Sec 8 (CV)

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 Image₁ pixel with image₂ pixel

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

Solution
- use parallel search for set start value, 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

$\boxed{\text{optical flow}}$ → 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

$\boxed{\text{Code}}$

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 nextpts , [winsize, maxlevel, criteria])

Params

Previmge → 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 nexpts
• 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
Crosponding
Features are
found
else = 0

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

Some Code snippets

/ /

cv2.videoCapture (video path) → عشان اجيب الفيديو

Params for Cornerdetection →

dict (① max Conrners 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 في الفيديو → تكون → Gray (مش ألوان)

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
Faneback 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
↓
قيمة التغير

Calc opticalFlow Frane back ( )

Prev → First input image
next → Second input image
pyr_scale → image scale to build pyramid $\boxed{\text{Scale} < 1}$

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 Flags → Computed flow image