagens com informação obtida da aplicação anterior.
Este documento serve para descrever o funcionamento da aplicação, bem como as tecnologias utilizadas no desenvolvimento para alcançarmos os objetivos descritos.

2 Tecnologias utilizadas e estruturação dos programas

Para o desenvolvimento das aplicações, utilizamos a linguagem Python com a biblioteca OpenCV, devido à rapidez e facilidade de desenvolvimento que a linguagem proporciona e ,também, às ferramentas de deteção de pontos-chave de imagens que o OpenCV proporciona.
O programa está estruturado de uma forma extremamente simples. Existem 3 ficheros: Classes.py, Preparation.py e, finalmente, Augmentation.py. O ficheiro Classes.py contém todas as definições de classes comuns aos dois programas; o Preparation.py contém o programa de preparação de imagens; por fim, o ficheiro Augmentation.py define o programa de aumento de imagens.

3 Aplicações

Para obter os resultados esperados, devem ser utilizadas as aplicações na seguinte ordem: 1. aplicação de preparação de imagens e 2. aplicação de aumento de imagens. As seguintes secções são para delinear a utilização dos programas de preparação e aumento de imagens.

3.1 Preparação de Imagens

Para o programa de preparação de imagens, o procedimento de utilização é o seguinte:

1. Seleciona-se uma ou mais imagens de perfil frontal do objecto a aumentar;
2. Após isso, faz-se o aumento do objecto de acordo com os seguintes controlos:
   - O programa possui três modos de desenhos que podem ser alternados com a tecla 'm': seta, círculo e rectângulo. A espessura destas três formas podem ser alternatas clicando nos botões '1' (diminiuir) e '2' (aumentar);
   - No caso da seta, deve-se fazer um duplo clique para selecionar o ponto final da mesma, com direção vertical e sentido de cima para baixo;
   - No rectângulo e círculo, clica-se no botão esquerdo para começar a desenhar. Ao mover o rato, é desenhada uma pré-visualização para auxiliar o utilizador. Finalmente, assim que é largado o botão, a forma fica desenhada;
   - No caso do texto, simplesmente pressiona-se o botão direito, e começa-se a escrever. Ao pressionar a tecla 'tab' o texto fica gravado;
   - O botão 'backspace' desfaz o último desenho.
3. Para terminar a edição e gravar as informações de aumento do objecto pressiona-se 'S', senão se pretender gravar, pressiona-se 'Esc'. Caso sejam selecionadas mais do que uma imagem para preparar, o programa só fecha assim que a preparação ou cancelamento das mesmas esteja concluído.

Caso seja selecionada uma imagem de um objecto previamente preparado pela aplicação, a informações prévias são descartadas (caso existam), se as informações forem gravadas. Todas as imagens preparadas e as suas informações são guardadas num ficheiro binário, servindo de base de dados.

Todas as ações de desenho são possíveis com uso do *OpenCV*, sendo que a seleção dinâmica de imagens vem da biblioteca *Tkinter*. A escrita em ficheiro binários foi possibilitada pela biblioteca *Pickle*.

3.2 Aumento de imagens

Para o programa de aumento de imagens, simplesmente seleciona-se uma imagem a aumentar; escrevendo-se *debug* como argumento no terminal, irá correr nesse modo. Caso existam imagens de preparação na base de dados, encontram-se os *keypoints* e *descriptors* de cada imagem, bem com da imagem original, através do *SIFT*. Cada imagem é, depois, comparada com a original através de *matches*, possibilitando com o algoritmo de classificação *k-Nearest Neighbour*; as boas *matches* são encontradas com um teste de rácio, descrito por Lowe. As imagens com boas *matches* suficientes foram definidas como obtendo um mínimo de 60. Caso sejam encontradas mais do que umas potenciais imagens, é escolhida a melhor. Assim que esteja escolhida a melhor imagem, é realizado o mesmo procedimento escrito anteriormente, mas para cada pedaço da imagem com aumento. Aplica-se, também, homografia através dos pontos encontrados. Por fim, aplica-se uma *perspective transform* de forma a obter os pontos de cada aumento na imagem a aumentar. É, então, guardada a imagem aumentada.

4 Anexo

Classes.py

```
from enum import Enum

class Mode(Enum):
    ARROW = 1
    RECTANGLE = 2
    CIRCLE = 3
    TEXT = 4

class RectangularObj:
    def __init__(self, init, final, color, thickness):
        self.init = init
        self.final = final
        self.color = color
        self.thickness = thickness

class CircleObj:
    def __init__(self, init, dist, color, thickness):
        self.init = init
        self.dist = dist
        self.color = color
        self.thickness = thickness

class TextObj:
    def __init__(self, rect, text, font_size, color, thickness):
        self.rect = rect
        self.text = text
        self.font_size = font_size
        self.color = color
        self.thickness = thickness

class Subset:

    def __init__(self, mode, obj, img):
        self.mode = mode
        self.obj = obj
        self.img = img

class Image:

    def __init__(self, img, filename, subsets):
        self.img = img
```

```
        self.filename = filename
        self.subsets = subsets
```

Preparation.py

```
import cv2
import numpy as np
from tkinter.filedialog import askopenfilenames
from tkinter import Tk
import pickle
import os
from Classes import Mode, RectangularObj, CircleObj, TextObj, Subse

# Crop image functiion
def crop_image(img, x1, y1, x2, y2):
    if x1 < x2 and y1 < y2:
        return img[y1:y2, x1:x2].copy()
    elif x1 < x2 and y1 > y2:
        return img[y2:y1, x1:x2].copy()
    elif x1 > x2 and y1 < y2:
        return img[y1:y2, x2:x1].copy()
    elif x1 > x2 and y1 > y2:
        return img[y2:y1, x2:x2].copy()


\# Mouse callback function
def draw_circle(event, x, y, flags, param):
    global ix, iy, drawing, mode, imgcopy, img, font_size, imgList,

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
        imgcopy = img.copy()

    # Show the user the result while moving the mouse
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing is True:
            img = imgcopy.copy()
            if mode == Mode.RECTANGLE:
                cv2.rectangle(img, (ix, iy), (x, y), (0, 0, 255), t
            elif mode == Mode.CIRCLE:
                a = np.array((ix, iy))
                b = np.array((x, y))
                dist = np.linalg.norm(a - b)
```

```
            cv2.circle(img, (ix, iy), int(dist), (0, 255, 0), t

    # Handle double click for arrow
    elif event == cv2.EVENT_LBUTTONDBLCLK:
        if mode == Mode.ARROW:
            cv2.arrowedLine(img, (x, y - 50), (x, y), (0, 0, 255),
            subsets.append(Subset(Mode.ARROW,
                                  RectangularObj((x, y - 50),
                                                 (x, y),
                                                 (0, 0, 255),
                                                 thickness),
                                  crop_image(imgList[0], x - 50, y - 50, x

    # When the user stops drawing, handles the final result
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        if mode == Mode.RECTANGLE:
            cv2.rectangle(img, (ix, iy), (x, y), (0, 0, 255), thick
            subsets.append(Subset(mode,
                                  RectangularObj((ix, iy),
                                                 (x, y),
                                                 (0, 0, 255),
                                                 thickness),
                                  crop_image(imgList[0], ix, iy, x, y)))
        elif mode == Mode.CIRCLE:
            a = np.array((ix, iy))
            b = np.array((x, y))
            dist = np.linalg.norm(a - b)
            cv2.circle(img, (ix, iy), int(dist), (0, 0, 255), thick
            subsets.append(Subset(mode,
                                  CircleObj((ix, iy),
                                            int(dist),
                                            (0, 0, 255),
                                            thickness),
                                  crop_image(imgList[0], ix - int(dist), i
                                             ix + int(dist), iy + int(dist
        imgList.append(img.copy())

    # Handle right click button to write
    elif event == cv2.EVENT_RBUTTONDOWN:
        s = ""
        imgcopy = img.copy()
        while(1):
            k = cv2.waitKey(0) & 0xFF
            if k != 27:
```

```
                s += chr(k)
            print(s)
            if k == 9:
                text = cv2.getTextSize(s[:-1], cv2.FONT_HERSHEY_SIM
                                       font_size, 1)[0]
                ix, iy = text
                cv2.rectangle(img, (x, y), (x + ix, y - iy),
                              (255, 255, 255), -1)
                cv2.putText(img, s[:-1], (x, y),
                            cv2.FONT_HERSHEY_SIMPLEX, font_size,
                            255, 1)

                subsets.append(Subset(Mode.TEXT,
                               TextObj(RectangularObj((x, y),
                                                      (x + ix, y -
                                                      (255, 255, 25
                                                      -1),
                                       s[:-1], font_size, 255, 1),
                               crop_image(imgList[0],
                                          x - 20, y + 20,
                                          x + ix + 20, y - iy - 20)
                break
        imgList.append(img.copy())


# Open window and let user choose files
window = Tk()
files = askopenfilenames()
if(len(files) <= 0):
    print("Cancelled!")
    exit()
filenames = window.tk.splitlist(files)
window.withdraw()

if os.path.exists('imagesdb.obj') and os.path.getsize('imagesdb.obj
    images = pickle.load(open("imagesdb.obj", "rb"))
else:
    images = []


for filename in filenames:
    img = cv2.imread(filename)

    drawing = False  # true if mouse is pressed
    mode = Mode.ARROW
```

```
ix, iy = -1, -1
thickness = 5
font_size = 1
imgList = [img.copy()]
subsets = []

cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_circle)

while(1):
    cv2.imshow('image', img)
    k = cv2.waitKey(1) & 0xFF
    # Change mode
    if k == ord('m'):
        if mode.value < 3:
            mode = Mode(mode.value + 1)
        else:
            mode = Mode(1)
    # Change thickness
    elif k == ord('2'):
        thickness += 1
    elif k == ord('1'):
            thickness -= 1
    # Change font size
    elif k == ord('4'):
        font_size += 0.1
    elif k == ord('3'):
        font_size -= 0.1
    # Undo last action
    elif k == 8:
        if(len(imgList) > 1):
            imgList.pop()
            img = imgList[len(imgList) - 1]
            subsets.pop()
    # Cancel
    elif k == 27:
        cv2.destroyAllWindows()
        print('Preparation canceled for this image.')
        break
    # Save
    elif k == ord('s'):
        cv2.destroyAllWindows()
        print('Preparation done. Saving...')
        newfilename = os.path.splitext(os.path.basename(filenam
        cv2.imwrite(os.path.join('prepared', newfilename), img)
```

```
            if(any(image.filename == newfilename for image in image
                index = next((i for i, image in enumerate(images)
                              if image.filename == newfilename), -1)
                images[index] = Image(imgList[0], newfilename, subs
            else:
                images.append(Image(imgList[0], newfilename, subset
            break
# Save database
if len(images) > 0:
    pickle.dump(images, open("imagesdb.obj", "wb"))
    print('Saved images to database!')
else:
    print('No images to save. Exiting...')
```

Augmentation.py

```
import cv2
import numpy as np
from tkinter.filedialog import askopenfilename
from tkinter import Tk
import pickle
import os
import sys
from Classes import Mode, RectangularObj, CircleObj, TextObj, Subse


window = Tk()
filename = askopenfilename()
window.withdraw()

if len(sys.argv) > 1 and sys.argv[1] == 'debug':
    debug = True
else:
    debug = False

if os.path.exists('imagesdb.obj') and os.path.getsize('imagesdb.obj
    images = pickle.load(open("imagesdb.obj", "rb"))
else:
    exit()
    print("ERROR: Database not found. Exiting...")

# start match
img = cv2.imread(filename)
if debug:
```

```
    print('Image loaded.')

sift = cv2.xfeatures2d.SIFT_create()
bf = cv2.BFMatcher()

kp, des = sift.detectAndCompute(img, None)
if debug:
    print('Detected image keypoints and descripts from image.')

if debug:
    print('There are ', len(images), ' images on database.')
    print('Detecting images with enough good matches...')

good_images = images.copy()
# Detect images in db with enough good matches
for i, image in enumerate(images):
    image.kp, image.des = sift.detectAndCompute(image.img, None)
    image.matches = bf.knnMatch(image.des, des, k=2)

    num_rows, num_cols = image.des.shape

    image.good_matches = []
    for m, n in image.matches:
        if m.distance < 0.75 * n.distance:
            image.good_matches.append(m)
    if debug:
        print('Database image #', i, ' has ',
              len(image.good_matches), ' good matches.')
    if len(image.good_matches) <= 60:
        if debug:
            print('Database image #', i, ' doesn\'t have enough mat
        good_images.remove(image)
    else:
        if debug:
            print('Database image #', i, ' has enough good matches.

if len(good_images) <= 0:
    print("No images found. Exiting...")
    exit()

if debug:
    print('Found ', len(good_images), ' images with enough good mat
    print('Choosing the best one...')

# Choose the best one
```

```
bestIndex = -1

for i, image in enumerate(good_images):
    if bestIndex == -1:
        bestIndex = i
    elif len(image.good_matches) > len(good_images[bestIndex].good_
        bestIndex = i

image = good_images[bestIndex]
if debug:
    print('Found best image  ,', bestIndex)
    print('Detecting all keypoints, descriptors and good matches fr

# Detect keypoints, matches and good matches from subsets
for subset in image.subsets:
    subset.kp, subset.des = sift.detectAndCompute(subset.img, None)

    subset.matches = bf.knnMatch(subset.des, des, k=2)

    num_rows, num_cols = subset.des.shape

    subset.good_matches = []
    for m, n in subset.matches:
        if m.distance < 0.75 * n.distance:
            subset.good_matches.append(m)

if debug:
    print('Detecting all points from good matches...')

# Keypoints from good matches
for subset in image.subsets:
    subset.subpt = []
    subset.imgpt = []
    for good_match in subset.good_matches:
        subset.subpt.append(subset.kp[good_match.queryIdx].pt)
        subset.imgpt.append(kp[good_match.trainIdx].pt)

if debug:
    print('Removing subsets without enough points...')
# Remove subset without enough points
for subset in image.subsets:
    if len(subset.subpt) == 0 or len(subset.imgpt) == 0:
        image.subsets.remove(subset)

if debug:
```

```
    print('Applying homography between each subset and image...')
# Homography
for subset in image.subsets:
    subset.homography, mask = cv2.findHomography(np.asarray(subset.
                                                 np.asarray(subset.
                                                 cv2.RANSAC)
    num_rows, num_cols = subset.homography.shape
    if num_rows == 0 or num_cols == 0:
        image.subsets.remove(subset)

if debug:
    print('Getting corners in original image...')
# Get corner points in original cropped image
for subset in image.subsets:
    h, w, _ = subset.img.shape
    pts = np.float32([[0, 0], [0, h - 1],
                      [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
    subset.subcorners = pts

if debug:
    print('Applying perspective transform...')
# Apply perspective transform to get corners in image
for subset in image.subsets:
    subset.imgcorners = cv2.perspectiveTransform(subset.subcorners,
                                                 subset.homography)

if debug:
    print('Display started...')
# Display
for subset in image.subsets:
    if subset.mode == Mode.ARROW:
        ix = int(subset.imgcorners[0, 0][0])
        iy = int(subset.imgcorners[0, 0][1])
        w = int(subset.imgcorners[2, 0][0]) - ix
        h = int(subset.imgcorners[2, 0][1]) - iy

        cv2.arrowedLine(img, (ix + int(w / 2), iy),
                             (ix + int(w / 2), iy + int(h / 2) + 25
                             (0, 0, 255), subset.obj.thickness)
        if debug:
            print('Arrow displayed.')
    elif subset.mode == Mode.RECTANGLE:
        ix = int(subset.imgcorners[0, 0][0])
        iy = int(subset.imgcorners[0, 0][1])
        x = int(subset.imgcorners[2, 0][0])
```

```
            y = int(subset.imgcorners[2, 0][1])

            cv2.rectangle(img, (ix, iy), (x, y),
                                (0, 0, 255), subset.obj.thickness)
            if debug:
                print('Rectangle displayed.')

        elif subset.mode == Mode.TEXT:
            x = int(subset.imgcorners[1, 0][0]) + 20
            y = int(subset.imgcorners[1, 0][1]) - 20
            text = cv2.getTextSize(subset.obj.text, cv2.FONT_HERSHEY_SI
                                   subset.obj.font_size, 1)[0]
            ix, iy = text

            cv2.rectangle(img, (x, y), (x + ix, y - iy),
                                (255, 255, 255), -1)
            cv2.putText(img, subset.obj.text, (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, subset.obj.font_size,
                        255, 1)
            if debug:
                print('Text displayed.')

        elif subset.mode == Mode.CIRCLE:
            ix = int(subset.imgcorners[0, 0][0])
            iy = int(subset.imgcorners[0, 0][1])
            w = int(subset.imgcorners[2, 0][0]) - ix

            cv2.circle(img, (ix + int(w / 2), iy + int(w / 2)), int(w /
                               (0, 0, 255), subset.obj.thickness)
            if debug:
                print('Circle displayed.')

    if debug:
        print('Display finished!')


    cv2.namedWindow('image')
    cv2.imshow('image', img)

    newfilename = os.path.splitext(os.path.basename(filename))[0] + ".p
    cv2.imwrite(os.path.join('augmented', newfilename), img)
    print('Augmented image is on \'augmented\' folder.')
```