Section 1

GrayScale → white → Strong intensity
→ black → weak intensity

object detection → location of object
object Recognition → object in image, position
object Classification → Category of object
object Segmentation → pixels belog to object
} open CV job

Python → Interpreted (parse line-by-line) Ex-1
→ dynamically Typed → no define data type
var = "hello"
→ Strongly typed
JS → 1 + "2" = "12" Implicit Conversion
python → 1 + "2" = error ✗ Implicit

$l_1 = [1,2,3]$
$l_1 * 2 = [1,2,3,1,2,3]$

Install numpy
① Python -m pip install --upgrade pip
② pip install numpy

numpy
① array = numpy.array([[1,2,3],[4,5,6]]) ⇒ 2 row, 3 Cols
1 2 3
4 5 6

② array values from 0 → 11 3 row 4 Cols

arr = numpy.arrange(12).reshape(3,4)
0 → 11 rows Cols

Image channel
1 → grayScale
3 → RGB

arr.size → num of items in array
arr.shape → (row, col)
arr.ndim → how many dimensions
arr.dtype.name → type of variable in array
arr.itemSize → one array element size in bytes

openCV-python → main modules
openCV-Contrib-python → Full package

Img of Zeros (3X3) = img = numpy.zeros(3,3, type=np.uint8)
Convert Img to RGB = CV_2.cvtColor(grayscale img, CV_2.Color_Gray2BGR)

Img.shape → row, Coloumn, number of Channels
RGB → 3
Grayscale → مش هترجع حاجة
لأنه الـ default = 1

Imread() → load img from specified file
CV2.imread(filename, mode of img read)

CV2.ImRead_Color → 3 Channel RGB
" " _GrayScale → 8 bit GrayScale
_AnyColor → 8 bit / Channel ← RGB / GScale
_unChanged → read all img data include alpha

imShow → Show img
CV2.waitKey(num of seconds) → عشان الـ window تظهر
CV2.destroyAllWindows() → اقفل الـ window

/ /

Cont mode of read

① • Anydepth ⟶ grayscale + original bit depth
② • anydepth | • Imread_Color ⟶ RGB + Bit depth

③ IMRead_Reduced_GrayScale_2 ⟶ قلل حجم الصورة للنص
_GrayScale_4 ⟶ للربع
_GrayScale_8 ⟶ للثمن
_Color_8 ⟶ صورة ألوان قللها للثمن

Image writing

imwrite (Image name +extension , Variable) ⟶ Same directory

im_write(r' Path' , Variable)

byte ⟶ range ⟶ 0 ⟶ 255

Reshape

X = bytearray(os.urandom(100) ⟶ Array byte
Random قيمة 100 قيمة

.reshape(row, Col

3 3 3 ⟹ numpy.random.randint(Stat, end, Values)
عدد

بس numpy :

/ / Path

read video → cv2.VideoCapture( )
get fps → video.get(cv2.CAP_Prop_fps)
size → .get(cv2.CAP_prop_Frame_width),
" " " " highet

video ⟹ cv2.VideoWriter(name, cv2.VideoWriter(I,4,2,0), fps, size)

VideoWriter_fourcc

① I,4,2,0 → uncompressed YUV 4:2:0 .avi
② P,I,M,1 → mpeg 1 .avi
③ X,V,I,D → Limit size Mpeg_4 .avi
④ m,P,4,V → " " " " "
⑤ X,2,6,4 → " " " " "
⑥ T h e o → ogg Vorbis .ogv
⑦ F l v 1 → Flash video .FLV

افضل حسب الحجم

video
open Camera → cv2.VideoCapture(0)

SECTION NOTES COMPUTER VISION

TO TEST THE CODE BY YOUR SELF(Github Repo)

Numpy.array([[1,2,3], [4,5,6]])
create an array with 2 rows & 3 columns

Numpy.arrange(12).reshape(3,4)
Make an array of 12 elements (with values from 0 to 1) with 3 rows and 4 columns

---

Array.size : number of items in array
Array.shape : rows , cols
Array.ndim : how many dimensions
Array.dtype.name : type of variable in array
Arr.itemSize: size in bytes of one array element

Numpy.zeros(row , col , type=np.unit8)
Return an array of zeros with row X col Dimensions

Cv2.imread("IMAGE PATH")
To read an image

CHANGE COLOR MODE…

cv2.cvtColor(img , color Mode)

MODES

bgr to grayscale
cv2.COLOR_BGR2GRAY

binary
img must be gray first
(thresh, blackAndWhiteImage) = cv2.threshold(grayImage, 127, 255, cv2.THRESH_BINARY)

Commented [FEA1]: Lower range

Commented [FEA2]: Upper range

Bgr to rgb
cv2.COLOR_BGR2RGB

BGR to HSV
cv2.COLOR_BGR2HSV

split image into separated channels RED,Green , BLUE

```
B, G, R = cv2.split(originalImage)
cv2.imshow("blue", B)
cv2.imshow("Green", G)
cv2.imshow("red", R)
```

---

merge image in one image

```
m=cv2.merge((B, G, R))
cv2.imshow("merged", m)
```

---

split image into separated channels HUE , SATURATION , VALUE

```
hsvImage=cv2.cvtColor(originalImage, cv2.COLOR_BGR2HSV)
cv2.imshow("original", originalImage)
cv2.imshow("HSV", hsvImage)
H=hsvImage[:,:,0]
S=hsvImage[:,:,1]
V=hsvImage[:,:,2]
```

Show only one color of HSV image (lower , Upper) values of color is needed

```
lower = np.array([0, 100, 100])
upper = np.array([10, 255, 255])
Mask = cv2.inRange(hsvImage, lower, upper)
shape = cv2.bitwise_and(originalImage, originalImage, mask= Mask)
```

Resize an Image

```
half = cv2.resize(image, (0, 0), fx = 0.5, fy = 0.5)
bigger = cv2.resize(image, (1050, 1610))
stretch_near = cv2.resize(image, (780, 540),interpolation = cv2.INTER_NEAREST)
```

Commented [FEA3]: Width

Commented [FEA4]: Height

Commented [FEA5]: Get nearest neighbours

low pass filter

```
#average
blurImage = cv2.blur(image,(5,5))
cv2.imshow('average Image', blurImage)

#Gaussian Blur
Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)

# Median Blur
median = cv2.medianBlur(image, 5)
cv2.imshow('Median Blurring', median)

# Bilateral Blur
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral Blurring', bilateral)

#custom filter
2d-Array  ( kernel )  :
Smooth kernel  :القيم قريبه من بعضها و موجب
Sharp kernel  :القيم كلها سالب ماعدا الي في النص

cv2.filter2D(img, -1, kernel)          [using cv2]

[using Scipy]
From scipy import ndimage
sharp = ndimage.convolve(img, kernel)
```

Commented [FEA6]: Filter size

Commented [FEA7]: Filter size

Commented [FEA8]: SIGMA

Commented [FEA9]: threshold

Commented [FEA10]: Filter size

Commented [FEA11]: Sigma space

EDGE DETECTION:

Edge examples : step , ramp , roof , spike

#Custom Kernel (ignore zero crossing edge)

Kernel to be used → sum must be = 0 & odd 33 , 5X5

Steps :

1- Kernel
2- Blur using gaussian
3- Edge detection img = original image - blured image

#laplacian (ignore zero crossing edge)

laplacian = cv2.Laplacian(img, cv2.CV_64F)

Commented [FEA12]: Depth
-1 means same depth of original image

#sobel (ignore zero crossing edge)

sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)

Commented [FEA13]: X , y

sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)

magnitude = cv2.add(sobelx,sobely)

scharry (ignore zero crossing edge)

schx= cv2.Scharr(img, cv2.CV_64F, 1, 0)

Commented [FEA14]: X , y

schy= cv2.Scharr(img, cv2.CV_64F, 0, 1)

to use plot (اعرض كل الصور في صوره واحدة)

from matplotlib import pyplot as plt

plt.subplot(2,3,1)

// 2 row , 3 col , اول صوره (2.. لو عايز اجيب الي بعدها)

1	2	3
4	5	6

canny (no ignore zero crossing)

imcanny= cv2.Canny(img, 200, 300) // 200 low threshold // 300 high treshold

---

Contours

- can be explained as the curve joining all the continuous points along the boundary which are having the same color or intensity . for shape analysis , object detection , object recognition

- For accuracy … Image must be binary first (cv2.COLOR_BGR2GRAY)

- Get threshold to Detect Edges + Remove some noise
  ret , thresh = cv2.threshold(grayimage ,127,255,0)

Commented [FEA15]: Threshold
why 127 ? average from 0 to 255

Commented [FEA16]: Max level of pixels
as it gray → 0:255

Commented [FEA17]: Type of threshold

- Get contours:
  Contours is a list containing number of points , these points have hierarchy (information)

  contours , hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
  params:
  1- threshold
  2- mode used to return hierarchy (information) like a tree
  3- method of returning contours

- To return number of contours (as it is a list)
  len(contours) → return integar str(len(contours)) → return string

- To draw contours
  cv2.drawContours(img, listOfContours , -1 , colorOfContour , thickness)
  - -1 means : draw all contours

  If you want to draw specific number (range of contours) : len(contours)=10 use index from 0 to 9
  - Color of contour will be in BGR (255,0,0) : BLUE
  - Thickness : 2

IMPORTANT → Contours will be drawn if image is coloured (BGR) Image

→ Thresholding will only be applied on Gray Image

DETECTING LINES

- IMAGE MUST BE IN GRAY SCALE
- APPLY FILTER ex: CANNY EDGE DETECTOR
  imcanny= cv2.Canny(img, 200, 300)

- Use Hough Transform
  lines = cv2.HoughLinesP(edges, 1, np.pi / 180.0 , 20 , minLineLength , maxLineGap)
  ⇨ Will return an array of coordinates

  edges : which is the result from canny filter

  1 means: rho (steps in pixels) np.pi / 180.0 means : theta θ (steps in angle (radian))

  20 means threshold : hough transform use voting … if line is less than threshold → discard

  minLineLength : اقل طول للخط maxLineGap : gap between lines

  To draw lines : loop through lines array

  for x1, y1, x2, y2 in lines[0]:

  cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

DETECTING CIRCLES

- Apply median filter
  median = cv2.medianBlur(image, 5)

Commented [FEA18]: threshold

- Apply hough transform
  Circles = HoughCircles(image,method,dp,minDist,param1,param2,minRadius,MaxRaduis)

  Dp → resolution if = 1 means resolution of input = resolution of output
  method → method used by hough transform : cv2.HOUGH_GRADIENT
  MinDist → minimum distance between 2 centers of circles
  Param1= outer edge detector param2 = center detector

- Normalize circles
  Circles= np.unit16(np.round(circles)

- Draw Circles
  cv2.circle(img , (center of circle) , raduis , color , thickness)

DRAW GEOMETRIC SHAPES

1- to draw line use following formula

imread(img path , mode) mode = 0 → grayScale mode=1 → RGB

cv2.line(img,point1,point2,color,thickness)

point1→ start coordinates point2 → end coordinates

if img is grayscale => color of line will be white even changed

to make color of line affected make img in rgb mode

2- to draw Arrowed line use following formula

cv2.arrowedLine(img,point1,point2,color,thickness)

3- to draw rectangle use following formula

cv2.rectangle(img,point1,point2,color,thickness)

point1 is the vertex of the rectangle

point2 is the vertex opposite to point1

if last value (thickness) is = to -1 → rectangle will be filled with the color

4- to draw circle use following formula

cv2.circle(img,center,radius,color,thickness)

if last value is = to -1 → circle will be filled with the color

5-to draw ellipse use following formula

cv2.ellipse(img,center,axes,angle,startAngle,endAngle,color,thickness)

center is the center of the ellipse

axes is the half of the size of the ellipse main axes

angle is the ellipse rotation angle in degrees

startAngle is the starting angle of the elliptic arc in degrees

endAngle is the ending angle of the elliptic arc in degrees

6-To draw text string we use the Following Function :

cv2.putText(img, text, org, fontFace, fontScale, color, thickness)

img is the source image

text is the text string to be drawn

org is the Bottom-left corner of the text strip

fontFace is the font type see #HersheyFonts ➔ cv2.FONT_HERSHEY_SIMPLEX

fontScale is the font scale factor that is multiplied by the font-specific base size

color is the circle color and thickness is the circle thickness and Negative values, like #FILLED

EXAMPLE

```
font = cv2.FONT_HERSHEY_SIMPLEX
org = (150, 350)
fontScale = 2
color = (0, 0, 0)
thickness = 2
img = cv2.putText(img, 'FADY', org, font, fontScale, color, thickness)
```

FEATURE EXTRACTION change in both values X , Y

Feature is a piece of information which is relevant for solving the computational task.

Types: Edges ,Corners , Blobs , Ridges.

- The regions in images which have maximum variation when moved (by a small amount) in all regions around it.
- So finding these image features is called **Feature Detection**.
- Computer also should describe the region around the feature (neighbours) so that it can find it in other images. So called description is called **Feature Description**.

HARRIS CORNER DETECTION

Determine which windows produce very large variations in intensity when moved in x and y direction.

With each such window found, a score R is computed. $\lambda 1 * \lambda 2 - k(\lambda 1 + \lambda 2)$

After applying a threshold to this score, important corners are selected& marked.

$\lambda 1 * \lambda 2 \rightarrow$ det(M) $\lambda 1, \lambda 2 \rightarrow Eigen\ Values$ $\lambda 1 + \lambda 2 \rightarrow trace(M)$

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

cv2.cornerHarris()

img : Input image, it should be grayscale and float32 type.

blockSize : It is the size of neighborhood considered for corner detection

ksize : Aperture parameter of Sobel derivative used (MATRIX USED).

k : Harris detector free parameter in the equation.

```
// change image to float32 Type
grayImage=np.float32(grayImage)
FINALIMAGE = cv2.cornerHarris(gray, 2, 3, 0.04)
// result is dilated for marking the corners, not important
FINALIMAGE = cv2.dilate(FINALIMAGE, None)
// get only important corners .. Threshold for an optimal value, it may vary depending on the image.
FINALIMAGE [FINALIMAGE >0.01* FINALIMAGE.max()]=[255, 0, 0]
```

⬆️ Get only edges > 0.01 and color them with BLUE in BGR MODE NOT IN RGB

SHI TOMASI CORNER DETECTION R= $\lambda 1$ لو عايز احدد عدد الكورنرز الي هتطلع

cv2.goodFeaturesToTrack(img, N, X, Y).

Img :image should be a grayscale image.

N: number of max corners.

X: CORNER QUALITY LEVEL (0-1).

Y: provide the minimum Euclidean distance between corners detected.

```
corners = cv2.goodFeaturesToTrack(gray,25,0.01,10)  // array of cooridantes of corners with float digits
corners = np.int0(corners)  // Convert to integars

for i in corners:
   x,y = i.ravel()                          // get center of each corner
   cv2.circle(img,(x,y),3,(255,0,0),-1)     // draw Filled BLUE circle  around each corner
```

CV SECTIONS 6-7

- An interest point (key point, salient point) detector is an algorithm that chooses points from an image based on some criterion. Typically, an interest point is a local maximum of some function, such as a "cornerness" = (R) metric.

- A descriptor is a vector of values, which somehow describes the image patch around an interest point. It could be as simple as the raw pixel values, or it could be more complicated, such as a histogram of gradient orientations.

- Together an interest point and its descriptor is usually called a local feature.

  Local features are used for many computer vision tasks, such as image registration, 3D reconstruction, object detection, and object recognition

- Harris, Min Eigen, and FAST are
  interest point detectors, or more specifically, corner detectors.

- SIFT includes both a detector and a descriptor.
  The detector is based on the difference-ofGaussians (DoG), which is an approximation of the Laplacian.
  The DoG detecto detect centers of blob-like structures.
  The SIFT descriptor is a based on a histogram of gradient orientations.

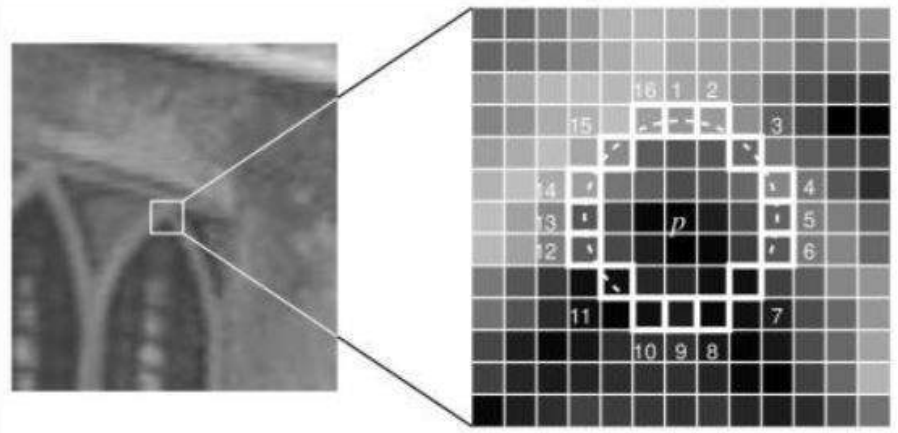
- SURF is meant to be a fast approximation of SIFT.

- BRISK, like SIFT and SURF, includes a detector and a descriptor.

  The detector is a corner detecto
  The descriptor is a binary string representing the signs of the difference between certain pairs of pixels around the interest point.

FAST ALGORITHM

1. Select a pixel p in the image which is to be identified as an interest point or not. Let its intensity be I_p.
2. Select appropriate threshold value t.
3. Consider a circle of 16 pixels around the pixel under test. (See the image below)



P is corner if BRIGHTER THAN Intensity of point + threshold (Ip + t) or DARKER THAN (ip - t)

IN MACHINE LEARINING PRESPECTIVE

- Select set of images to train
- Run FAST Algorithm
- For every feature point store 16-pixel around It as vector
- Do it for all images until get feature vector "P"
- EACH PIXEL HAVE ONE OF 3 STATES: DARK , Simillar , Bright
- Depend on these states → feature vector sub divided into 3 subset p(dark) , p(simillar) , p(Bright)
- Kp → boolean variable define that p is corner or not
- Do this for all images untill → Entropy = 0
- MAKE DECISION TREE

Non Maximum suppression

Detecting multiple interest points in adjacent locations is another problem. It is solved by using Non-maximum Suppression.

1. Compute a score function, V for all the detected feature points. V is the sum of absolute difference between p and 16 surrounding pixels values.
2. Consider two adjacent keypoints and compute their V values.
3. Discard the one with lower V value.

FAST ALGORITM CODE

```
// Initiat... FAST object... with default values
fast = cv.FastFeatureDetector_create()

// find the key point
kp = fast.detect(img, None)  // find
img2 = cv.drawKeypoints(img, kp, None, color=(255, 0, 0))  //draw   // 255,0,0 → BLUE
// drawKeypoint ( img , keypoint variable , NONE , BGR COLOR )


// get all parameters of FAST as  Threshold , Non-Maximum Suppression , neighbourhood
```

- fast.getThreshold() // return threeshold
- fast.getNonmaxSuppression() // return Non maximum suppression (Boolean)
- fast.getType() // return Neighbourhood

```
// Total Key Points without maximum suppression
len(kp)  // length of key point variable


// return image with  non maximum suppression
NewImage = ("New.jpg", img2)


// return Image without non-maximum Suppression
fast.setNonmaxSuppression(0)   // more corners
 kp = fast.detect(img, None)
```

SIFT (SCLAE INVARIANT Feature Transform) // uses LoG

Harris → rotation invariant // لا يتأثر بال روتيشن

Harris → scale variant // يتأثر بالتكبير و التصغير

⇨ SIFT IS FOUNDED FOR THIS PROBLEM

- Scale space extrema detection
- Key point localization
- Orientation assignment
- Key point descriptor // 16x16 → 4x4
- Key point matching

SIFT CODE

```
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

// CREATE SIFT OBJECT
sift = cv.SIFT_create()

// Get Key Point
kp = sift.detect(gray, None)

// Draw Key Point
img1 = cv.drawKeypoints(gray, kp, None)

علي حسب  حجم الفيتشر ارسم دايرة اكبر
img2 = cv.drawKeypoints(gray, kp, None,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

// return keypoint and descriptor
Kp , desc = Sift.detectAndCompute(img,None)
// return an array its size  = number of Keypoint X 128
```

SURF ALGORITHM SPEEDED UP ROBUST FEATURES // uses box filter + integral images

It is a speeded-up version of SIFT.
SURF goes a little further and approximates LoG with Box Filter.
Convolution with box filter can be easily calculated with the help of integral images.
And it can be done in parallel for different scales.

SURF CODE

```
Create surf object : surf= cv2.xfeatures2d.SURF_create(threshold)
Find keypoints using surf: kp= surf.detect(img)
Compute the descriptors with surf : des= surf.compute(img, kp)
Draw only keypoints : cv2.drawKeypoints(img, kp, None, color)
Number of features (KEY POINTS) =  len(kp)
Number of discriptors = surf. descriptorSize()
```

To Convert image into grayscale

img = cv2.imread('img path', cv2.IMREAD_GRAYSCALE) or img = cv2.imread("img path",0)

BRIEF (Binary Robust Independent Elementary Features)

- BRIEF is a faster method feature descriptor calculation and matching. It also provides high recognition rate unless there is large in-plane rotation.
- One important point is that BRIEF is a feature descriptor, it doesn't provide any method to find the features.
- So you will have to use any other feature detectors like SIFT, SURF etc.

BRIEF CODE

ϒ Create STAR detector : star = cv2.xfeatures2d.StarDetector_create()
ϒ find the keypoints : kp = star.detect(img,None)
ϒ Initiate BRIEF extractor : brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
ϒ compute the descriptors with BRIEF : kp, des = brief.compute(img, kp)
ϒ Draw only keypoints : cv2.drawKeypoints(img,kp,None)

ORB (Oriented FAST and Rotated BRIEF)

- **ORB: An efficient alternative to SIFT or SURF** .
- it is a good alternative to SIFT and SURF in computation cost, matching performance and mainly the patents.
- ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance.

ORB CODE

-Create ORB object : orb= cv2. ORB_create(max_num_feature) #500
- Find keypoints using orb: kp= orb.detect(img)
- Compute the descriptors with orb : des= orb.compute(img, kp)
-Draw only keypoints : cv2.drawKeypoints(img, kp, None, color)

how to align images automatically?

① Feature based alignment
- Find few matching features in both images
- Compute alignment

② Direct (pixel based) alignment
- Search for alignment where most pixels agree

↓

Brute Force Search
① - Define image matching function
SSD, Normalize Co-relation
② - Search over all params within reasonable range

loop in all pixels in X, y axis in 2 images
Compare Image, pixel with image$_2$ pixel

Problems
- Big (O) is high $O(N^8)$
- Not Clear for starting value initation and step

Solution
- use parallel search for set start vale, step
In special cases Big O = $O(N^4)$

Alternative
- use gradient decent on error function
- Motion Estimation

→ optical flow
- get motion of each pixel
- get motion of entire image

Motion estimation usages

① track object behavior ② Correct Camera jitter
⇒ ③ Align images (mosaics) ④ 3d reconstruction
⑤ Spatial effects

optical flow → motion of objects between Consecutive frame of sequence

① (dx, dy) ⟶ ② $(x+dx, y+dy)$ (after displacement)

Caused by: relative movment between object and Camera

① $I(x, y, t)$ intensity (space, time)

② Apply Taylor series approximation

③ divide to the Change

} Calculate of optical flow

Code

① lucas kanade method → track some point in video
② find points to be tracked
③ cv2.goodFeaturesToTrack() → find points to track
④ Capture first frame and detect corner points
⑤ These points will be tracked using luc as kanade

⟹

art

Lucas Kanade

Function used

cv2.CalcOpticalFlowPyrLK(① prevImg, ② nextImg, ③ prevPts, ④ [winSize, maxLevel, criteria])

Params

prevImg → First input image
nextImg → Second input image
prevPts → vector of 2d Points → flow needed to be found
winSize → Size of search window at each pyramid lvl

maxLevel → 0 based maximal pyramid level number
0 → Pyramid are not used (single level)
1 → 2 Pyramid levels are used

Criteria → specify the termination criteria of iterative search algorithm

Return

① nextPts
- output vector of 2d points
Contain
Calculated new
Position of input
Features in 2nd
image

② status
output status
vector
each element
of vector = 1
if flow of
Crossponding
Features are
found
else = 0

③ err
output vector of
errors
each element
in vector = error
if Crossponding
Feature
wasn't found

Some Code Snippets

cv2.VideoCapture(videoPath) → عشان اجيب الفيديو

Params for Cornerdetection →
dict (① max Corners ② quality level ③ min distance ④ block size)

Params for lucas kanade
dict(① WinSize (x,y) ② max level

random → np.random.randint( )

الـ Frames بتاعة الفيديو → تتحول → Gray (من أول مرة)

cv2.goodFeatureToTrack (img, mask = None, Params of Corner)

Dense optical flow

optical flow vector for every pixel of Frame
slow speed, more accurate

used in
① Video Segmentation
② Learning structure from motion

Dense optical flow has many implementation
but we will learn with
Farneback method

① approximate window of image by quadratic Polynomial with help of polynomial expansion

② observing polonomyal transform under State of motion

③ Dense optical flow

Need magnitude, direction → angle from 2d-Channel array

Calc optical flow Farne back ( )

Prev → First input image
next → Second input image
pyr_scale → image scale to build pyramid (Scale<1)

levels → (=1) → no extra layers on image
winSize → average window size
iteration → number of iterations

Poly_n → 5, 7 pixel-neighbourhood

Polysigma → standard deviation of gaussian deriv.

1.1 Poly = 5

1.5 Poly = 7

Pflow → Computed flow image

Flags

CV Section 9 Stero vision

(3d)

- Depth img → give you depth (added Z-axis)
- Intensity values in image represent distance of object from viewpoint
- You Can Color Code imgs to visually represent the Close, Far objects
- Depth maps Can be obtained using ① stero Camera, ② laser traingulation
- Depth maps used in 3D vision Algorithms

disParity maps → depth → gray scale images in which each pixel value is the stero disparity of surface

Concept of Sterovision

① two images ② shot from diffrent views
③ Result would be similar to be double
④ measure distance between pixels of same objects

Near object → grater stero than → far obj
brighter than

/ /

2 Normal Cameras → estimate relative distance to
(1) objects based on triangulation → from diffrent Camera prespective

⇓

Stero vision

(2) one normal Camera → move it over time to obtain diffrent prespective

⇓

Structure from motion

Code

Initial values كل ما أقل كل ما أحسن

(1) min Disparity → minimum possible disparity value ⇓ (-16)
(2) num Disparities → max disparity − min disparity
(3) block size → Window size, odd, 3-11
(4) P1 → disparity smoothing
(5) P2 → disparity smoothing }

أكبر كل ما كانت الصورة احسن

P_1 → gap between pixel and its neighbour (+1) (−1)
P_2 → " " " " " " $(1 < x)$
لو كانت ال distance أكبر من واحد

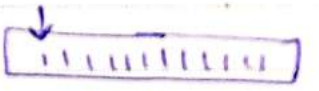
Function

(1) StereoSGBM_Create(
minDisparity
numDisparity
BlockSize
P_1
P_2)

SGBM → Semi Global Block Matching
↳ Compute disparity maps

(2) stero.Compute(img_1, img_2)

(3) cv2.imShow('Disparity img',
(disparity - mindisparity) / num disparities



Change BlockSize
Stereo.SetBlockSize(cv2.getTrackPos( ),
blocksize,
disparity