

IMP 301

Progress Test 3

Lê Đức Anh - Anhldhe150197 - AI1602

**Q1 (1.5pts) Notch reject filter.**

| Source code | **def** notch\_reject\_filter**(**shape**,** d0**=**9**,** u\_k**=**0**,** v\_k**=**0**):**  P**,** Q **=** shape  # Initialize filter with zeros  H **=** np**.**zeros**((**P**,** Q**))**  # Traverse through filter  **for** u **in** **range(**0**,** P**):**  **for** v **in** **range(**0**,** Q**):**  # Get euclidean distance from point D(u,v) to the center  D\_uv **=** np**.**sqrt**((**u **-** P **/** 2 **+** u\_k**)** **\*\*** 2 **+** **(**v **-** Q **/** 2 **+** v\_k**)** **\*\*** 2**)**  D\_muv **=** np**.**sqrt**((**u **-** P **/** 2 **-** u\_k**)** **\*\*** 2 **+** **(**v **-** Q **/** 2 **-** v\_k**)** **\*\*** 2**)**  **if** D\_uv **<=** d0 **or** D\_muv **<=** d0**:**  H**[**u**,** v**]** **=** 0.0  **else:**  H**[**u**,** v**]** **=** 1.0  **return** H |
| --- | --- |
| Input | Output |
|  | |

**Q2 (1.5pts) Median adaptive filter.**

| Source code | **def** Median\_ad\_filter**(**img**,**ksize**,**Smax**):**  m**,**n **=** img**.**shape  img\_filtered**=** np**.**zeros**([**m**,** n**])**    h **=** **(**Smax**-**1**)//**2    padded\_img **=** np**.**pad**(**img**,(**h**,**h**),**mode**=**'reflect'**)**  **for** i **in** **range(**m**):**  **for** j **in** **range(**n**):**  k **=** ksize  img\_k\_size **=** padded\_img**[**i**:**i**+**k**,**j**:**j**+**k**]**    **while** **True:**    A1 **=** np**.**median**(**img\_k\_size**)** **-** np**.min(**img\_k\_size**)**  A2 **=** np**.**median**(**img\_k\_size**)** **-** np**.max(**img\_k\_size**)**  **if** A1 **>** 0 **and** A2 **<**0**:**    B1 **=** **int(**img**[**i**,** j**])** **-** **int(**np**.min(**img\_k\_size**))**  B2 **=** **int(**img**[**i**,** j**])** **-** **int(**np**.max(**img\_k\_size**))**  **if** B1**>**0 **and** B2 **<**0**:**  img\_filtered**[**i**,**j**]** **=** img**[**i**,**j**]**  **else:**  img\_filtered**[**i**,** j**]** **=** np**.**median**(**img\_k\_size**)**  **break** #  **else:**  k **+=** 1  Snew **=** k**\***2**+**1  **if** Snew **<=** Smax **:**  img\_k\_size **=** padded\_img**[**i**:**i**+**k**,**j**:**j**+**k**]**  **else** **:**  img\_filtered**[**i**,**j**]** **=** np**.**median**(**img\_k\_size**)**  **break** #  **return** img\_filtered |
| --- | --- |
| Input | Output |
|  | |

**Q3 (1.5pts) Median Filter.**

| Source code | **def** median\_filter**(**data**,** filter\_size**):**  temp **=** **[]**  indexer **=** filter\_size **//** 2  data\_final **=** **[]**  data\_final **=** numpy**.**zeros**((len(**data**),len(**data**[**0**])))**  **for** i **in** **range(len(**data**)):**  **for** j **in** **range(len(**data**[**0**])):**  **for** z **in** **range(**filter\_size**):**  **if** i **+** z **-** indexer **<** 0 **or** i **+** z **-** indexer **>** **len(**data**)** **-** 1**:**  **for** c **in** **range(**filter\_size**):**  temp**.**append**(**0**)**  **else:**  **if** j **+** z **-** indexer **<** 0 **or** j **+** indexer **>** **len(**data**[**0**])** **-** 1**:**  temp**.**append**(**0**)**  **else:**  **for** k **in** **range(**filter\_size**):**  temp**.**append**(**data**[**i **+** z **-** indexer**][**j **+** k **-** indexer**])**  temp**.**sort**()**  data\_final**[**i**][**j**]** **=** temp**[len(**temp**)** **//** 2**]**  temp **=** **[]**  **return** data\_final |
| --- | --- |
| Input | Output |
|  | |

**Q4 (1.5pts) Motion deblurring .**

| Source code | **def** blur\_edge**(**img**,** d**=**31**):**  h**,** w **=** img**.**shape**[:**2**]**  img\_pad **=** cv2**.**copyMakeBorder**(**img**,** d**,** d**,** d**,** d**,** cv2**.**BORDER\_WRAP**)**  img\_blur **=** cv2**.**GaussianBlur**(**img\_pad**,** **(**2**\***d**+**1**,** 2**\***d**+**1**),** **-**1**)[**d**:-**d**,** d**:-**d**]**  y**,** x **=** np**.**indices**((**h**,** w**))**  dist **=** np**.**dstack**([**x**,** w**-**x**-**1**,** y**,** h**-**y**-**1**]).min(-**1**)**  w **=** np**.**minimum**(**np**.**float32**(**dist**)/**d**,** 1.0**)**  **return** img**\***w **+** img\_blur**\*(**1**-**w**)**  **def** kernel\_motion**(**angle**,** d**,** sz**=**65**):**  kern **=** np**.**ones**((**1**,** d**),** np**.**float32**)**  c**,** s **=** np**.**cos**(**angle**),** np**.**sin**(**angle**)**  A **=** np**.**float32**([[**c**,** **-**s**,** 0**],** **[**s**,** c**,** 0**]])**  sz2 **=** sz **//** 2  A**[:,**2**]** **=** **(**sz2**,** sz2**)** **-** np**.**dot**(**A**[:,:**2**],** **((**d**-**1**)\***0.5**,** 0**))**  kern **=** cv2**.**warpAffine**(**kern**,** A**,** **(**sz**,** sz**),** flags**=**cv2**.**INTER\_CUBIC**)**  **return** kern |
| --- | --- |
| Input | Output |
|  | |

**Q5 (1.5pts) Median Filter.**

| Source code | **def** calcPSF**(**size**,** R**):**    h **=** np**.**zeros**(**size**,** dtype**=**np**.**float32**)**  cv**.**circle**(**h**,(**size**[**1**]//**2**,**size**[**0**]//**2**),** R**,** 1**,** **-**1**)**  psf **=** h**/**np**.sum(**h**)**    **return** psf  **def** calcWnrFilter**(**psf**,** SNR**):**    h\_psf **=** np**.**fft**.**fftshift**(**psf**)**  h\_planes **=** **[**np**.**float32**(**h\_psf**),** np**.**zeros**(**h\_psf**.**shape**,** np**.**float32**)]**  h\_complexI **=** cv**.**merge**(**h\_planes**)**  h\_complexI **=** cv**.**dft**(**h\_complexI**)**  h\_planes **=** cv**.**split**(**h\_complexI**)**  denom **=** np**.**power**(**np**.abs(**h\_planes**[**0**]),**2**)** **+** **(**1**/**SNR**)**  wiener **=** np**.**divide**(**h\_planes**[**0**],** denom**,** dtype **=** np**.**float32**)**  **return** wiener  **def** filter2DFreq**(**img**,** wiener**):**    planes **=** **[**np**.**float32**(**img**),** np**.**zeros**(**img**.**shape**,** np**.**float32**)]**  complexI **=** cv**.**merge**(**planes**)**  complexI **=** np**.**divide**(**cv**.**dft**(**complexI**),** complexI**.**size**,** dtype **=** np**.**float32**)**    planesH **=** **[**np**.**float32**(**wiener**),** np**.**zeros**(**wiener**.**shape**,** np**.**float32**)]**  complexH **=** cv**.**merge**(**planesH**)**  complexIH **=** cv**.**mulSpectrums**(**complexI**,** complexH**,** 0**)**    complexIH **=** cv**.**idft**(**complexIH**)**  planes **=** cv**.**split**(**complexIH**)**  out **=** planes**[**0**]**    **return** out  **def** deBlur**(**img**,** R**,** SNR**):**    rows**,** cols **=** img**.**shape  m **=** cv**.**getOptimalDFTSize**(** rows **)**  n **=** cv**.**getOptimalDFTSize**(** cols **)**  img **=** **(**cv**.**copyMakeBorder**(**img**,** 0**,** m **-** rows**,** 0**,** n **-** cols**,** cv**.**BORDER\_CONSTANT**,** value**=[**0**,** 0**,** 0**])/**255**).**astype**(**np**.**float32**)**    h **=** calcPSF**((**m**,**n**),** R**)**  Hw **=** calcWnrFilter**(**h**,** SNR**)**  out **=** filter2DFreq**(**img**,** Hw**)**    **return** out |
| --- | --- |
| Input | Output |
|  | |

**Q6 (1.5pts) image anti-aliasing.**

| Source code | # convert to gray  gray **=** cv2**.**cvtColor**(**img**,** cv2**.**COLOR\_BGR2GRAY**)**  # threshold  thresh **=** cv2**.**threshold**(**gray**,** 32**,** 255**,** cv2**.**THRESH\_BINARY**)[**1**]**  # blur threshold image  blur **=** cv2**.**GaussianBlur**(**thresh**,** **(**0**,**0**),** sigmaX**=**3**,** sigmaY**=**3**,** borderType **=** cv2**.**BORDER\_DEFAULT**)**  # stretch so that 255 -> 255 and 127.5 -> 0  stretch **=** skimage**.**exposure**.**rescale\_intensity**(**blur**,** in\_range**=(**127.5**,**255**),** out\_range**=(**0**,**255**)).**astype**(**np**.**uint8**)**  # threshold again  thresh2 **=** cv2**.**threshold**(**stretch**,** 0**,** 255**,** cv2**.**THRESH\_BINARY**)[**1**]**  # get external contour  contours **=** cv2**.**findContours**(**thresh2**,** cv2**.**RETR\_EXTERNAL**,** cv2**.**CHAIN\_APPROX\_SIMPLE**)**  contours **=** contours**[**0**]** **if** **len(**contours**)** **==** 2 **else** contours**[**1**]**  big\_contour **=** **max(**contours**,** key**=**cv2**.**contourArea**)**  # draw white filled contour on black background  contour **=** np**.**zeros\_like**(**thresh**,** dtype**=**np**.**uint8**)**  cv2**.**drawContours**(**contour**,** **[**big\_contour**],** 0**,** 255**,** **-**1**)**  # dilate mask for dark border  kernel **=** cv2**.**getStructuringElement**(**cv2**.**MORPH\_ELLIPSE**,** **(**20**,**20**))**  mask **=** cv2**.**morphologyEx**(**contour**,** cv2**.**MORPH\_DILATE**,** kernel**)**  # create red colored background image  bckgrnd **=** np**.**full\_like**(**img**,** **(**0**,**0**,**255**),** dtype**=**np**.**uint8**)**  # apply mask to img  img\_masked **=** cv2**.**bitwise\_and**(**img**,** img**,** mask**=**mask**)**  # apply inverse mask to colored background image  bckgrnd\_masked **=** cv2**.**bitwise\_and**(**bckgrnd**,** bckgrnd**,** mask**=**255**-**mask**)**  # combine the two  result **=** cv2**.**add**(**img\_masked**,** bckgrnd\_masked**)**  # Creating the kernel with numpy  kernel2 **=** np**.**ones**((**5**,** 5**),** np**.**float32**)/**25    # Applying the filter  result**=** cv2**.**filt |
| --- | --- |
| Input | Output |
|  | |