HW1: Harris Corner Detection and Intro to Video Processing

Due date: See in Moodle.

Read the entire exercise before you solve it / submit it.

Opening remarks:

- We will run your code on a Linux machine. Specifically, our machine runs Ubuntu 16.04.
  - Your code needs to run on our machine without any changes to it.
  - If your PC does not run Ubuntu 16.04, we highly recommend you to use a Virtual Machine / use Dual Boot / use a docker with this OS installed on it.
  - See detailed running instructions at the end of this document.
- We use conda as our virtual environment and package management system.
  - We have supplied an **environment.yml** file that we will use to install the environment in which your code will run.
  - This environment includes everything that you need in order to solve this exercise.
  - Set up an environment using this file. Read the conda documentation to
  - If you decide that you need to install any other packages - ask for permission in the course forum. Permission requests via email will not be addressed.
- The goal of these exercises is for you to learn.
  - Feel free to use the internet as a resource for inspiration and a learning playground.
  - But, please, do not copy solutions of your friends / previous years. It is against the University's code of conduct and, on a personal note, it will hurt your ability to learn and understand the course material thoroughly.
- The goal of this exercise is to two-fold:
  - Implement your own Harris Corner Detector.
  - Implement your first Video Processing procedures.
- Open questions should be answered in a PDF file where the first line in this PDF is the IDs of the submitters.
- The python parts are "Fill your code here" kind of exercises. Most of the relevant information regarding what you need to implement is in the code.

Section 1 – Python Basic (0 points)

This part includes several simple assignments and its purpose is to make sure you grasp the basic operations in Python.

You won't be graded on this part, but you are expected to understand it. Failing to understand this part might lead to mistakes in the parts that are graded. There is no single answer for each question here. We expect you to be able to answer these questions. If this is not the case for you - consider taking an online python course as a prep for the assignments.

1. How can we select a portion of code and run it without running the rest of the code?
2. How is it possible to stop the code execution at a certain point? For example, before execution of a line of code that may raise an exception. Can we see the call stack and run commands with existing variables?
3. Write a short Python code to randomly generate a 5x6 matrix with values between 0 and 9. Add extra code lines to replace the 3rd smallest value in row #5 with the value 10.

Section 2: Harris Corner Detector :

Harris Corner Detector is a detector for corners. It is based on a response matrix which is obtained from the x and y gradient maps of the grayscale version of the image. A fancy Harris Corner Detector such as the one we will implement here, also involves Non-Maximal Suppression as we will see later on.

Our roadmap for implementing a Harris Corner Detector (HCD) is:

1. We will answer some questions to make sure we understand HCD.
2. We will implement three helper functions which will make our lives easier later on:
    a. a method which converts BGR images to RGB images (will help us to overcome the fact that images which are read with *OpenCV*'s *imread* are in BGR format and *matplotlib*'s convention for *imshow* is RGB).
    b. a method which splits images into tiles. This method will help us to create tiles for the image on which we will implement a very basic NMS algorithm (keep only the maximal value).
    c. a method which rearranges the tiles back into one image (matrix).
3. We will implement a matrix to compute I_x and I_y for an image I.
4. We will create a method which uses the method from (3) and takes I_x and I_y to compute the basic response matrix R:

    $$R = \frac{\lambda_- \cdot \lambda_+}{\lambda_- + \lambda_+} \approx det(M) - k \cdot [trace(M)]^2 = S_{xx} \cdot S_{yy} - S_{xy}^2 - k \cdot (S_{xx} + S_{yy})^2$$

    where:

    $$S_{xx} = conv(I_x^2, g),\ S_{yy} = conv(I_y^2, g),\ S_{xy} = conv(I_x \cdot I_y, g)$$

    for $0.04 < k < 0.06$.

5. Then, at the katharsis step we will apply NMS on the response matrix R and threshold the result with some threshold hyperparameter. This will yield the final corner map. This map will have the same height and width of the original image. It will also be binary: 0 means no corner and 1 means corner in that pixel.

Coding in this Section is done in the **harris_corner_detector.py** file.

Part 2.1: Open questions (9 points)

In the first part, answer (**add your answer in a PDF file**) :

1. Is Harris corner detector invariant to translation? Yes/No? Explain.
2. Is Harris corner detector invariant to rotation? Yes/No? Explain.
3. Is Harris corner detector invariant to constant illumination ($I_{out} = a * I_{in} + b$) ? Yes/No? Explain.

Hint: When we say that the detector is invariant to X, we mean that: if we take an image I1 and apply translation/rotation/illumination (and get a new image I2), will the two images have the same corners?

Keep your answers short.

Part 2.2 – Harris Corner Detector Perps (1 points)

1. Fill in the code in: **bgr_image_to_rgb_image**:

bgr_image_to_rgb_image		Type and shape	Description
Input			
	bgr_image	np.ndarray of shape: (height, width, 3)	a BGR image.
Output:			
	rgb_image	np.ndarray of shape: (height, width, 3)	same image as bgr_image, but red is first and blue is last.

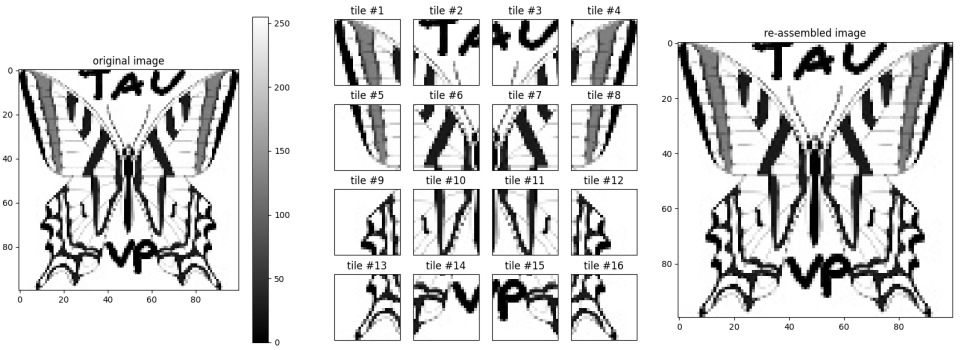
2. Fill in the code in: **black_and_white_image_to_tiles**

black_and_white_image_to_tiles		Type and shape	Description
Inputs			
	arr	np.ndarray of shape (h, w).	one channel image.
	nrows	the number of rows in each tile.	the number of rows in each tile.
	ncols	the number of columns in each tile.	
Output:			
	output	np.ndarray, shape: ((h//nrows) * (w //ncols) , nrows, ncols).	a tensor of non-overlapping nrowsXncols tiles.

3. Fill in the code in:**image_tiles_to_black_and_white_image**

image_tiles_to_black_and_white_image		Type and shape	Description
Inputs			
	arr	np.ndarray of shape (nTiles, nRows, nCols)	tiles tensor.
	h	the height of the original image.	the number of rows in each tile.
	w	the width of the original image.	
Output:			
	output	(h, w) np.ndarray	a reassembled version of the tiles back to the original image size and according to the correct order.

After you implement the methods from 2 and 3, you can validate your solution is correct by running the **main** function up to the **test_tiles_functions** method. The result should look like this:



Part 2.3 – Harris Corner Detector Basic Response Image (30 points)

1. Fill in the code in: **create_grad_x_and_grad_y.**

   This method receives the original image and calculates the gradients across the x and y-axes.

   Note that this method needs to support both RGB and grayscale images.

create_grad_x_and_grad_y		Type and shape	Description
Inputs			
	input_image	np.ndarray array of shape: (h, w, 3) or (h, w).	input image.
Output:			
	Ix, Iy	tuple (Ix, Iy). Ix is an np.ndarray of shape: (h, w). Iy is an np.ndarray of shape: (h, w).	Ix is the gradient across the x-axis and Iy is the gradient across the y-axis.

2. Fill in the code in: **calculate_response_image**.

   Note that this method needs to support both RGB and grayscale images.

   Note that this method uses the previous method.

   The response image is $R \approx det(M) \ - \ k \cdot [trace(M)]^2$

   where:

   $$det(M) \ = \ Sxx \cdot Syy \ - \ Sxy^2$$

   $$trace(M) \ = \ Sxx \ + \ Syy$$

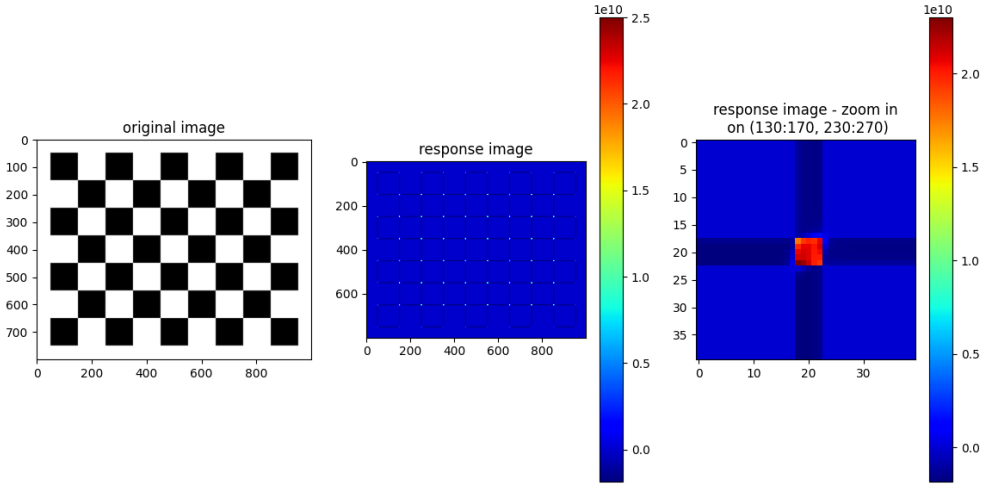
   $$Sxx \ = \ conv(I_x^{\ 2}, \ g)$$

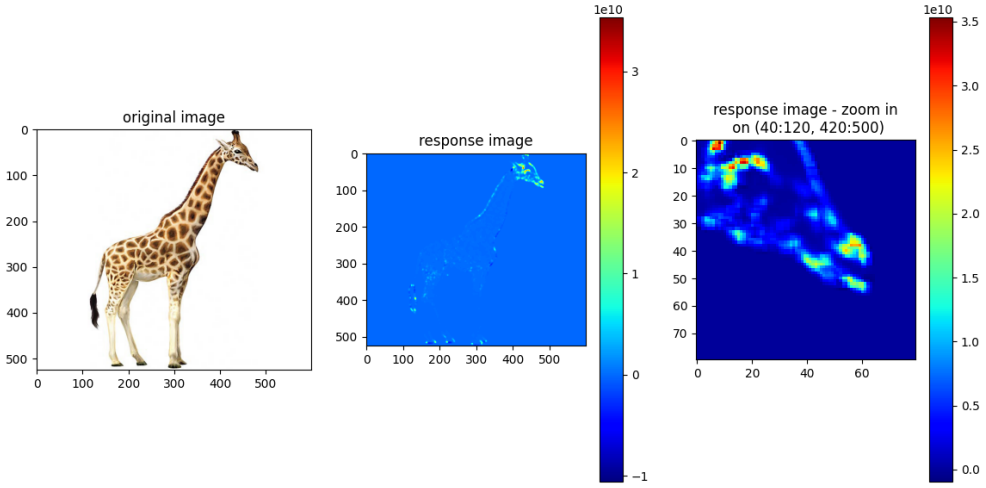
   $$Syy \ = \ conv(I_y^{\ 2}, \ g)$$

   $$Sxy \ = \ conv(I_x \cdot I_y, \ g)$$

calculate_response_image		Type and shape	Description
Inputs			
	input_image	np.ndarray array of shape: (h, w, 3) or (h, w).	tiles tensor.
	K	float	K: float. the K from the equation: $R \approx det(M) \ - \ k \cdot [trace(M)]^2$
Output:			
	response_image	np.ndarray of shape (h,w)	The response image is R: $R \approx det(M) \ - \ k \cdot [trace(M)]^2$

If you implemented both of these methods and the methods in the previous sections, then you should be able to run the code of the main script which calculates and prints the response image for the giraffe and checkerboard images. The result should look like this (see next page):





Note that my giraffe is yellow and not blue. Yours should also be yellow. Blue giraffes do not exist.

Part 2.4 – Our Harris Corner Detector (30 points)

1. Fill in the code in: **our_harris_corner_detector**.

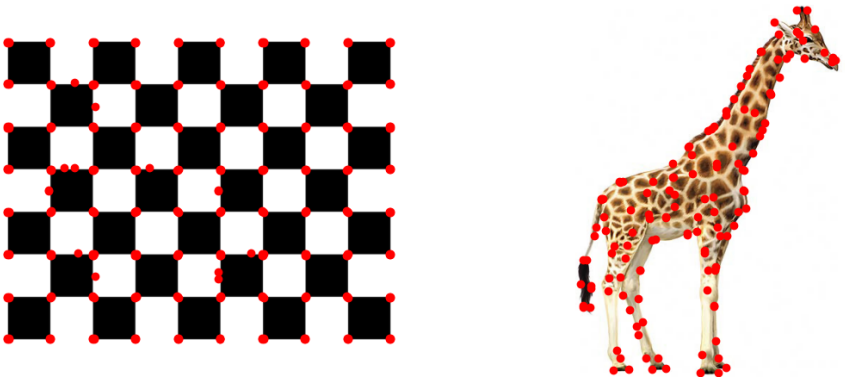
This method receives an input image, a parameter K and a threshold. The procedure it needs to implement is the following:

1. Calculate the response image from the input image and the parameter K.
2. Apply Non-Maximal Suppression per 25x25 tile:
    a. Convert the response image to tiles of 25x25. (use the method **black_and_white_image_to_tiles** from part 2.2).
    b. For each tile:
        i. create a new tile which is all zeros except from one value - the maximal value in that tile.
            1. Keep the maximal value in the same position it was in the original tile. **Hint:** use np.argmax to find the index of the largest response value in a tile and read about (and use): np.unravel_index.
3. Convert the result tiles-tensor back to an image. Use the method: **image_tiles_to_black_and_white_image** from part 2.2.
4. Create a zeros matrix of the shape of the original image, and place ones where the image from (3) is larger than the threshold.

our_harris_corner_detector		Type and shape	Description
Inputs			
	input_image	np.ndarray array of shape: (h, w, 3) or (h, w).	tiles tensor.
	K	float	K: float. the K from the equation: $R \approx det(M) \ - \ k \cdot [trace(M)]^2$
	threshold	float	minimal response value for a point to be detected as a corner.
Output:			
	output_image	np.ndarray of shape (h,w)	np.ndarray with the height and width of the input image. This should be a binary image with all zeros except from ones in pixels with corners.

Now you can run the full **main** method to enjoy your own implementation of the Harris Corner Detector.

Example results are in the following page. Note that the results are not presented for optimized parameters of K and threshold (try to answer - how do we know that they're not optimized?):



2. Run the **main** script with the ***to_save*** flag set to True. Make sure your script generates all images correctly. **Your image should have optimized parameters. Change K and / or threshold to get perfect results.** Submit the **harris_corner_detector.py** with the optimized K and threshold and with the ***to_save*** flag set to True.

<u>Remarks for the Harris Corner Detector Section:</u>

1. Make sure your IDs are found in the **harris_corner_detector.py** file.
2. You may ignore pixels on the very edge of the images (the 'frame' pixels).
3. You are NOT ALLOWED to use ready-made functions off the internet (or library functions that implement Harris corner detection algorithm).
4. For finding the derivatives in x and y you can apply a filter on the image or create a shifted copy of the image file (after converting to grayscale) and simply subtract in x and in y. Such that:

$$I_x = I - I_{shifted\ right} \quad , \quad I_y = I - I_{shifted\ down}$$

Section 3 – Video Manipulation Basics (30 points)

In this part you will open the attached video file (atrium.avi), conduct several operations on it and save the altered videos to new video files.

Load the video file using VideoCapture and save the new video files using VideoWriter. Read about those functions and classes in the OpenCV help or look it up on the internet.

The relevant file for this section is **basic_video_processing.py**.

This is also an "implement your code here" type of exercise. Fill in the code in the appropriate functions and delete the ***pass*** keyword from the python file.

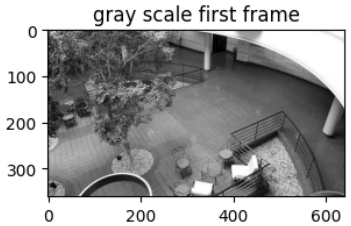
Note that we've supplied a helper function to extract video parameters from a VideoCapture object. Use this function when needed.

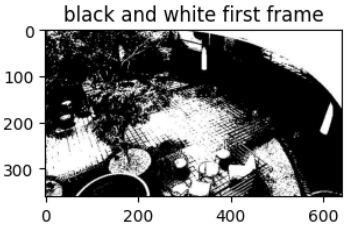
All functions read the same input video and save the output video with the same frame rate/ resolution/video size of the input video. Don't write the values hard coded, read the values from the input video using our supplied method.

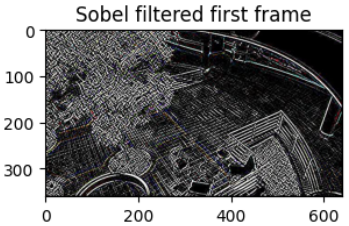
All functions have the same inputs: a path to the input video and the path of the output video. You cannot change the input / output video names.

1. Fill in the code in **convert_video_to_grayscale**. This method should load the original video, convert each frame to a grayscale image and save the new avi video file.
2. Fill in the code in **convert_video_to_black_and_white**. This method should load the original video, convert the video to black and white (not grayscale) and save it to a new avi video file. For each frame, the conversion from colour RGB to black and white has to be done using the global threshold level (Hint: Use *cv2.THRESH_OTSU)* and then set that threshold in the function cv2.threshold.
3. Describe the Sobel operator (use this resource) and explain how it may be used to obtain the edge map of each frame of the video. Put the answer to this question in the SAME pdf file of the answers to Section2/Part2.1.
4. Fill in the code in **convert_video_to_sobel**.
   Apply the Sobel operator on each frame as mentioned. Use cv2.Sobel on every frame, with the parameters: *ddepth=-1, dx=1, dy=1, ksize=5*.

The expected result of the first frame of each video in this section is:







Summary of files to submit

Section 1 – NOTHING to submit.

Section 2 – Submit the file **harris_corner_detector.py + 3** answers for Part 2.1 (in a PDF file with your IDs written inside the file in the first row).

Section 3 – Submit the file **basic_video_processing.py + 1** answer for question 3 in Section 3.

If your Python code fails to read my video files/images you will lose all the points for this question!

You will also need to submit **a single PDF** (**ex1_ID1_ID2.pdf)** containing the answers to the questions in Part 2.1 and Section3.

Any compilation errors of any sort equal zero points for that respective question. Please double check your solutions before submitting them!

ALL FILES WILL BE PLACED IN ONE ZIP FILE CALLED: **vp2020_ex1_ID1_ID2.zip** , where ID1&ID2 should be your ID's.

You have to include the supplied resources to the submitted ZIP file:

atrium.avi

butterfly.jpg

checkerboard.jpg

giraffe.jpg

Make sure to zip the files such that when extracting them - all PDF+py+avi+jpg files are all in the same folder. No subfolders are allowed. Don't zip .idea / .git / __pychace__ folders.

Only one of each student pair should submit the HW.

Good luck! ☺