Python CV2 module

Opencv → ~~Com~~ Computer Vision (module)
→ open soure library for image processing.

loading an image → import cv2
img = cv2.imread('assets file_path, 1)
↑ loading type

load by default in → rbg pattern

type of loading (modes) → 1. CV2 IMREAD_COLOUR → G
→ load colour image. (-1)
(default) → Any transparency will be neglected
→ It is the default flag.

2. CV2. IMREAD_GRAYSCALE → in grayscale mode (0)

3. (1) or CV2. IMREAD_UNCHANGED → 1. load image such as including alpha channel
2. Also take care of transparency.

To show image → cv2.imshow('Image', img)
↑ name of image (any)

cv2.waitKey(0) → waits until you enter ~~ke~~ anything keyboard.
↑ infinite seconds → if not entered any key untill the time automatically skip this function.

→ to remove windows on desktop
cv2.destroyAllWindows() →

Resizing the images → img = cv2.resize(img), (400, 400)
↑ dimensions — 400 x 400 pixels

→ To resize by floating points →
⇒ img = cv2.resize(img, (0, 0), fx = 2, fy = 2)
↑ doubles the size

Rotating image → img = cv2.rotate(img, cv2.cv2.ROTATE

Writing an image → cv2.imwrite('name of file', sourceimage)

Copying one part of an image to another →

Image Representation

in computer → print(img variable)

→ opencv uses numpy.

→ It loads image pixels into numpy array.

→ img.shape → (995, width, channel)

↑ colour space

→ opencv represent image using (~~red~~, blue, green)

(blue, green, red)

black → white

raw → [[[0,0,0], [255,255,255]],

[],

]

~~vari~~ values are b/w → [0 → 255]

Accessing pixel values → print(img[0][5])

↑ one pixel

Changing pixel values → for i in range(100):

for j in range(img.shape

img[i][j] = [,,]

↑ value of pixel

cv2.imshow('Image', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

)

copying & pasting one part of an image ⇒ logo = img[500:700, 600:900]
must be same dimensions ← img[180:300, 700:1000] = logo

Camera loading ⇒

```
import numpy as np
import cv2
#loading a video
#capture device
cap = cv2.VideoCapture(0)
while True:
    #get frame from device
    ret, frame = cap.read()
    cv2.imshow('frame', frame)
    if cv2.waitKey(150) == ord('q'):
        break
#release the
#camera resource
cap.release()
cv2.destroyAllWindows()
```

(0) or filename of video (without camera)
↑ no. of camera on system

returns frame

return it to tell that camera is working fine or not. False if not working

milliseconds (returns ascii value of key what pressed)

(ASCII variable integer for ASCII)

1)∴ To turn this into 4-parts: ⇒

```
# put in while loop above
while True:
    ret, frame = cap.read()
    #pasting img in it
    image = np.zeros(frame.shape, np.uint8)
    (...)
```

unsigned int (8 bit)

gives width property

→ while True:

~~(...)~~

ret, frame = cap.read(); width = int(cap

height = int (cap.get(4)); image = np.zeros

Smaller_frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)

image[:~~width~~ height //2, :width //2] = smaller_frame

image[hight //2:, width //2:] = smaller_frame

image[:height //2, width //2:] —

[height //2:, width //2:] —

To rotate these frame:

image[:height //2, :width //2] = ~~smaller_frame[~~

cv2.ROTATE(smaller_frame, ROTATE ------)

← ---------- ; shapes should be same

Drawing → while True:

ret, frame = cap.read()

width = int(cap.get(3))

height = ---------- (4) ----

cv2.imshow('frame', frame)

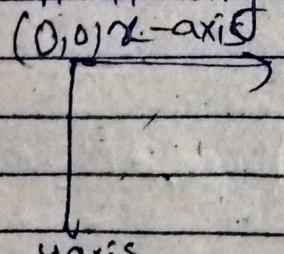
if cv2.waitKey(1) == ord('q'):

break

#drawing

img = cv2.line(frame, (0, 0), (width

starting & end coords.



cv2.imshow('frame', img)

(...)

Coordinate system in computer vision systems.

get(3));

(2.) Ex:- while True:
    (...)
    img = cv2.line(— —)
    img = cv2.line(img, (0,height)
        (width,0), (0,255,255), 5)
    img = cv2.rectangle(img, #topleft
    corner# (100,100), # (200,200),
    """ Colour """ (128, ---), -1)
    Top right
    ↑ fills (otherwise only thickness)
    img = cv2.circle(base image,
    """ center """ (300,300), 60, (0,0,255), -1)
    ↑ radius
    (...)

Drawing Text:- while True:
    (...)
    img = cv2.circle(— —)
#createfont
    font = cv2.FONT_ xxxx
    img = cv2.putText(img, 'Time is",
    (200,200), font, (0,0,0), 5,)
    ↑ bottom left corner
    font scale (int) (size of the font (default))
    thickness (optional) (can't (-1))
    line type (optional)
    (...)

, height)), (255,0,0), 10)
Colour ↑   Thickness ↑

HSV → Hue and saturation and light
colour scheme

[[( )]]

Colours →

```
(...)
while True:
    (...)
    height = int
    hsv = cv2.cvtColour(frame, cv2.COLOUR_BGR2HSV)
    cv2.imshow('image', hsv)
# To extract colours
    lower_blue = np.array([[ ]])      (colour values)
    upper_blue = np.array([ ])
# part of an image (pixel in range are 1)
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
```

→ Return a portion of an image

```
result = cv2.bitwise_and(frame, frame, mask=mask)
```

(frame → source, second frame → second source)

```
cv2.imshow('frame', result)
(...)
```

Corner detection →

Shi-Tomasi corner detection Algorithm → it is used for corner detection.

Drawing corners →

```
import cv2
img = cv2.imread("file path")
img = cv2.resize(img, (0,0), fx=0.75
cv2.imshow('frame', img)
# These algorithms work best on greyscale images
gray_img = cv2.cvtColour(img, cv2.COLOUR_BGR2GRAY)
```

(by default these are floating point values)

```
corners = cv2.goodFeaturesToTrack(img, 100,
0.01,          10          )
# quality of corners → [0-1]
# convert these corners into
# string
corners = np.int0(corners)
for corner in corners:
    print(corner)
    print = corn
    x, y = corner.ravel()
    cv2.circ
```

⇒ minimum euclidien distance between corners

$\Rightarrow \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

[[[0,1,2]]]

flatten it → [0,1,2]

returns only these elements

```
    cv2.circle(image, (x,y), 5,
      (0,0,0), -1)
← cv2.imshow('frame', img)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
```

Object detection → (Template matching)

Ex:

⇒ Detecting a football ⇒

```
⇒ import numpy as np
  import cv2
    img = cv2.imread("filename soccer.png",0)
    template = cv2.imread("filename ball.png",0)
    img = img.copy()
  h, w = template.shape
#grayscale image is two dimensional array
```

methods of template matching
copy from video

methods = [CV2.TM_CCOEFF ----]

```
for method in methods:
    img2 = img.copy()
    result = cv2.matchTemplate(img2, template)
```

do "convalusion" - taking template image and sliding it on base image and matches it.

returns in shape (W - w+1, H - h+1)

W: width of base image; w: temp; H: height of base image

Ex: Base = 4x4 ; template = 2x2 → [3,3]

→ Base image = [[255, 255, 255, 255],
[255],
[ ],
[ ]]

template image = [[255, 255],
[255, 255]]

output array of match ⇒ [[1, 1, 1], ← for true
[ ],
[ ]]

(returns rectangle)

```
min_value, maxval, min_loc, max_loc = cv2.minMaxLoc(result)
print(min_loc, max_loc)
~~cv2.recto~~
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    location = min_loc
else:
    location = max_loc
```

bottom-right = (location[0] + width, location[1] + height)
cv2.rectangle(img2, location, bottom-right, (255, 0, 0), 5)
cv2.imshow('match', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()

---

live face and eye detection → (use Haar Cascade classifier → finds a specific feature in an image)

import numpy as np
import cv2
cap = cv2.VideoCapture(0)
#loading classifiers
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cascade.eye
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,
    for (x, y, w, h) in faces:

1.3, 5) ← minNeighbours (Accuracy)
↑ scaleFactor of image
30% → shrink

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 5)
roi_gray = gray[y:y+h, x:x+w]
roi_color = frame[y:y+h, x:x+w]
eyes = eye_cascade.detectMultiScale(roi_gray
1.3, 8)

for (ex, ey, ew, eh) in eyes:
    cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

cv2.imshow('frame', frame)
if(cv2.waitKey(...)